Comparative Study of Deep Q Learning and Double Deep Q Learning on Atari Breakout

Vansh Porwal
Department of Industrial & Enterprise Systems Engineering
University of Illinois, Urbana-Champaign
vporwal3@illinois.edu

Abstract

The project aims to implement and compare the performance of Deep Q Learning (DQL) and Double Deep Q Learning (DDQL) algorithms on the Atari Breakout game. Both approaches will employ the concept of experience replay to enhance learning by reusing past experiences. Moreover, the exploration-exploitation dilemma will be addressed by testing epsilon-greedy action selection policy. Utilizing the OpenAI Gym as a testing environment and PyTorch for model implementation, this study seeks to investigate the effectiveness of these advanced reinforcement learning strategies in mastering a classic control problem. The training process will be accelerated using GPU resources on Google Cloud Platform (GCP), enabling a focus on the learning efficiency and the quality of the trained models. The comparative analysis will shed light on the algorithms' ability to navigate complex environments and improve over time with a systematic approach to experience and policy.

1 Introduction

Reinforcement Learning (RL) has emerged as a pivotal technique in the domain of artificial intelligence, particularly in solving complex decision-making problems. At the core of RL is the idea of learning optimal behaviors through interactions with an environment to maximize cumulative rewards. This methodology has shown remarkable success in various fields, including robotics, game playing, and autonomous systems.

A notable application of RL is in the realm of gaming, where it enables agents to learn and excel in diverse and complex environments. The Atari Breakout game, a classic arcade game, presents an ideal testbed for RL algorithms due to its straightforward yet challenging gameplay. In this game, an agent learns to break bricks with a ball by controlling a paddle, making it an excellent case for studying the efficacy of RL strategies.

This paper delves into a comparative study of two advanced RL algorithms: Deep Q Learning (DQL) and Double Deep Q Learning (DDQL). Both approaches, while grounded in the fundamental principles of Q-learning, exhibit distinct mechanisms in handling the learning process. We will explore how these methods perform in the Atari Breakout game, focusing on their learning efficiency, ability to balance exploration with exploitation, and overall effectiveness in mastering the game.

2 Reinforcement Learning Basics

2.1 Overview of Reinforcement Learning

Reinforcement Learning (RL) is a paradigm in machine learning where an agent learns to make decisions within an environment to achieve a goal. The process involves an agent interacting with an

Preprint. Under review.

environment represented by states s, beginning with an initial state s_0. The agent selects actions a according to a policy $\pi(s)$, which is a function dictating the action taken in state s. Actions lead to transitions between states according to a transition model $P(s'|s, a)$, which denotes the probability of moving to a new state s' from the current state s after taking action a. For each action taken, the environment provides a reward $r(s)$, based on a reward function. The agent's objective is to maximize the cumulative reward by learning the optimal policy.

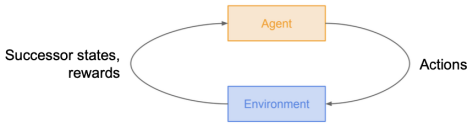


Figure 1: Reinforcement Learning Loop

The learning loop in RL includes:

- Observing the current state s,
- Taking an action a as determined by the policy $\pi(s)$,
- Receiving a subsequent state s' based on the transition model $P(s'|s, a)$ and reward $r(s')$,
- Updating the policy based on the observed transition and reward.

In this framework, the RL agent aims to learn the optimal policy π^* that maximizes the expected cumulative reward over time, often represented as the value function in RL algorithms.

2.2 Reinforcement Learning in Atari Games

In the domain of Atari games, RL agents are trained using the OpenAI Gym environment, which simulates each game as an episodic task with clear definitions of state, action, and reward:

- **State:** The state s_t is defined by the pixel values of the game screen at time t, which provides a complete visual representation of the game environment to the agent.
- **Actions:** The action set a_t includes discrete choices such as 'left', 'right', and 'fire'. The 'fire' action is particularly used to start a new game or continue after a life is lost.
- **Reward:** The reward r_t is given for game-specific achievements, such as breaking bricks in Breakout. The magnitude of the reward may depend on factors like the color of the brick, aligning the agent's objective with the game score.

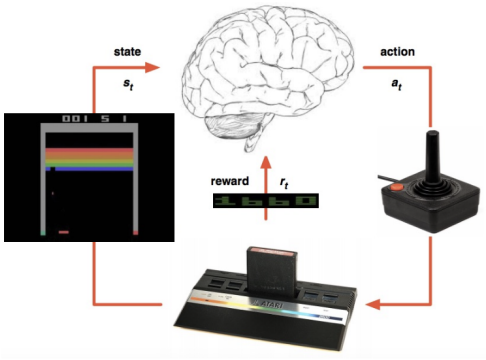


Figure 2: Reinforcement Learning in Atari Games

The OpenAI Gym provides a consistent interface for these elements, allowing researchers to focus on improving the RL algorithms' decision-making capabilities rather than the intricacies of game environment simulation.

2.3 Value Function and The Bellman Equation

The value function, denoted as $V^{\pi}(s)$, is a fundamental concept in reinforcement learning, representing the expected cumulative reward of following policy π from state s. Formally, it is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\tau}[r(\tau)|S_0 = s, \pi]$$

where τ denotes a trajectory starting at state s and following policy π, and $r(\tau)$ is the total reward obtained through the trajectory. The transition from state s to s' under action a is determined by the transition model $P(s'|s, a)$.

The optimal value function $V^*(s)$ represents the maximum value achievable from any state s under the best possible policy:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

The Bellman equation encapsulates a recursive decomposition of the value function:

$$V^*(s) = \max_{a} \left[r(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \right]$$

where $r(s, a)$ is the immediate reward after action a in state s, and γ is the discount factor for future rewards. This equation forms the basis for many algorithms to iteratively approximate the value function towards optimality.

2.4 Discounting of Rewards

Discounting is an approach used in reinforcement learning to handle the problem of infinite return sequences when the agent-environment interaction can continue indefinitely. It ensures the cumulative reward remains finite by defining the discounted cumulative reward $r(\tau)$ of a trajectory $\tau = (s_0, s_1, s_2, \ldots)$ as:

$$r(\tau) = r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \ldots = \sum_{t=0}^{\infty} \gamma^t r(s_t)$$

The discount factor γ, where $0 \leq \gamma \leq 1$, determines the present value of future rewards, with rewards received sooner being more valuable than those received later. This sum of discounted rewards is bounded by $\frac{r_{max}}{1-\gamma}$, assuming rewards are bounded by r_{max}. Such discounting is crucial for the convergence of learning algorithms and ensures that the agent's policy can be optimized effectively.

3 Introduction to Q Learning

Q-Learning is an off-policy algorithm in reinforcement learning that seeks to find the best action to take given the current state. It's about learning the action that maximizes the total reward of the agent. Unlike policy-based algorithms, Q-Learning focuses on learning the value of the action-value pair, denoted as Q-values. These Q-values indicate the quality of an action taken in a particular state, and the goal is to learn the optimal policy by learning these Q-values.

The relationship between value functions and Q-values is given by:

$$V^*(s) = \max_{a} Q^*(s, a)$$

The standard Bellman equation for value functions is:

$$V^*(s) = \max_{a} \left[r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V^*(s') \right]$$

The Bellman equation for Q-values is:

$$Q^*(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a'} Q^*(s',a') \right]$$

Q-Learning utilizes this Bellman equation to update Q-values iteratively, which guides the agent to make decisions that increase the cumulative reward.

4 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is an intersection of reinforcement learning (RL) and deep learning, bringing together the decision-making capabilities of the former with the powerful function approximation abilities of the latter. DRL utilizes deep neural networks to approximate value functions, policies, or models of the environment, enabling agents to learn optimal behaviors in high-dimensional, complex spaces. It has been responsible for significant breakthroughs across various domains, propelling RL applications to new heights.

4.1 Deep Q Networks (DQN)

Deep Q Networks (DQN) employ deep neural networks to approximate the action-value function, known as Q-values, in high-dimensional state spaces. These networks, parameterized by weights w, estimate the expected return $Q(s,a;w)$ for taking an action a in a given state s and following a policy π thereafter.

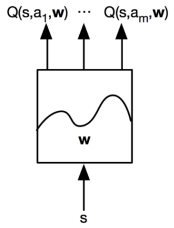


Figure 3: Deep Q-Learning model

The DQN aims to learn the optimal Q-values $Q^*(s,a)$ that maximize the expected reward:

$$Q^*(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') | s,a \right]$$

During training, the neural network parameters w are updated to align the predicted Q-values $Q_w(s,a)$, produced by the current network, with the target Q-values. The target Q-values are calculated as:

$$y_{\text{target}}(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r(s,a) + \gamma \max_{a'} Q_{\text{target}}(s',a') | s,a \right]$$

The loss function $L(w)$ is used to measure the discrepancy between $Q_w(s,a)$ and $y_{\text{target}}(s,a)$, which guides the updates to the weights w to reduce this difference:

$$L(w) = \mathbb{E}_{s,a} \left[\left(y_{\text{target}}(s,a) - Q_w(s,a) \right)^2 \right]$$

Gradient descent is then applied to minimize the loss function $L(w)$, updating the weights w of the neural network. The gradient of the loss function with respect to the weights is given by:

$$\nabla_w L(w) = \mathbb{E}_{s,a} \left[(y_{\text{target}}(s,a) - Q_w(s,a)) \nabla_w Q_w(s,a) \right]$$

This gradient update step is critical for the learning process, adjusting the weights in the direction that most reduces the loss, thus improving the policy over time.

Here, Q_{target} represents the Q-values as estimated by a separate, stable target network. This target network's weights are periodically synchronized with the weights of the primary network, w, to ensure stable convergence of the learning process.

4.2 Double-Deep Q Networks (DDQN)

Double Deep Q Networks (DDQN) address the overestimation bias inherent in the standard DQN algorithm. This bias arises because the max operator in Q-learning uses the same values to both select and evaluate an action. DDQN introduces a crucial modification: it selects the action using the current network (online network) but evaluates the action's value using the target network.

The target for DDQN is formulated as:

$$y_{\text{target}}(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r(s,a) + \gamma \max_{a'} Q_{\text{target}}(s',a') | s, a \right]$$

In this equation, Q_{target} is the Q-value estimated by the target network, which is used for evaluation, and Q_w is the value estimated by the current network, which is used for selecting actions. This separation mitigates the overestimation by providing a more unbiased estimate for the Q-values.

Double Deep Q-Networks (DDQN) offer several improvements over standard DQN:

- **Reduced Overestimation:** DDQN corrects the overestimation of action values by decoupling the action selection from its value evaluation, leading to more accurate Q-value estimates.
- **Stable Learning:** By using two networks, DDQN stabilizes the training process, as the less frequently updated target network provides a stable learning target.
- **Improved Convergence:** The separation of concerns in DDQN leads to more robust convergence properties and can prevent the value function from diverging.
- **Better Policy Evaluation:** DDQN often results in better policy evaluation, especially in environments with a high variance in rewards, contributing to more effective decision-making.

4.3 Deep Q Learning Implementation: Challenges and Solutions

Deep Q Learning (DQL) faces multiple challenges that can impact the stability and efficacy of the training process. These include:

- Non-stationary targets due to policy improvements over time.
- Correlation between successive experiences, violating the i.i.d. assumption standard in machine learning algorithms.
- Sensitivity to small changes in Q values, which can cause large shifts in the data distribution and policy.

Solutions such as the epsilon-greedy policy, experience replay, and freezing the target Q network have been developed to address these challenges.

4.3.1 Epsilon-greedy policy

An epsilon-greedy policy aids exploration and exploitation by selecting a random action with probability ϵ and the best-known action with probability $1 - \epsilon$. This approach alters the expectation from $\mathbb{E}_{s' \sim P(\cdot|s,a)}$ to $\mathbb{E}_{s,a \sim \rho(\cdot)}$, where ρ represents a distribution over states and actions that incorporates the epsilon-greedy decision-making process.

4.3.2 Experience Replay

Experience Replay is a fundamental technique in stabilizing Deep Q Learning. This method involves the following steps:

- Storing agent's experiences at each time step, $e_t = (s_t, a_t, r_t, s_{t+1})$, in a data set $\mathcal{D} = \{e_1, ..., e_t\}$, referred to as the replay buffer.
- Sampling random mini-batches from this buffer to break the correlation of sequential data, which can introduce bias and inefficiency in the learning process.
- Using these random samples to update the network weights as if they were new data, which provides a more diversified data set and promotes a more robust convergence.

This strategy mimics the random data sampling of i.i.d datasets commonly assumed in supervised learning tasks, thus significantly improving the learning outcomes in DQL algorithms.

4.3.3 Freeze target Q network

By freezing the target Q network for several iterations, we decouple the target from the rapid updates of the network, reducing the oscillations in the learning process and promoting stable convergence.

Each technique contributes to a more stable and robust DQL implementation, enhancing the agent's ability to learn and perform in complex environments.

4.4 Final Implementation of Deep-Q Learning

The algorithm below provides a high-level overview suitable for both Deep Q-learning (DQN) and Double Deep Q-learning (DDQN). While fundamentally similar, the key distinction in DDQN lies in its approach to updating Q-values, where it separates the action selection process from the action evaluation process.

Algorithm 1 Deep Q-learning

```
Prepare memory storage for learning experiences
Set up the primary Q-function with initial parameters
Set up the secondary Q-function for stable learning targets
for each learning episode do
    Initialize the starting state
    for each step of the episode do
        Choose an action based on a mix of exploration and best-known strategy
        Apply the action, observe the outcome and reward
        Store the experience for future learning
        Use a random set of past experiences to update the Q-function
        Periodically update the secondary Q-function for consistency
    end for
end for
```

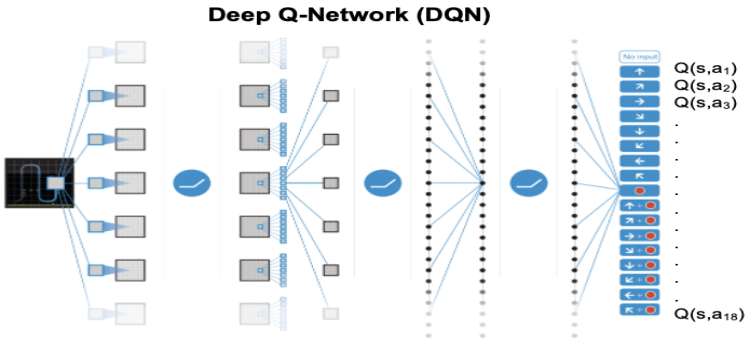
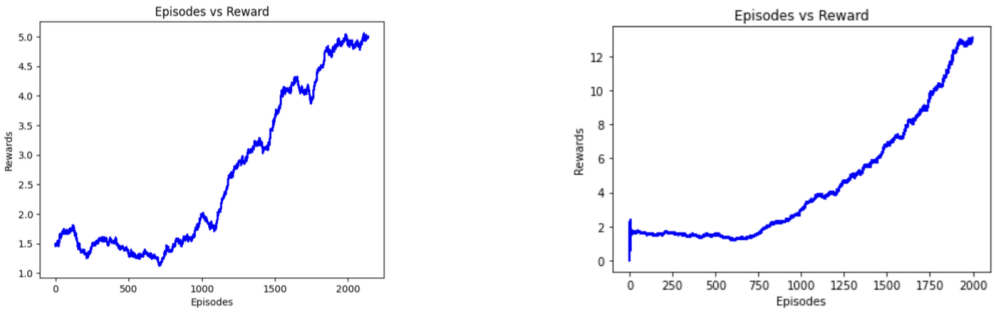


Figure 4: Deep Q Network for Atari Games

5 Results and Analysis

This section discusses the performance of the implemented models. The focus is on evaluating the learning efficacy over episodes. Each model's performance is analyzed based on the rewards accumulated over 100 episodes. The plots compare these rewards to illustrate the learning curve and overall effectiveness of the training process.



(a) Plot for DQN model

(b) Plot for DDQN model

Figure 5: Episodes vs Reward plots for the DQN and DDQN models

5.1 Learning Trends and Performance Metrics

The results depicted in the plots illustrate distinct learning trends for the DQN and DDQN models. The DQN model shows a steady increase in rewards as the number of episodes progresses, suggesting a robust learning process. However, the DDQN model demonstrates a more rapid improvement, achieving higher rewards in fewer episodes. This indicates that DDQN may be more efficient in learning optimal policies, likely due to its reduced overestimation bias. Specific performance metrics, such as the average reward over the last 100 episodes, further quantify these observations.

5.2 Stability Analysis

The stability of the learning process can be inferred from the smoothness of the reward curves. The DQN plot presents a gradual ascent with fewer fluctuations, indicating stable learning. In contrast, the DDQN plot, while achieving higher rewards, shows more variability, suggesting potential periods of instability. This could be due to DDQN's more complex learning dynamics, which may cause more frequent policy changes and affect stability.

5.3 Exploration vs Exploitation Analysis

Both models utilize an epsilon-greedy policy to balance exploration with exploitation. Initially, a higher degree of exploration is expected, with a gradual shift towards exploitation as the model learns. The increasing trend in rewards for both DQN and DDQN suggests an effective balance, with sufficient exploration to discover optimal actions and adequate exploitation to maximize rewards. The sharper increase in rewards for DDQN may indicate a more efficient exploitation phase, potentially due to a more accurate Q-value estimation.

5.4 Parameter Tuning Analysis

Parameter tuning plays a pivotal role in the performance of reinforcement learning algorithms. The fine-tuning of hyperparameters, such as the learning rate, discount factor, and the balance between exploration and exploitation, can significantly influence the learning efficiency and stability of the models. In our experiments, careful adjustment of these parameters was crucial in achieving the observed learning trends and stability in both DQN and DDQN models. This underscores the importance of methodical hyperparameter optimization in the field of deep reinforcement learning.

6 Conclusion

In conclusion, this project elucidates the comparative effectiveness of DQL and DDQL algorithms within the Atari Breakout game framework. The study reveals that DDQL's refined approach to Q-value estimation significantly enhances policy learning, leading to more efficient performance gains. Despite DQL's commendable stability, DDQL's proficiency in overcoming overestimation bias positions it as a superior technique for complex problem-solving in RL domains. These findings not only affirm the robustness of DDQL but also pave the way for future advancements in RL methodologies, potentially catalyzing further innovations in artificial intelligence applications.

The implication of these findings extends beyond gaming, offering potential applications in areas such as autonomous systems and robotics. Moreover, the scope for future research is expansive, with opportunities to refine these algorithms further and explore their applicability to even more complex, real-world tasks.

References

[1] V. Mnih et al., *Playing Atari with Deep Reinforcement Learning*, arXiv preprint arXiv:1312.5602, 2013. https://arxiv.org/pdf/1312.5602.pdf

[2] H. van Hasselt, A. Guez, D. Silver, *Deep Reinforcement Learning with Double Q-learning*, arXiv preprint arXiv:1509.06461, 2016. https://arxiv.org/pdf/1509.06461.pdf

[3] *Tutorial: Double Deep Q Learning with Dueling Network Architectures*, Towards Data Science. Available online: https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756

[4] *A Technical Introduction to Experience Replay for Off-policy Deep Reinforcement Learning*, Towards Data Science. Available online: https://towardsdatascience.com/a-technical-introduction-to-experience-replay-for-off-policy-deep-reinforcement-learning-9812bc920a96