REINFORCEMENT LEARNING FOR ATARI *Freeway*

Wenzheng Wu[*] **Panxin Tao**[†] **Zhengtao Han**[‡] **Runkang Yang**[§] **Yourong Cao**[§]
{wuwz,taopx,hanzt,yangrk,caoyr}2022@shanghaitech.edu.cn

ABSTRACT

Reinforcement learning (RL) has shown significant potential in solving complex decision-making problems across various domains. This paper explores the application of RL to the Atari *Freeway* game in which an agent must navigate a busy highway, avoiding collisions with vehicles while reaching its goal. Initially, we apply the Q-learning algorithm to learn an optimal policy through state-action value iteration. To address the limitations of Q-learning in high-dimensional state spaces, we extend the framework to Deep Q-Learning (DQN), which utilizes neural networks as function approximators and incorporates key improvements like experience replay and target networks to stabilize training. Additionally, motivated by the efficiency of Actor-Critic algorithm in environments with large action spaces and continuous learning, we analyze its potential for solving this problem. Experimental results demonstrate the efficacy of both Q-learning and DQN across varying state configurations, and we empirically find that *DQN significantly outperforms Q-learning in environments with high-dimensional input spaces, particularly in image-based observations*. Our findings emphasize the role of state representation, reward design, and algorithmic modifications on efficient RL learning, offering insights for future advancements in complex decision-making tasks.

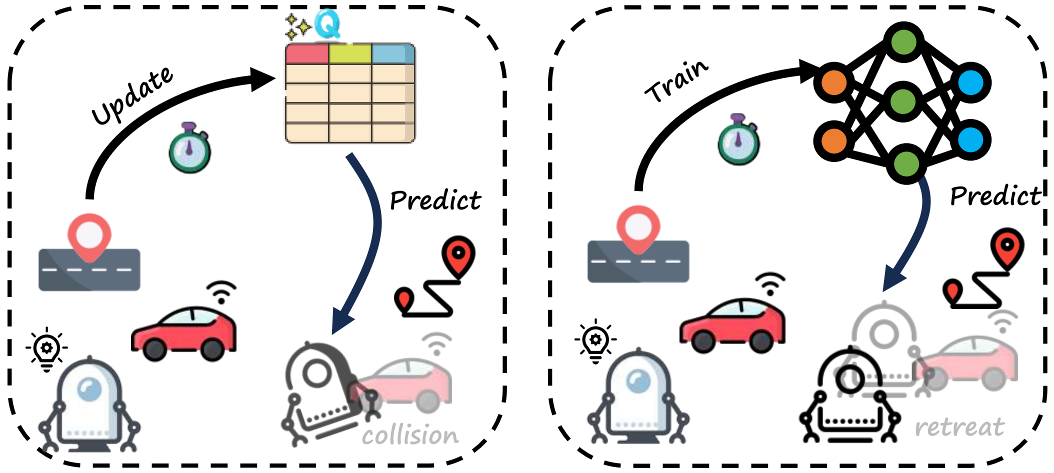

Figure 1: **Traditional reinforcement learning based on Q-Learning (left) and neural network-based deep Q-Learning (right).** Q-Learning is a model-free reinforcement learning algorithm that directly estimates the optimal action-value function using a table to store the Q-values for discrete state-action pairs. However, it is limited to small-scale problems with discrete state and action spaces. In contrast, DQN extends Q-Learning by approximating the Q-function with deep neural networks and can handle high-dimensional and continuous state spaces such as images.

[*]Project Leader.
[†]Experiments and Ablation study.
[‡]Algorithm implementation.
[§]Paper Writing and presentation.

1 Introduction

Reinforcement learning (RL) (Sutton & Barto, 2018) has emerged as a transformative approach to solving sequential decision-making problems, where an agent learns to maximize cumulative rewards through interaction with an environment. The Atari Learning Environment (ALE) (Bellemare et al., 2013; Machado et al., 2018) has been widely adopted as a benchmark for evaluating RL algorithms due to its diverse range of challenging tasks. Among these, the *Freeway* game presents a unique challenge, requiring the agent to navigate a chicken across a multi-lane highway while avoiding oncoming traffic to reach the goal at the top of the screen.

The state space of the *Freeway* game consists of the vertical position of the player (chicken) and the positions of nearby vehicles. The action space is limited to three discrete actions: moving forward, staying stationary, and moving backward. The agent receives a positive reward only when it successfully reaches the goal, while collisions with vehicles result in penalties or loss of progress.

Traditional tabular Q-learning provides a foundational approach to RL by maintaining a Q-value table to represent state-action values. However, its scalability is limited by the exponential growth of the state-action space in high-dimensional environments. To overcome these limitations, Deep Q-Learning (DQN) (Mnih et al., 2015) extends Q-learning by employing a neural network to approximate the Q-value function, enabling generalization across continuous and large state spaces. DQN also introduces techniques such as experience replay and target networks to stabilize training and mitigate the effects of non-stationarity during learning.

The remainder of this paper is organized as follows. section 2 describes the problem formulation, including the state-action space representation and reward mechanism. section 3 presents the Q-learning algorithm, while section 4 extends the discussion to DQN. section 6 reports experimental results and analyzes the findings. Finally, section 7 concludes the paper with several insights and deficiency.

2 Problem Formulation

The objective is to use Q-learning, a model-free RL algorithm, to learn an optimal policy π^* that maximizes the expected cumulative discounted reward. We define the problem as follows:

2.1 Reinforcement Learning Framework

The environment is represented as a tuple $(\mathcal{S}, \mathcal{A}, R, \gamma)$, where:

- $\mathcal{S}$: The state space, consisting of discretized positions of the player and nearby enemy vehicles.
- $\mathcal{A} = \{0, 1, 2\}$: The action space, where 0, 1, and 2 correspond to moving forward, staying still, and moving backward, respectively.
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: The reward function, designed to incentivize upward movement and discourage backward motion.
- $\gamma \in [0, 1)$: The discount factor, quantifying the importance of future rewards.

The goal is to find an optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ that maximizes the cumulative discounted reward:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0, \pi\right]. \tag{1}$$

2.2 State Discretization

In the *Freeway* game, the raw state space consists of continuous positions of the player and enemy vehicles. To make the problem tractable for Q-learning, we apply state discretization:

- The player's vertical position $y_p \in [0, 160]$ is discretized into $n_b = 10$ buckets, with each bucket spanning a range of $\Delta_b = \frac{160}{10} = 16$.

- The horizontal positions of the nearest three enemy vehicles x_{e1}, x_{e2}, x_{e3} are similarly discretized into the same number of buckets.

The mapping function for a scalar value v is defined as:

$$\text{bucket}(v) = \min\left(\left\lfloor \frac{v}{\Delta_b} \right\rfloor, n_b - 1\right). \tag{2}$$

The state is represented as a tuple $(b_{x_{e1}}, b_{x_{e2}}, b_{x_{e3}})$, where each component corresponds to the bucket index of the respective position.

3 Q-Learning Algorithm

Q-learning is a model-free reinforcement learning algorithm that learns the action-value function $Q(s, a)$. The update rule is based on the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a) + \gamma \max_{a'} Q(s', a') - Q(s,a)\right], \tag{3}$$

where α is the learning rate, γ is the discount factor, and s' is the next state after taking action a in state s.

3.1 Algorithm Description

The pseudocode for the Q-learning algorithm is shown in Algorithm 1.

Algorithm 1 Q-Learning for *Freeway*

```
Require: Learning rate $\alpha$, discount factor $\gamma$, exploration rate $\epsilon$, number of episodes $N$
 1: Initialize $Q(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
 2: for each episode $i = 1$ to $N$ do
 3:     Reset environment to initial state $s$
 4:     for each timestep $t = 1$ to maximum timesteps do
 5:         With probability $\epsilon$, choose a random action $a \in \mathcal{A}$, otherwise select $a = \arg\max_a Q(s,a)$
 6:         Execute action $a$, observe reward $r$ and next state $s'$
 7:         Update $Q(s,a)$ using the Bellman equation
 8:         Set $s \leftarrow s'$
 9:         if $s'$ is terminal then
10:             Break
11:         end if
12:     end for
13: end for
```

3.2 State design for Freeway

In Q-learning algorithm, we use a Q-table storing the info of state. In the freeway environment, the state can be a rgb image or a list containing the ram information of the game. However, these two kinds of states are too large. Thus we reduce the state dim by blocking and only considering partial information of the game.
We first get the position of cars and player. We map the positions into 10 blocks so that we can only use 10 state to represent each car. But it is still hard to take all the cars and players into consideration due to that the total 11 objects generate $10^1 1$ states, which is impossible to transverse.
As a result, we only consider the closest three cars, whose Q-table size are 10, 100 and 1000.

3.3 Reward Design

The reward function is designed to guide the agent towards favorable behaviors:

- $r_y = y'_p - y_p$, the vertical displacement of the player, incentivizes moving upward.

- A penalty of $-100 \cdot r_y$ is applied if the player moves backward for the first time in a sequence.
- Successfully reaching the goal yields a reward of $r_{\text{goal}} = +100$.

The total reward at each timestep is computed as:

$$R(s,a) = \begin{cases} r_y + +r_{\text{goal}} - 0.01, & \text{if } r_y > 0, \\ -100 \cdot r_y + r_{\text{goal}} - 0.01, & \text{if } r_y \leq 0. \end{cases} \tag{4}$$

4 Deep Q-Learning (DQN)

While Q-learning has proven to be effective in tabular settings, its scalability to high-dimensional state spaces is limited by the reliance on discrete state-action pairs and the explicit storage of the Q-table. To overcome these limitations, Deep Q-Learning (DQN) (Mnih et al., 2015) introduces a neural network as a function approximator for the Q-value function. This section extends the Q-learning framework discussed earlier by presenting a DQN-based approach to solve the *Freeway* game.

4.1 Motivation for Deep Q-Learning

In environments with large or continuous state spaces, maintaining a Q-table becomes computationally infeasible. Instead, DQN approximates the Q-value function $Q(s,a;\theta)$ using a parameterized deep neural network, where θ denotes the trainable parameters of the network. This allows generalization across similar states and actions. Furthermore, DQN incorporates additional techniques, such as experience replay and target networks, to stabilize training and mitigate the challenges of non-stationarity inherent in reinforcement learning.

4.2 Deep Q-Learning Framework

The DQN framework follows the same reinforcement learning setup as Q-learning but replaces the Q-table with a neural network.

4.2.1 Q-Value Approximation

The Q-value function $Q(s,a;\theta)$ is approximated using a neural network f_θ with weights θ:

$$Q(s,a;\theta) = f_\theta(s,a). \tag{5}$$

Given a state s, the network outputs Q-values for all actions, allowing the selection of the action a^* with the highest estimated Q-value:

$$a^* = \arg\max_a Q(s,a;\theta). \tag{6}$$

4.2.2 Experience Replay

To improve data efficiency and decorrelate training samples, DQN employs a replay buffer $\mathcal{D}$, which stores transitions (s,a,r,s',d), where d indicates whether the next state s' is terminal. During training, a batch of transitions is sampled from $\mathcal{D}$ to compute the loss and update the network parameters.

4.2.3 Target Network

To stabilize training, a separate target network $Q(s,a;\theta^-)$ is maintained, where θ^- is a delayed copy of the current network's weights θ. The target Q-value is computed as:

$$y = r + \gamma \max_{a'} Q(s',a';\theta^-)(1-d). \tag{7}$$

The target network is periodically synchronized with the current network to reduce instability during training.

4.2.4 Loss Function

The mean squared error (MSE) between the predicted Q-values and the target Q-values is used as the loss function:

$$L(\theta) = \mathbb{E}_{(s,a,r,s',d)\sim\mathcal{D}}\big[(Q(s,a;\theta) - y)^2\big]. \tag{8}$$

4.2.5 Algorithm

The pseudocode for the DQN algorithm is provided in Algorithm 2.

Algorithm 2 Deep Q-Learning (DQN)

```
Require: Replay buffer capacity $\mathcal{D}$, batch size $B$, learning rate $\alpha$, discount factor $\gamma$, target update
    frequency $T$.
 1: Initialize Q-network $Q(s, a; \theta)$ with random weights $\theta$.
 2: Initialize target network $Q(s, a; \theta^-)$ with $\theta^- \leftarrow \theta$.
 3: Initialize replay buffer $\mathcal{D}$.
 4: for each episode do
 5:     Reset environment to initial state $s$.
 6:     for each timestep do
 7:         With probability $\epsilon$, choose a random action $a$, otherwise select $a = \arg\max_a Q(s, a; \theta)$.
 8:         Execute action $a$, observe reward $r$, next state $s'$, and done signal $d$.
 9:         Store transition $(s, a, r, s', d)$ in $\mathcal{D}$.
10:         Sample a random minibatch of transitions $(s, a, r, s', d)$ from $\mathcal{D}$.
11:         Compute target Q-value $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)(1 - d)$.
12:         Update network parameters by minimizing:
                $$L(\theta) = \frac{1}{B}\sum_{i=1}^{B}\big(Q(s_i, a_i; \theta) - y_i\big)^2.$$
13:         Every $T$ steps, synchronize $\theta^- \leftarrow \theta$.
14:         Update state $s \leftarrow s'$.
15:         if $d$ is true then
16:             Break.
17:         end if
18:     end for
19: end for
```

5 Actor-Critic Algorithm

The Actor-Critic method combines both value-based and policy-based reinforcement learning approaches. In contrast to Q-learning, which uses a value function to directly estimate the optimal policy, Actor-Critic methods learn both a policy and a value function. This section provides an overview of the Actor-Critic algorithm, its motivation, and its key innovations, particularly in the context of handling environments with complex state spaces.

5.1 Motivation for Actor-Critic

While value-based methods like Q-learning are effective in environments with discrete state-action spaces, they can face challenges in environments with continuous state or action spaces. Q-learning and its deep variant, DQN, consider only the value. However, we also want to use policy gradient to optimize the policy.

Actor-Critic addresses this limitation by decoupling the policy and value functions. Instead of explicitly computing Q-values for every action, the Actor-Critic method uses two separate components:

- **Actor:** A neural network (policy network) that directly parameterizes the policy $\pi(a|s; \theta_\pi)$. The actor selects actions based on the current policy.

- **Critic:** A neural network (value network) that estimates the value function $V(s;\theta_V)$, evaluating the actions taken by the actor by computing the expected return from the current state.

This separation allows the actor to improve the policy directly, while the critic evaluates the quality of the selected actions based on the current value function. The critic's feedback is then used to update the actor's policy, guiding it toward more rewarding actions.

Furthermore, the Actor-Critic method can handle both discrete and continuous action spaces, making it highly flexible and suitable for a wide range of reinforcement learning problems.

5.2 Actor-Critic Framework

The Actor-Critic algorithm is a reinforcement learning framework that combines value-based and policy-based methods. It uses two components: the actor, which chooses actions, and the critic, which evaluates the actions taken by the actor.

5.2.1 Policy and Value Function Approximation

The actor's policy $\pi(a|s;\theta_\pi)$ is parameterized by a neural network with weights θ_π, while the critic estimates the state value function $V(s;\theta_V)$, also parameterized by a neural network with weights θ_V:

$$\pi(a|s;\theta_\pi) = f_{\theta_\pi}(s), \quad V(s;\theta_V) = f_{\theta_V}(s). \tag{9}$$

The actor selects actions according to the policy:

$$a^* = \arg\max_a \pi(a|s;\theta_\pi). \tag{10}$$

The critic evaluates the action by estimating the expected return:

$$\hat{A}(s,a) = Q(s,a) - V(s). \tag{11}$$

5.2.2 Advantage Function

The advantage function $\hat{A}(s,a)$ is used to guide the actor's updates. It provides a measure of how much better or worse an action is compared to the expected value of the state:

$$\hat{A}(s,a) = r + \gamma V(s';\theta_V) - V(s;\theta_V). \tag{12}$$

This helps to reduce the variance of the policy gradient estimate, improving learning stability.

5.2.3 Loss Functions

The actor's loss is based on the policy gradient theorem, which maximizes the expected return. The critic's loss is based on minimizing the temporal difference (TD) error.

For the actor, the loss is:

$$L_{\text{actor}}(\theta_\pi) = -\mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\big[\hat{A}(s,a)\log\pi(a|s;\theta_\pi)\big]. \tag{13}$$

For the critic, the loss is:

$$L_{\text{critic}}(\theta_V) = \mathbb{E}_{(s,r,s')\sim\mathcal{D}}\big[\big(r + \gamma V(s';\theta_V) - V(s;\theta_V)\big)^2\big]. \tag{14}$$

5.2.4 Algorithm

The pseudocode for the Actor-Critic algorithm is provided in Algorithm 3.

6 Experiment

6.1 Environment Setup

In this section, we describe the experimental setup and environment used to evaluate the proposed model. Our experiments focus on the Freeway game from the Atari suite, which is provided through

Algorithm 3 Actor-Critic Algorithm

```
1: Initialize policy network parameters $\theta_\pi$ and value network parameters $\theta_V$
2: for each episode do
3:     Initialize state $s_0$ from environment
4:     Initialize empty trajectory $\tau = \{(s_0, a_0, r_0, s_1), \dots\}$
5:     while not terminated do
6:         Sample action $a_t$ from current policy $\pi(a_t|s_t;\theta_\pi)$
7:         Take action $a_t$ and observe reward $r_t$ and next state $s_{t+1}$
8:         Append $(s_t, a_t, r_t, s_{t+1})$ to trajectory $\tau$
9:         Update state $s_t = s_{t+1}$
10:    end while
11:    for each step in trajectory $\tau$ do
12:        Compute TD target:
               $$\hat{A}_t = r_t + \gamma V(s_{t+1};\theta_V) - V(s_t;\theta_V)$$
13:        Update value network parameters $\theta_V$ using:
               $$\theta_V \leftarrow \theta_V + \alpha_V \nabla_{\theta_V}\left(\hat{A}_t^2\right)$$
14:        Update policy network parameters $\theta_\pi$ using:
               $$\theta_\pi \leftarrow \theta_\pi + \alpha_\pi \nabla_{\theta_\pi}\left(\log \pi(a_t|s_t;\theta_\pi)\hat{A}_t\right)$$
15:    end for
16: end for
```

the OpenAI Gym platform (Brockman et al., 2016). We test the performance of our model under two difficulty modes of the Freeway game, which requires the agent to cross a busy street while avoiding traffic, with the difficulty increasing in higher modes due to faster-moving cars and more complex obstacle patterns. This setting allows us to assess the model's ability to adapt to varying levels of complexity and learn an optimal policy under different conditions.

For this experiment, we utilize the open-source environment wrapper provided by Anand et al. (Anand et al., 2019), which helps us extract useful information from the environment, such as state representations, that are crucial for training and evaluating the model. This wrapper ensures consistency and standardization in how the agent interacts with the environment, making it easier to focus on evaluating the model's performance across difficulty modes.

The experimental setup includes both training and evaluation stages, where we assess the model's performance based on cumulative reward and agent stability. The following subsections provide a detailed description of the environment configuration, model settings, and evaluation methodology.

6.2 Q-Learning experiment

The Q-learning algorithm is trained over 10 episodes, each with a maximum of $10,000$ timesteps. The following hyperparameters are used:

- Learning rate $\alpha = 0.1$
- Discount factor $\gamma = 0.99$
- Exploration rate $\epsilon = 0.1$

On this experimental setup, we further investigate the impact of the observable state space on the performance of the Q-learning algorithm. Specifically, we consider three configurations of the player's observable state space:

- (i) Observing only the vehicles in the previous row.
- (ii) Observing the vehicles in the current row and the previous row.

- (iii) Observing the vehicles in the current row, the previous row, and the next row.

We conduct experiments under each configuration and observe that the Q-learning algorithm converges effectively only in configuration (ii). This is because:

- A reduced state space, as in configuration (i), lacks sufficient information for the player to make informed decisions, resulting in suboptimal strategies.
- An excessively large state space, as in configuration (iii), leads to insufficient coverage of all possible states during training, thereby hindering the learning process and preventing convergence.

	One Row	**Two Rows**	**Three Rows**
Average Reward	15	25	21

Table 1: Q-learning average reward under different observing settings.

6.3 DQN EXPERIMENT

6.3.1 DQN WITH POSITIONAL INFORMATION

The DQN algorithm, with positional information as input, is trained over 500 episodes, each with a maximum of $10,000$ timesteps. The following hyperparameters are used:

- Learning rate: $\alpha = 10^{-4}$ (1e-4) with the Adam optimizer.
- Discount factor: $\gamma = 0.99$
- Batch size: 64

The epsilon decay formula is defined as:

$$\epsilon = \epsilon_{\text{end}} + (\epsilon_{\text{start}} - \epsilon_{\text{end}}) \cdot \exp\left(-\frac{\texttt{steps\_done}}{\texttt{EPS\_DECAY}}\right), \tag{15}$$

where:

- ϵ_{start}: Initial exploration rate (`EPS_START` = 0.9),
- ϵ_{end}: Minimum exploration rate (`EPS_END` = 0.1),
- `steps_done`: Number of steps completed so far,
- `EPS_DECAY`: Decay rate (`EPS_DECAY` = 1000).

Since the original reward signal is sparse, being received only when the agent reaches the goal, we designed a shaped reward function to guide the agent toward favorable behaviors. This augmented reward function provides intermediate feedback, encouraging the agent to make progress toward the goal by rewarding desirable actions along the way.

- $r_y = y'_p - y_p$, the vertical displacement of the player, incentivizes moving forwards.
- A linear term $r_l = (10 \cdot y'_p)/180$ if the player moves forwards. (where $180 =$ max_length)
- A penalty of $-10 \cdot r_y$ is applied if the player moves backwards.
- Successfully reaching the goal yields a reward of $r_{\text{goal}} = +50$.

The total reward at each timestep is computed as:

$$R(s,a) = \begin{cases} r_y + r_l + r_{\text{goal}} - 0.01, & \text{if } r_y > 0, \\ -10 \cdot r_y + r_{\text{goal}} - 0.01, & \text{if } r_y \leq 0. \end{cases} \tag{16}$$

For the Deep Q-Network (DQN), we use a discretized observation space that captures the positional information of its surroundings. Specifically, the observation space is designed as a list consisting of the positional information of 10 rows of cars.

The detailed implementation of this design can be found in Section 2.2.

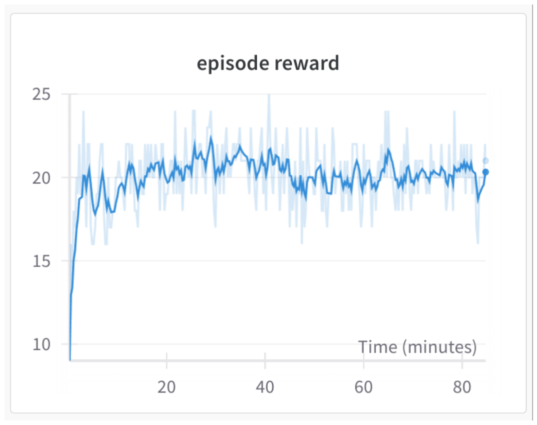

Figure 2: Reward curve of DQN-info

The reward curve of DQN with positional information is shown in Figure 2. We observe that the reward quickly reaches approximately 25 in a relatively short period. Afterward, the curve stabilizes and the reward only changes within a small range.

This behavior indicates that while DQN with positional information can quickly capture some initial patterns or relationships in the environment, it struggles to learn more complex or useful information as training progresses.

6.3.2 DQN WITH IMAGE INFORMATION

In this section, we describe experimental setup, especially the training process for the DQN model with image information.

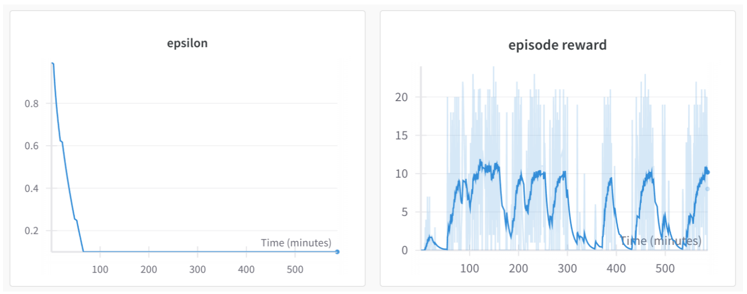

Figure 3: Epsilon decay and reward curve of one-stage training

The RGB image retrieved from the environment is converted to grayscale, and resized to 84x84 pixels. This process reduces the complexity of the input by eliminating color information, which is often irrelevant in this spatial-related task. The resizing to 84x84 ensures consistency in input dimensions, optimizing computational efficiency while retaining enough detail for effective learning.

Initially, we used a one-stage training strategy, training the policy in a single stage. As shown in Figure 3, we observed that the reward curve begins with significant fluctuations, particularly in the first 100 minutes. This is common in the early stages of reinforcement learning where the agent is exploring the environment and learning its dynamics. After the initial fluctuations, there is a noticeable trend of gradual improvement, as the rewards consistently increase over time. But throughout the training process, the reward curve shows periodic oscillations, which could be indicative of the agent getting stuck in local optimal because of the remarkably decayed epsilon and the insufficient learned experience.

To address the limitations observed in the one-stage approach, we shifted to a more structured training strategy consisting of three distinct stages. The training process collaborates in three serial stages due to the complex state space. For the sake of the convergence speed, we elaborate every stage, each with a specific goal aimed to progressively improving the model's performance. The target is to maximize the reward while ensuring stable and efficient learning.

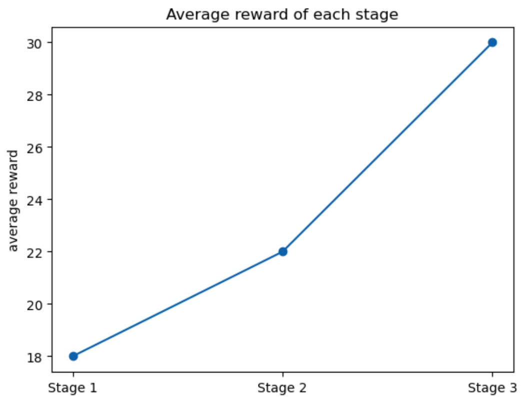

Figure 4: Performance of the agent after each stage

For each stage, the key characteristic and hyperparameters are given as follows,

1. Initial Training with a Small Replay Buffer
   - Training episodes $num\_episodes = 600$
   - Replay Buffer size $capacity = 40,000$
2. Fine-Tuning with Reset Epsilon
   - Training episodes $num\_episodes = 800$
   - Replay Buffer size $capacity = 40,000$
3. Enhancing Performance with Larger Replay Buffer
   - Training episodes $num\_episodes = 1400$
   - Replay Buffer size $capacity = 90,000$

The primary objective of the first stage is to quickly develop an initial model that could perform reasonably well, albeit with less stability and generalization. To achieve this, we utilize a relatively small replay buffer to allow for faster updates and quicker accumulation of experiences, though the small size limited the diversity of experiences, leading to potential overfitting on recent interactions. The model reaches a cumulative reward of 18 after training as shown in 4.

In the second stage, the model from stage 1 is fine-tuned with reset epsilon greedy exploration strategy. This stage aims to stabilize and refine the model. The cumulative reward is improved to 22, indicating better performance and more stable learning. The decision to seperate this from stage 1 allows more chance to explore rather than exploiting the insufficient experience because of the highly decayed epsilon.

The final stage increases the replay buffer size. This expansion allowed for more diverse experiences, thereby reducing the temporal correlation of the experiences sampled for updates. As a result, the model demonstrated improved generalization and stability, with a significant increase in the cumulative reward to 30. The larger buffer enhanced the Q-value updates and minimized the risk of overfitting, ultimately leading to a more robust policy.

6.4 EVALUATION

The trained policy for each model is evaluated five times in the Arcade Learning Environment (ALE) under both mode 0 and mode 1 difficulty settings. During the evaluation stage, the optimal action is selected to ensure a consistent and fair assessment of model performance.

	Mode 0		**Mode 1**	
	Avg	**Max**	**Avg**	**Max**
DQN_info	23.8 ± 0.32	25	4.8 ± 0.17	5
DQN_image	30.2 ± 0.17	31	6.0 ± 0.76	9
Human	28.2 ± 0.17	29	4.2 ± 0.32	5

Table 2: DQN reward evaluation and Human performance

The DQN model trained with only positional information showed suboptimal performance, particularly in its ability to generalize its actions, such as waiting and avoiding cars. The policy exhibited a tendency to continue moving forward in most situations, only waiting during specific instances. This behavior can be attributed to the limited observation space, which encodes only 10 discrete integers, providing a sparse representation of the environment. As a result, the model struggled to represent complex scenarios effectively. Interestingly, a smaller network architecture resulted in better performance, suggesting that the reduced model capacity helped mitigate overfitting of the sparse input data.

We posit that the primary factor behind the poor generalization of DQN when using positional information is the inherent limitations of the observation space. To overcome these constraints, we propose shifting to an image-based observation modality. This approach allows for a richer, more detailed representation of the environment, which we hypothesize will enable the model to better capture the complexities of the task. Preliminary results in mode 0, as shown in Table 2, support this hypothesis, with the DQN trained on image data outperforming its positional information counterpart, both in terms of average and maximum rewards. The enhanced observation space provided by images appears to significantly improve the model's ability to generalize across different situations, resulting in a more robust policy.

Further experiments will explore the impact of image-based observations in mode 1, where the results are still comparable to human performance. These findings underline the importance of choosing the right form of input data to facilitate effective learning in reinforcement learning tasks.

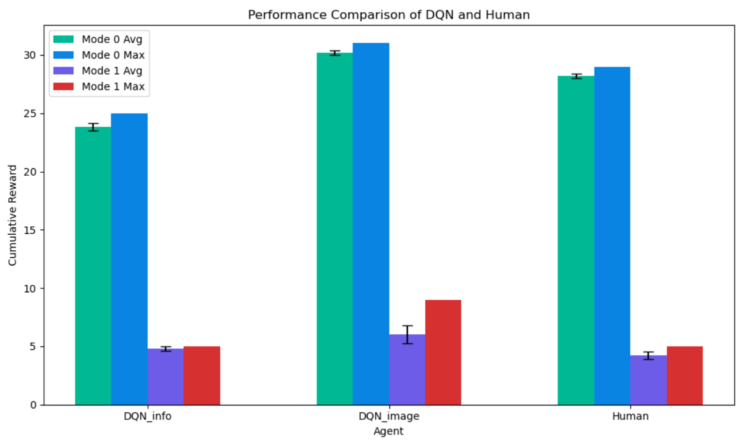

Figure 5: Performance Comparison of Our Model and Human

7 Conclusion

This research set out to investigate the performance of reinforcement learning (RL) algorithms on the Atari 2600 Freeway game, focusing on the comparison between Q-learning and Deep Q-Networks (DQN) under varying conditions of observations and input formats. Specifically, we conducted experiments to assess the impact of different observation sizes in Q-learning and explored the effects of using both informational and image-based inputs in DQN.

Our key findings indicate that increasing the number of observations in Q-learning does enhance performance up to a certain point, beyond which diminishing returns are observed. Q-learning, when provided with a larger observation window, demonstrates better generalization and decision-making, but its effectiveness plateaus as the complexity of the input grows. On the other hand, DQN, which integrates a neural network to process inputs, significantly outperforms Q-learning, especially when provided with image-based inputs. This suggests that DQN is more suited for environments with complex and high-dimensional observation spaces, such as those in Atari games, where raw pixel data carries more relevant information for decision-making than a traditional feature-based approach.

Despite the promising results, our study has several limitations. Although we implemented an Actor-Critic architecture, we did not conduct experiments to evaluate its performance within the context of this task. As a result, the potential advantages of Actor-Critic methods, known for their efficiency in certain RL problems, remain unexplored in this study.

In conclusion, this study contributes to the ongoing research in reinforcement learning by providing a detailed comparison between Q-learning and DQN on the Freeway task, highlighting the advantages of DQN when dealing with high-dimensional inputs.

8 External resources

As for all the external resources (e.g., code, library, tools) we used, we have clearly stated in subsection 6.1

References

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised state representation learning in atari. *arXiv preprint arXiv:1906.08226*, 2019.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. doi: 10.1613/jair.3912. URL https://jair.org/index.php/jair/article/view/10819.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, et al. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018. doi: 10.1613/jair.5699. URL https://jair.org/index.php/jair/article/view/11230.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018. ISBN 978-0262039246. URL https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf.