Enhancing DQN and Comparing with Actor-Critic: Performance on Classic Control Tasks

Hao Yuan [1] Faquan Wang [1] Yixuan Liu [1] Yao Jia [1]

Abstract

This work investigates the performance of advanced Deep Q-Networks (DQN) in Actor-Critic (AC) methods on three classic control problems: CartPole, Acrobot, and MountainCar. This work assesses their impact on learning stability, efficiency, and convergence by incorporating state-of-the-art techniques such as soft updates, double Q-learning, and reward shaping specific to the task. Empirical results showed that while traditional approaches—pure DQN and Actor-Critic—excel in structured tasks with dense rewards, the more challenging tasks containing sparse or delayed rewards are found in, for example, Acrobot and MountainCar, which demand much stronger strategies like reward shaping and overestimation-bias correction. This therefore has brought out an intricate interplay among algorithm design, task complexity, and reward structure to offer insights on how to advance deep reinforcement learning in complex environments.

1. Introduction

Over the decades, Reinforcement Learning (RL) has transitioned from a theoretical concept to well defined algorithms that can solve more complicated decentralized decision making problems.

Dynamic programming (DP) was introduced by Richard Bellman in 1950s that laid the early foundation of RL.(Bellman, 1957) Dynamic Programming (DP) based methods like Policy Iteration and Value Iteration offered a structured approach to calculating the optimal policies using the recursive structure of the Bellman Equation. But these techniques depended on a complete model of the environment, making them computationally expensive and intractable for large-scale tasks.

To overcome these limitations, model-free methods were developed in the 1980s, such as based on Monte Carlo and Temporal-Difference (TD) learning. Monte Carlo methods estimated value functions by averaging the returns over sampled episodes, whereas TD learning introduced the technique of bootstrapping: updating value estimates using the predicted future rewards instead of true returns. The most notable of these was Q-learning, proposed by Watkins in 1989,(Watkins, 1989) which provided a foundational RL algorithm for agents to learn an optimal policy regardless of the actions they take. The main novelty that became enabling here, by opening a much broader set of possible use-cases, was that this was allowing agents to operate in complex environments without a complete model.

In the 1990s and early 2000s, research in reinforcement learning began to center more and more around Policy Gradient (PG) methods. Unlike value-based methods, PG methods placed greater emphasis on initializing parameters directly based on performance. The REINFORCE algorithm (Williams, 1992) embodied such an approach, judging how well it worked by comparing outputs to truth one sentence at a time. Using large, but finite random trajectories of traffic simply giving infinite variance, the most straightforward policy-gradient methods could be quite computationally expensive and slow.

In 2000, Konda and Tsitsiklis proposed **Actor/Phase (AC) algorithms** which are specially oriented towards the problem of having continuous action spaces and using reinforcement learning. (Konda & Tsitsiklis, 2000) These algorithms are a mixture of strength policy-based and value based methods. There is a ”regular” actor that selects actions. And then there is a critic who assesses the actions with an estimate of value–function. This new mixture can greatly speed up learning and make it more stable–particularly in situations where fine-grained control is needed. The key innovation of AC algorithms is that they make it impossible for continuous actions to escape notice, something quite difficult with traditional Q-learning-based methods. By optimizing policy directly and also using the policy to guide the estimation of values, AC methods offer both scalability and

[1]Department of Electrical Engineering, Columbia University, New York, USA. Correspondence to: Hao Yuan <hy2814@columbia.edu>, Faquan Wang <fw2412@columbia.edu>, Yixuan Liu <yl5393@columbia.edu>, Yao Jia <yj2788@columbia.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

precision at once. Another benefit is that they hold up well in high-dimensional environments; these two are making AC methods the clear choice for applications in the real world.

AC algorithms' adaptability to large-scale cases means that they have found many uses in various fields. In robotics, for example, schemes like Deep Deterministic Policy Gradient (DDPG) (Fujimoto et al., 2018) and Soft Actor-Critic (SAC) (Haarnoja et al., 2018)have been applied to machines used for operations such as robotic arm manipulation or walking on two legs–allowing them learn stable and effective forms; other possibilities are still waiting. For example, in the field of unmanned systems, AC methods can be used to teach drones and vehicles adaptive planning strategies which accommodate changing conditions where risks abound. In addition, AC technology has helped usher in a new era for competitive gaming systems–for instance OpenAI Five in Dota 2 (OpenAI, 2019) and AlphaStar in StarCraft II (Vinyals et al., 2019), which offers human-like skills that surpass those of all but professional players! These instances provide good examples for showing how broad the range of applications can get with a technique like ANP or AGP on continuous control issues.

In 2013, Mnih et al. presented the **Deep Q-Network (DQN) algorithm** and thus a major breakthrough in reinforcement learning. (Mnih et al., 2013) The algorithm is specially designed to tackle problems in discrete action spaces: places where you've got a limited number of different things that might happen from which to choose and order them according to their likelihoods which element happens next or first. Combining Q-learning with deep neural networks, researchers applied DQN to learn directly from raw environmental inputs. But these initial versions were very slow to train and prone to diverge.

In 2015, Mnih et al. took advantage of their original paper to build on it key innovations such as target networks to stabilize the Q-value updates and experience replay to remove correlations between consecutive samples. (Mnih et al., 2015) As a result, DQN was highly suitable for discrete domains: it evaluated and chose actions from some limited number of finite choices, even in highly complicated environments. DQN was able to achieve human-level or superhuman performance in many Atari games where each game required selecting from a given set of discretes actions such as these three, moving left-right-and down.

DQN has been widely employed in discrete action spaces. In video games, it had a revolutionary impact on those old classics like Atari ones. Without needing prior features learned from data that humans worked on it even reached some top scores. For robotics, DQN is used to control the route of robots in areas divided into small lattices. A robot simply has to move through certain cells (toward its goal), turning at junctures just so as not enter a prohibited cell at any time. In financial markets, DQN is used—entryprices, which particular stocks to buy and when to sell them and (unless you expect that information is to continue) how much. These examples point up seemingsincery how DQN's innovations have transformed reinforcement learning incases with finite, discrete action sets.

In addition to foundational advancements like DQN and AC, methods such as soft updates and reward shaping have been widely adopted to optimize reinforcement learning algorithms.

In 1999, Ng et al. introduced reward shaping as a form of learning efficiency, which bothers the process itself by adding source-generated rewards into training for guiding behavior optimization. (Ng et al., 1999) So an agent could explore desirable behaviours, steering its exploration toward them; this solved the problems that special or slow returns bring about. By providing feedback in the intermediary steps, reward shaping alleviates the tribulation of exploratory search and quickens convergence.

In 2015 Lillicrap (et al.) introduced the notion of soft updates, where in Deep Deterministic Policy Gradient (DDPG) we gradually update the target network weights as a weighted average between our current network's weights and those used just before set to be a targets. (Lillicrap et al., 2015) This has the positive effect that in continuous action spaces with off-the-shelf algorithms such as TD (Temporal Difference) learning or DDPG one can reduce oscillations or completely eliminate them. It also allows us to avoid such phenomena as flipping wei more easily giving readers some context for their reading as well as ensuring that any advertisements are useful to help out consumers looking at options instead of pestering around everywhere on the Web. By stabilizing the target network, soft updates improve convergence, and a number of deep reinforcement learning frameworks now use them as a standard component.

The development of classical reinforcement learning algorithms reflects the process by which theories are continuously revisited and corrected to reflect practical experience as well. For example, DQN, AC (Actor-Critic) and various evolutions of that method have now become the cornerstones of modern RL (Reinforcement Learning); they have brought breakthroughs throughout a wide range domains, and will pave tomorrow's new heights in intelligent machines.

2. Principles

2.1. Actor-Critic(AC) Principles

Actor-Critic (AC) is a class of policy gradient algorithms that integrate value-based and policy-based reinforcement learning approaches. There are two main parts of this: an actor, which is the policy itself, and chooses what action to take, and a critic, which evaluates such actions by predicting a value function. This framework enables fast and scalable training in search spaces where both the states and actions are large.

In Actor-Critic methods, the state-value function $v(s; w)$ is applied to the second state, S_{t+1}, to compute the one-step return $G_{t:t+1}$, which combines the immediate reward R_{t+1} and the discounted value of the next state. This one-step return serves as a low-variance estimate of the return and provides a way to evaluate the action taken. Specifically, the one-step return is calculated as:

$$G_{t:t+1} = R_{t+1} + \gamma v(S_{t+1}; w).$$

The temporal difference (TD) error, δ, is then defined as the difference between the one-step return and the current state-value estimate:

$$\delta = R_{t+1} + \gamma v(S_{t+1}; w) - v(S_t; w).$$

This error serves as the basis for updating both the actor and critic.

The policy update in Actor-Critic methods is derived from the policy gradient theorem and adjusted using the TD error. The parameters of the policy, θ, are updated as follows:

$$\theta_{t+1} = \theta_t + \alpha_\theta \, \delta \, \nabla_\theta \ln \pi(A_t|S_t; \theta),$$

where α_θ is the step size, $\pi(A_t|S_t; \theta)$ represents the policy's probability of taking action A_t given state S_t, and $\nabla_\theta \ln \pi$ is the gradient of the log-policy with respect to θ. This update encourages the policy to increase the likelihood of actions that lead to positive TD errors while reducing the likelihood of actions that result in negative TD errors.

The state-value function is updated using a semi-gradient approach, similar to TD(0). The update for the critic's parameters w is given by:

$$w_{t+1} = w_t + \alpha_w \, \delta \, \nabla_w v(S_t; w),$$

where α_w is the step size for the value function. This update reduces the discrepancy between the estimated and actual values of states, improving the accuracy of the critic's evaluation over time.

The one-step Actor-Critic algorithm is an incremental, online process. At each time step, the agent observes the current state S_t, selects an action A_t based on the policy $\pi(A_t|S_t; \theta)$, executes the action, and receives the next state S_{t+1} and reward R_{t+1}. While this transition occurs, the TD error is calculated and both the actor and critic are updated. This process continues until an episode finishes, making sure to improve both its policy and value function. For more complex tasks, the one-step return can be extended to n-step returns.

Actor-Critic methods are widely used in continuous control tasks, where the action space is infinite, and traditional value-based methods like Q-learning struggle. By leveraging the TD error and maintaining separate updates for the actor and critic, these methods achieve both stability and efficiency in learning policies for complex environments.

2.2. Deep Q-Network (DQN) Principles

The foundation of DQN is the Q-learning algorithm, which updates $Q(s, a)$, the action-value function, session after session to make it approximate as much as possible the optimal $Q^*(s, a)$.The Q-learning update rule is defined as:

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

where s, a, R, s' represent the current state, action, reward, and next state, α is the learning rate, and γ is the discount factor. $\max_{a'} Q(s', a')$ represents the estimated maximum future reward for the next state s'.

In DQN, the action-value function $Q(s, a; w)$ is parameterized by the weights w of a deep neural network. The loss function used to train the network minimizes the temporal difference (TD) error, expressed as:

in DQN, the weights w of a deep neural network parameterize the action-value function $Q(s, a; w)$.The temporal-difference (TD) error is minimized to train the network with the following loss function:

$$L(w) = E\left[\left(R + \gamma \max_{a'} \tilde{Q}(s', a'; w^-) - Q(s, a; w) \right)^2 \right],$$

where $\tilde{Q}(s', a'; w^-)$ represents the action-value function estimated by a target network with fixed weights w^-, updated periodically to improve training stability.

To address instability in Q-learning caused by bootstrapping, DQN employs a target network, $\tilde{Q}(s', a'; w^-)$, whose weights are updated less frequently than the primary network. The weight update rule for the primary network is:

$$w_{t+1} = w_t + \alpha \left[R + \gamma \max_{a'} \tilde{Q}(s', a'; w_t^-) - Q(s, a; w_t) \right] \nabla Q(s, a; w_t).$$

where w_t represents the current network weights, w_t^- represents the fixed weights of the target network, and $\nabla Q(s, a; w_t)$ is the gradient of the predicted action-value with respect to the network weights. This decoupling of the target network w_t^- from the primary network reduces oscillations and improves stability.

DQN also uses gradient clipping to prevent excessively large updates. The TD error is clipped to a predefined range, typically $[-1, 1]$, as follows:

$$\delta = \text{clip}\left(R + \gamma \max_{a'} \tilde{Q}(s', a'; w_t^-) - Q(s, a; w_t), -1, 1\right).$$

With this clipping, even in cases of high variance rewards, the updates of gradients will remain stable.

DQN achieves better learning efficiency through experience replay. A buffer stores transitions (s, a, R, s'), then random samples from training are used as batches. This means that subsequent transitions are no longer correlated in time-many previous experiences can be put to use many times; thereby improving sample efficiency and reducing the variance of updates.

3. Learning Framework And Algorithms

Task Overview and Environment Settings

In this section, we provide an overview of the three selected tasks—**MountainCar**, **CartPole**, and **Acrobot**—from the OpenAI Gym library. We also outline their default configurations, including observation spaces, action spaces, reward structures, and termination criteria.

Task Descriptions

CartPole In the CartPole task, the agent must balance a pole on a cart by applying forces to move the cart left or right. The goal is to prevent the pole from falling or the cart from moving out of bounds.

Acrobot The Acrobot task involves a two-link robotic arm. The goal is to swing the end of the arm above a certain height, defined as above the horizontal axis.

MountainCar The objective of the MountainCar task is to drive a car up a steep hill. The agent must learn to build momentum by moving back and forth to reach the flag at the top of the hill.

Environment Parameters

CartPole: State Space: 4D vector (Position $[-4.8, 4.8]$, Velocity [unbounded], Angle $[-24°, 24°]$, Angular Velocity [unbounded]). Action Space: 2 discrete actions (0: Push cart left, 1: Push cart right).

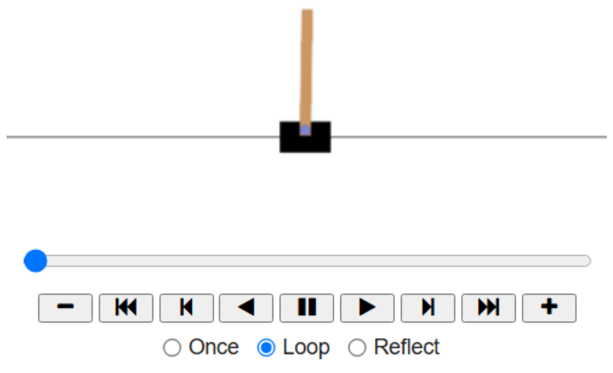

Figure 1. Cartpole Task

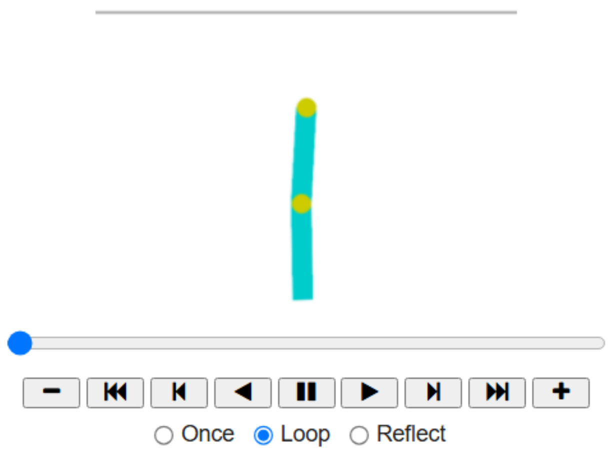

Figure 2. Acrobot Task

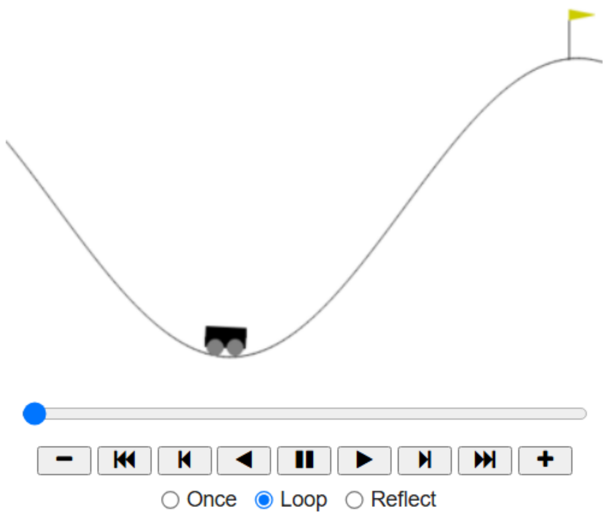

Figure 3. Mountaincar Task

Acrobot: State Space: 6D vector (Cosine/Sine of two joint angles $[-1, 1]$, Angular velocities of the joints [unbounded]). Action Space: 3 discrete actions (0: Torque joint 1, 1: Torque joint 2, 2: No torque).

MountainCar: State Space: 2D vector (Position $[-1.2, 0.6]$, Velocity $[-0.07, 0.07]$). Action Space: 3 discrete actions (0: Accelerate left, 1: No acceleration, 2: Accelerate right).

Reward and Termination Criteria

CartPole: Reward: $+1$ per time step the pole is balanced. Termination: Angle exceeds $\pm 12°$, cart moves ± 2.4, or 500 time steps. **Acrobot:** Reward: -1 per time step until the goal height is reached. Termination: End-effector reaches target height (0.5 above horizontal) or 500 time steps. **MountainCar:** Reward: -1 per time step until the goal is reached. Termination: Position ≥ 0.5 or 200 time steps.

Table 1. Reward and Termination Criteria

Task	Reward	Termination Criteria
CartPole	$+1$ per time step	Angle $\pm 12°$, Position ± 2.4, or 500 steps.
Acrobot	-1 per time step	Goal height (0.5 above horizontal) or 500 steps.
MountainCar	-1 per time step	Position ≥ 0.5 or 200 steps.

Environment Settings

Performance is measured by running each task for 1,000 epochs, and the average reward is used as the evaluation metric.

To better understand model performance on each task, we define reward ranges that indicate learning levels:

- **Start Learning**: The model begins to exhibit meaningful behavior but still struggles to optimize performance.
- **Gradual Mastery**: The model demonstrates improving behavior, achieving intermediate rewards.
- **Perfect Mastery**: The model fully learns the task, achieving near-optimal rewards.

Table 2 summarizes the reward ranges for CartPole, Acrobot, and MountainCar.

Table 2. Reward Ranges Representing Learning Levels in Classic Control Tasks

Task	Start Learning	Gradual Mastery	Perfect Mastery
CartPole	0–100	100–450	450–500 (Max Steps)
Acrobot	-500 to -200	-200 to -100	-100 to -50 (Goal Height Reached)
MountainCar	-1000 to -300	-300 to -150	-150 to around -110 (Position $\geq$ 0.5)

3.1. Original Model

In DQN and it's improvement experiments, we use following parameters and environments in the Table 3 and Table 4 with standard QNetwork and replay buffer implementation.

Table 3. Experimental Environment Configuration

Component	Specification
Hardware	NVIDIA Tesla T4 GPU
Framework	PyTorch 2.5.1 + CUDA 12.1
RL Library	Gym 0.25.2
Programming Language	Python 3.10
Training Platform	Google Colab
Operating System	Ubuntu 20.04 (Colab default)

Table 4. Hyperparameter Settings for Different Tasks

Parameter	CartPole	Acrobot	Mountain Car
Learning Rate	5×10^{-4}	5×10^{-4}	5×10^{-4}
Batch Size	64	64	64
Hidden Layers	64,64	64,64	64,64
Discount Factor (γ)	0.99	0.99	0.99
Replay Buffer Size	10,000	50,000	50,000
Optimizer	Adam	RMSprop	RMSprop
ϵ Start	1.0	1.0	1.0
ϵ End	0.01	0.01	0.01
ϵ Decay Steps	5,000	10,000	100,000
Target Update Steps	1,000	1,000	1,000
Total Episodes	1000	1000	1000

*Exponential decay is applied to ϵ.

The training procedure for the DQN agent, including ϵ-greedy exploration, replay buffer sampling, target network updates, and loss optimization, is summarized in Algorithm 1. The key components and flow are as follows:

Algorithm 1 Agent Training with DQN and Reward Shaping

```
1: Input: Environment $env$, Q-network architecture $hidden\_layers$, hyperparameters $\gamma$, $\epsilon_{start}$, $\epsilon_{end}$, $decay$, replay buffer size, batch size, optimizer, loss function, optional reward shaping function $shape\_reward$
2: Initialize Q-network $policy\_net$ and target network $target\_net$
3: Initialize replay buffer $buffer$
4: Update $target\_net \leftarrow policy\_net$
5: for episode = 1 to $num\_episodes$ do
6:     Reset environment, get initial state $state$
7:     for $t = 1$ to max steps per episode do
8:         Update $\epsilon \leftarrow \epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) \cdot \exp(-steps/decay)$
9:         Select action $a$ using $\epsilon$-greedy strategy
10:        Execute action $a$, observe reward $r$, next state $next\_state$, done flag
11:        if $shape\_reward$ is not None then
12:            Apply $r \leftarrow shape\_reward(r, state, next\_state, a, done)$ {Reward Shaping}
13:        end if
14:        Store $(state, a, r, next\_state, done)$ in $buffer$
15:        $state \leftarrow next\_state$
16:        if buffer size $> batch\_size$ then
17:            Sample minibatch from $buffer$
18:            Compute Q-values and target Q-values
19:            Compute loss and update $policy\_net$ using optimizer
20:            Update $target\_net$ (soft update with $\tau$ or periodic hard update)
21:        end if
22:        if done or truncated then
23:            Break episode
24:        end if
25:    end for
26:    Record episode reward
27:    Log average reward every 50 episodes
28: end for
29: Output: Training rewards
```

4. Empirical Results

4.1. Actor-Critic

4.1.1. Cart Pole

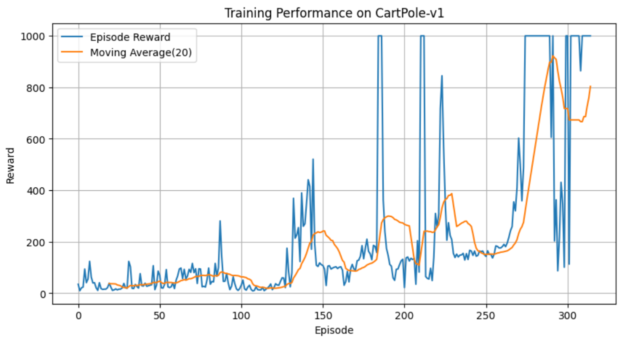

Figure 4. Training Performance of Actor-Critic on CartPole.

Experimental results and plots provided in Fig.4 depict the outcome of an ActorCritic model run for the reinforcement learning CartPole-v1 environment. The results plotted here prove that there is progressive improvement in episodic rewards and convergence of returns as episodes of training progress. These initially have high-variance rewards corresponding to the exploration phase while the policy learns how to balance the cart and pole system. This is further supported by the relatively low values of running rewards and the high variability in episode rewards within the first 100 episodes.

Starting from episode 150 onwards, a marked improvement can be seen: the moving average of rewards begins to go upwards noticeably. It is the period when the policy actually starts to exploit, and it starts to pick up more optimal actions, hence returns higher rewards. The moving average steadily goes uphill, which reflects intense learning behavior and points out that the Actor-Critic model is very capable of performing efficient policy optimization in this continuous action space.

The model was working great at the 300th episode, with an average reward of 546.68. It can be seen here that the model is working at a policy close to the optimum, with a number of episodes reaching the limit of 1000 as the maximum reward. Indeed, the final result obtained, where the model drove in the environment at episode 314 with an average reward of 481, pinned its efficiency in mastering the task after a few training cycles. A standard deviation of 0.0 on the final evaluation reflects not only convergence in this model but also the settling of a deterministic policy adept at solving the environment with repetitions.

Results also showed the effectiveness of the Actor-Critic model on reinforcement learning problems that demand both accuracy of control and flexibility. A model that develops from initial volatility to strong convergence will give a lot of insight into the appropriateness of the contexts with a reward structure and difficulties. Such results can be taken further to study the use of Actor-Critic frameworks for different reinforcement learning scenarios to show capabilities of fast learning and policy improvement.

4.1.2. ACROBOT

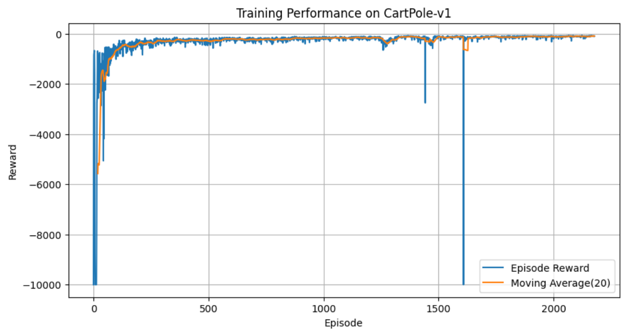

Figure 5. Training Performance of Actor-Critic on Acrobot-v1.

Given that the environment was eventually solved, it is very likely that the performance for the Acrobot-v1 task in Fig5 is far from optimal. The trajectory of training has slow convergence-it takes more than 2000 episodes for the model to achieve a decent average reward of -103.35. Such slow improvements hint at implied difficulty faced by the Actor-Critic framework in acquiring a decent policy for Acrobot-v1, which is known to have high dimensional space with a sparse reward framework.

The episodic rewards are highly unstable, having a standard deviation of 426.85. Large negative rewards frequently show that it is common that the model fails to progressively raise its performance, and sometimes always oscillates between a suboptimal and marginally improved policy. Even though, at the final solution, the model learns, convergence is really very inefficient and variable, reflecting the poor sample efficiency and lack of robustness.

Most importantly, while the Actor-Critic model showed clear improvements and much smoother convergence on the simpler task that is CartPole-v1, it appears incapable of handling the complexity brought on by Acrobot-v1. The absence of well-defined and consistent improvement in the early and mid-training stages could point to the model's needing advance techniques for exploration, more reliable methods of updating the policy, or such frameworks developed for sparse and delayed rewards.

In all, the Actor-Critic model eventually solves the Acrobot-v1 environment, but the very long training time and huge reward variance with initial instability underpin its limitation to cope well with this task. These observations show that there is still a need for improvements either in architecture or algorithms to enhance the model's efficiency and robustness for such challenging control scenarios.

4.1.3. MOUNTAIN CAR

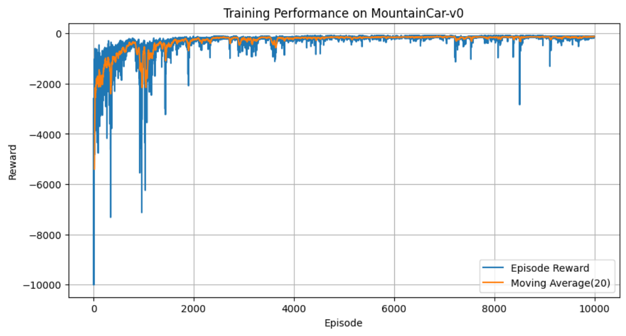

Figure 6. Training Performance of Actor-Critic on MountainCar-v0.

From the experimental results, it can be seen that the AC method was used to train the agent in the MountainCar-v0 environment with a huge improvement in learning performance. The x-axis is the number of training episodes, and the y-axis is the cumulative reward (Reward) per episode. The blue curve is the immediate reward for each episode, and the orange curve is a moving average over 20 episodes.

In the first few episodes of training, the reward curve has a lot of large fluctuations; the reward values fall in highly negative ranges, for example, close to -10000. This reflects that the agent struggled during exploration to identify an effective sequence of actions that could push the car over the hill. Such poor early performance is compatible with the sparsity of rewards characterizing MountainCar-v0: this means that reaching meaningful experience (by extensive trial and error) is necessary for good performance of the agent.

It can be observed that with more training, the moving average curve (orange) shows an obvious upward tendency, while both the episode rewards and the moving average converge to a rather stable region. This is also reflected in the statistical results, with an average episodic return of -118.0 and a standard deviation of 15.79. Compared to earlier results, this reflects a significant reduction in variability and a marked improvement in performance. The average reward of -118.0 means that the agent has learned a quite effective strategy, requiring fewer steps to complete the task and approaching the optimal policy for MountainCar-v0.

Although minor fluctuations still occur throughout the training process, particularly in individual episodes, these are expected due to the stochastic nature of the environment or residual exploration. The low standard deviation suggests that the agent's performance has become highly consistent across episodes, indicating robust policy learning.

4.2. Baseline Performance of DQN

To establish a baseline, we compared the performance of the standard Deep Q-Network (DQN) against a Random Policy across three classic control tasks: CartPole, Acrobot, and Mountain Car. Table 5 summarizes the average rewards and standard deviations obtained by both algorithms on each task.

Table 5. Baseline DQN Performance on Classic Control Tasks

Agent	Average Reward	Standard Deviation
CartPole Random	21.0	8.933
CartPole DQN	1000.0	0.0
Acrobot Random	-968.1	95.7
Acrobot DQN	-88.3	18.623
Mountain Car Random	-1000.0	0.0
Mountain Car DQN	-108.4	1.959

Figure 7, Figure 8, and Figure 9 illustrate the training performance of DQN over epochs for CartPole, Acrobot, and Mountain Car, respectively.

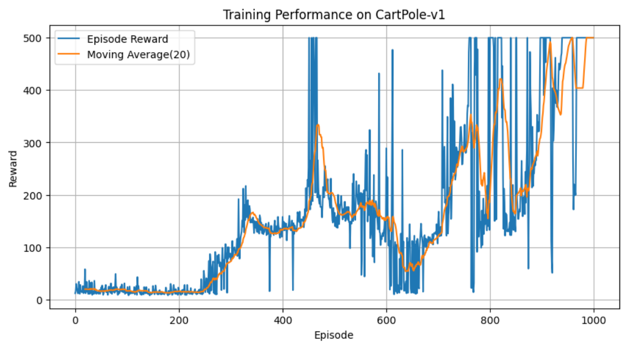

Figure 7. DQN Training Curve for CartPole. The curve shows the reward progression during training.

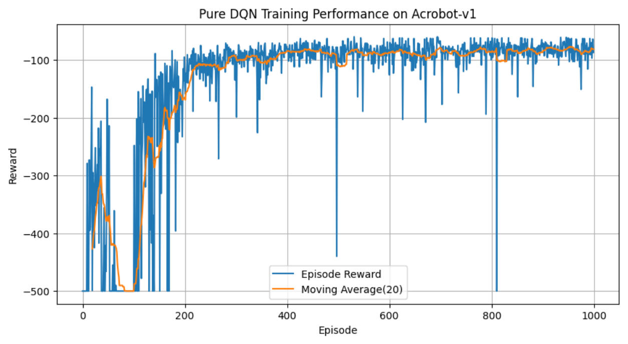

Figure 8. DQN Training Curve for Acrobot. The training curve illustrates the improvement in performance over episodes.

The results clearly demonstrate that DQN significantly outperforms the Random Policy across all three tasks. In CartPole, DQN achieves optimal performance with an average reward of 1000.0 and a standard deviation of 0.0, whereas the Random Policy only obtains an average reward of 21.0. Similarly, for Acrobot and Mountain Car, DQN achieves higher average rewards and lower variability compared to the Random Policy, indicating more consistent and effective learning. Notably, the training curves for CartPole exhibit significant fluctuations, suggesting potential instability during training. This instability is primarily caused by the hyperparameter settings being slightly too large for the simplicity of the task. In contrast, Acrobot and Mountain Car show rapid convergence around 200 epochs.

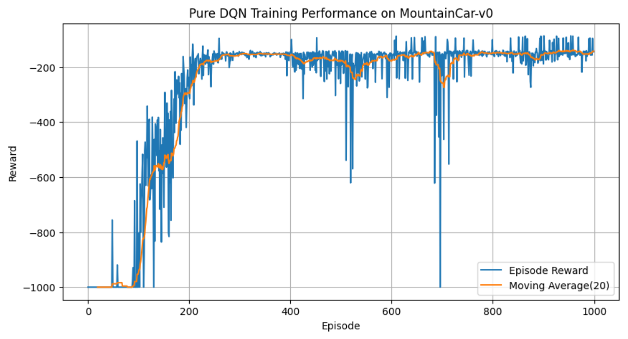

Figure 9. DQN Training Curve for Mountain Car. The reward increases as the agent learns to solve the task.

We can also observe the direct visualizations of the comparisons. In CartPole, the Random Policy fails rapidly, while DQN remains stable until reaching the maximum number of steps. In Acrobot, the Random Policy is unable to achieve the goal, whereas DQN nearly follows the optimal strategy. In Mountain Car, the Random Policy remains stuck in the valley, while DQN effectively utilizes velocity and gravity to reach the peak. All related code, animations, and plots are available in the appendix IPython Notebook file, under the Pure DQN section.

Both the experimental data and the visualizations demonstrate the effectiveness of DQN across these three tasks.

4.3. Double DQN

To evaluate the effectiveness of Double DQN, we keep all parameters the same except for the update where we use Double DQN instead. Table 6 shows their performance.

Analyzing the results in Table 6, we observe that for **CartPole**, both models achieve identical performance with an average reward of 1000.0 and no variability. For **Acrobot**, Double DQN demonstrates better results, achieving a higher average reward and a lower standard deviation, indicating more stable and effective learning. However, for **Mountain Car**, the performance of Double DQN is slightly worse, with a lower average reward and a significantly higher standard deviation, suggesting less consistent learning compared to Pure DQN.

Figure 10, Figure 11, and Figure 12 illustrate the training performance of Double DQN for CartPole, Acrobot, and Mountain Car, respectively.

Table 6. Performance Comparison between Double DQN and Pure DQN on Classic Control Tasks

Agent	Average Reward	Standard Deviation
CartPole DQN	1000.0	0.0
CartPole Double DQN	1000.0	0.0
Acrobot DQN	-88.3	18.623
Acrobot Double DQN	-82.7	9.889
Mountain Car DQN	-108.4	1.959
Mountain Car Double DQN	-128.7	42.067

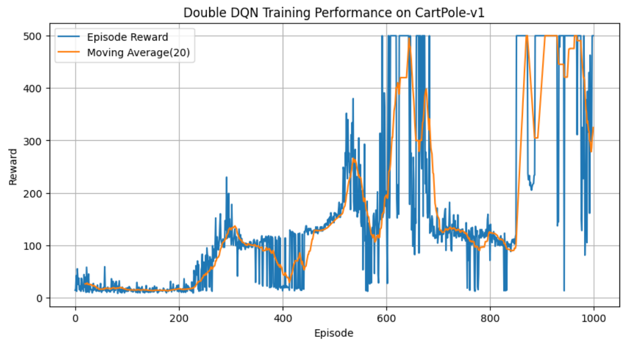

Figure 10. Training Curves: Double DQN vs Pure DQN for CartPole.

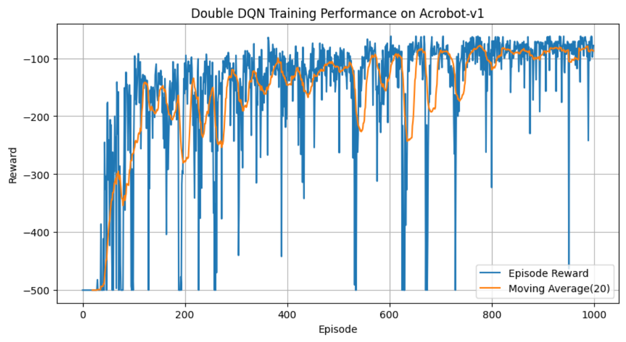

Figure 11. Training Curves: Double DQN vs Pure DQN for Acrobot.

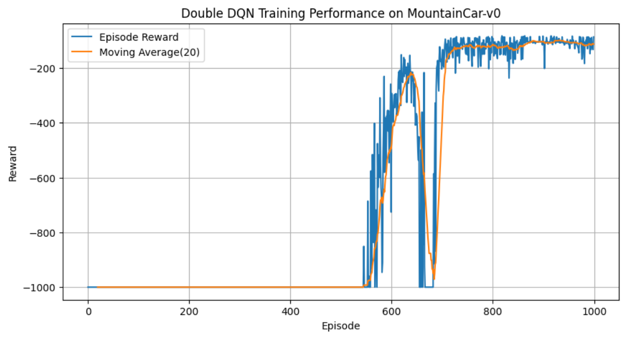

Figure 12. Training Curves: Double DQN vs Pure DQN for Mountain Car.

From the training curves, we observe significant differences in the learning dynamics after applying Double DQN.

For CartPole, as shown in Figure 10, the reward steadily increases and reaches the maximum reward of 500 around episode 600. However, notable fluctuations occur during the training process, particularly between episodes 500 and 700, indicating temporary instability despite eventually stabilizing at the optimal performance.

For Acrobot, Figure 11 shows a clear upward trend where the rewards improve steadily from -500 to approximately -100. The training process is smoother with fewer fluctuations compared to Pure DQN. This aligns with the quantitative results in Table 6, which show higher average rewards and lower standard deviations, demonstrating more stable learning.

For Mountain Car, as seen in Figure 12, the reward remains stagnant around -1000 in the early episodes and begins improving rapidly after episode 500. While the final rewards converge around -130, the learning process exhibits large fluctuations in the later stages. This may suggest that Double DQN struggles to maintain consistency for this task compared to Pure DQN.

The animation results further validate these observations. For CartPole, the agent remains stable but occasionally exhibits unstable swings. For Acrobot, the agent's motion amplitude is larger, enabling it to reach the goal more effectively. In Mountain Car, the car reaches the peak with fewer oscillations compared to Pure DQN, as it just uses a single forward motion when it is near the peak.

4.4. Soft Updates

We investigated the impact of soft updates with different τ values on the performance of DQN. We keep the same params and use soft updates instead of hard updates. Table 7presents the average rewards and standard deviations for various τ settings compared to Pure DQN.

Table 7. Performance of Soft Updates with Different τ Values on Classic Control Tasks

Task	τ	Average Reward	Standard Deviation
CartPole	0.001	337.66	49.828
	0.005	66.88	4.008
	0.01	139.94	4.957
	0.05	9.46	0.779
Acrobot	0.001	-87.16	24.184
	0.005	-81.5	14.780
	0.01	-78.44	15.342
	0.05	-80.9	12.769
Mountain Car	0.001	-130.5	43.333
	0.005	-143.8	9.767
	0.01	-134.58	22.924
	0.05	-144.82	7.287

Algorithm 2 Shaping Reward for CartPole

```
1:  Input: $reward, state, next\_state, action, done$
2:  Extract: $pole\_angle \leftarrow state[2], next\_pole\_angle \leftarrow next\_state[2]$
3:  $shaping\_reward \leftarrow 0.0$
4:  if ($action == 1$ and $pole\_angle > 0$) or ($action == 0$ and $pole\_angle < 0$) then
5:      $shaping\_reward \leftarrow 0.1$
6:  end if
7:  if $done$ then
8:      $shaping\_reward \leftarrow shaping\_reward - 1.0$
9:  end if
10: Return: $reward + shaping\_reward$
```

Table 7 shows that for CartPole, performance decreases as τ increases, with the best result (337.66) at $\tau = 0.001$ and a sharp drop at $\tau = 0.05$. For Acrobot, $\tau = 0.01$ achieves the best performance (-78.44) with low variability, while smaller τ values are also competitive. In Mountain Car, results are less consistent; $\tau = 0.001$ performs best (-130.5) but with high variability, whereas larger τ values lead to slightly worse rewards. Overall, smaller τ values generally yield better results, particularly in CartPole and Acrobot.

Compared to Pure DQN, Acrobot benefits from soft updates, achieving better performance with lower variability. However, for CartPole, the model fails to converge, and Mountain Car shows overall worse performance with higher variability, indicating that soft updates do not improve learning in these two tasks.

4.5. Reward Shaping Functions

To improve the learning efficiency, we apply task-specific reward shaping for CartPole, Acrobot, and Mountain Car. The pseudo-code for the reward shaping functions is provided below:

This CartPole shaping reward function (Algorithm 2) (Hu et al., 2020) encourages actions that stabilize the pole by providing a small positive reward (+0.1) when the chosen action aligns with the pole's tilt direction. If the episode ends, a penalty of -1.0 is applied.

The Acrobot shaping reward function (Algorithm 3) rewards height gains by comparing the current and next heights of the pendulum, while penalizing excessive angular velocities. An additional reward (+1.0) is given when the episode completes successfully.

The Mountain Car shaping reward function (Algorithm 4) guides the agent towards the goal position (0.5) by reducing the distance to the goal. A potential-based shaping reward is applied, incorporating a discount factor ($\gamma = 0.99$) to encourage forward progress.

Algorithm 3 Shaping Reward for Acrobot

```
1: Input: $reward, state, next\_state, done$
2: Extract: current and next heights based on state and angles
3: $height\_shaping\_reward \leftarrow \max(0, next\_height - current\_height)$
4: $velocity\_penalty \leftarrow -0.01 \cdot (|theta1\_dot| + |theta2\_dot|)$
5: if $done$ then
6:     $height\_shaping\_reward \leftarrow height\_shaping\_reward + 1.0$
7: end if
8: Return: $reward + height\_shaping\_reward + velocity\_penalty$
```

Algorithm 4 Shaping Reward for Mountain Car

```
1: Input: $reward, state, next\_state, done$
2: Define: $goal\_position \leftarrow 0.5$
3: $\phi_{current} \leftarrow |goal\_position - state[0]|$
4: $\phi_{next} \leftarrow |goal\_position - next\_state[0]|$
5: $\gamma \leftarrow 0.99$
6: $shaping \leftarrow \gamma \cdot \phi_{next} - \phi_{current}$
7: Return: $reward + shaping$
```

Table 8 shows that shaping rewards significantly underperform on CartPole, where the model fails to converge. In contrast, for Acrobot and Mountain Car, shaping rewards improve performance, achieving higher average rewards with reduced variability.

The training results for shaping rewards show distinct outcomes across tasks. For CartPole, the model fails to converge, with rewards fluctuating significantly throughout training. In contrast, Acrobot demonstrates steady improvement, achieving higher rewards and stable performance. For Mountain Car, the model quickly progresses from low rewards to consistent performance, showing effective learning with shaping rewards.

4.6. Overall Discussion

Table 9 shows the overall performance of all methods tested in the three tasks, providing a very fine-grained view of the performance of different reinforcement learning approaches-

Table 8. Shaping Rewards DQN Performance on Classic Control Tasks

Task	Average Reward	Standard Deviation
CartPole	72.5	29.254
Acrobot	-84.3	11.180
Mountain Car	-109.3	8.222

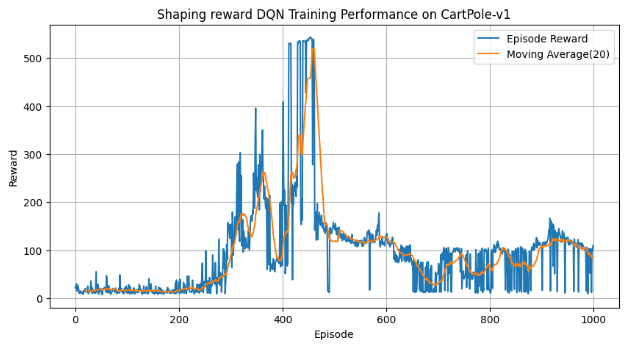

Figure 13. Shaping Reward Training Curve for CartPole.

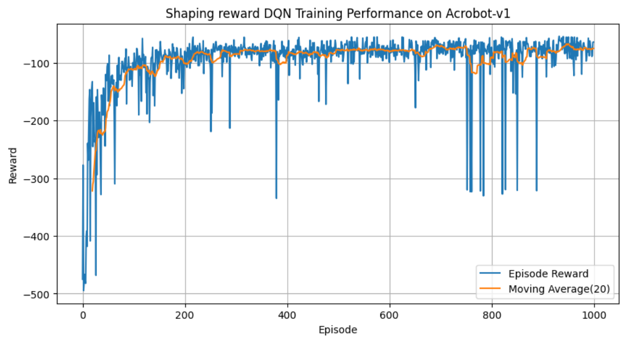

Figure 14. Shaping Reward Training Curve for Acrobot.

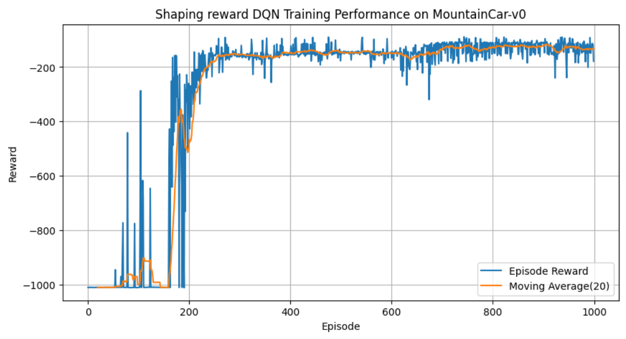

Figure 15. Shaping Reward Training Curve for Mountain Car.

Table 9. Summary of overall performance for all methods on classic control tasks.

Task	Method	Average Reward	Standard Deviation
CartPole	Actor-Critic	1000.0	0.0
	Pure DQN	1000.0	0.0
	Double DQN	1000.0	0.0
	Soft Updates	337.66	49.83
	Shaping Rewards	72.5	29.25
Acrobot	Actor-Critic	-348.0	426.83
	Pure DQN	-88.3	18.62
	Double DQN	-82.7	9.88
	Soft Updates	-78.44	15.34
	Shaping Rewards	-84.3	11.18
Mountain Car	Actor-Critic	-118.0	15.78
	Pure DQN	-108.4	1.96
	Double DQN	-128.7	42.07
	Soft Updates	-130.5	43.33
	Shaping Rewards	-109.3	8.22

Actor-Critic, DQN, Double DQN, Soft Updates, and Reward Shaping-on CartPole, Acrobot, and Mountain Car. Such results outline the connection of task complexity, reward settings, and algorithmic structure and provide deep insight into advantages and disadvantages of each approach.

The CartPole environment is a representative example of a simple, fully observable control problem with a deterministic reward system. The Actor-Critic, Pure DQN, and Double DQN algorithms reach the optimal performance and achieve an average reward of 1000.0 with zero variance. Such a result points to the ability of these methods to quickly identify and exploit the deterministic nature of the problem and converge to a stable policy with minimal computational cost. By contrast, Soft Updates and Reward Shaping drastically lower performance. While Soft Updates do have the theoretical benefit of allowing the target network to be smoothly updated, they do not converge because of inappropriate τ values. A lower τ encourages stability but slows down convergence while higher τ values amplify instability-as can be seen by the rapid deterioration in performance. Similarly, Reward Shaping disrupts the innate reward structure of CartPole. The additional feedback created by shaping disturbs the process of deterministic policy optimization; the model deviates from the optimal solution. From these results, one might conclude that for relatively simple environments with well-defined reward dynamics, classic DQN frameworks already suffice for performance that is good enough and thus superfluous changes to the framework could bring about unnecessary complexity.

In contrast, the Acrobot environment highlights the challenges involved in sparse rewards and high-dimensional state-action spaces. Among the four, the Actor-Critic performs the worst, with an average reward of -348.0 and a large standard deviation of 426.83, demonstrating clearly that this model is quite insufficiently sample-efficient and has lots of trouble navigating through sparse reward spaces. Due to the approximations of value functions and the policy gradient updates, which are inherently used in the Actor-Critic method, the model has a high chance of facing variance, causing it to frequently fluctuate. In contrast, Pure DQN significantly improves this, lowering the average reward to -88.3, while Double DQN further improves stability and reaches an average reward of -82.7 with lower variance. The better performance of Double DQN can be explained by its elimination of the overestimation bias inherent in Q-learning, which is accentuated for problems like Acrobot with sparse and delayed rewards. Most notably, Reward Shaping provides intermediate improvements, yielding an average reward of -84.3. Through providing task-specific feedback in terms of height gains and velocity penalties, shaping rewards facilitate much faster learning and guide the agent towards meaningful state exploration. This indicates that in a sparse reward environment, shaping functions can be a good complement to conventional reinforcement learning frameworks for fast convergence.

Although the Mountain Car environment increases the difficulty of sparse and delayed rewards, all of the methods perform well. Actor-Critic performs well, with an average reward of -118.0 and slightly higher deviation. The model benefits from its ability to leverage policy gradient updates, which are less susceptible to Q-learning's overestimation bias. Pure DQN is the best, which can be represented by an average reward of -108.4 and nearly zero standard deviation. Double DQN and Softupdates are highly volatile, yielding an average reward around -130, with a much larger standard deviation. They fail to outperform Pure DQN in the Mountain Car problem indicates that the benefits due to overestimation correction are limited to settings with sparse and delayed rewards. Moreover, while theoretically appealing for stabilization, Soft Updates do not converge well in this framework, since inappropriate τ values amplify learning instability. Reward Shaping enjoys moderate success: an average reward of -109.3 with reduced variability. By implementing a potential-based reward shaping that encourages progress toward goals, reward shaping mitigates the challenge of exploration due to sparse rewards and allows the agent to better learn strategies that exploit momentum and gravity.

The most salient patterns across these tasks are strong interactions between task complexity, reward sparsity, and algorithmic design. In simple and well-structured environments like CartPole, traditional algorithms like DQN and Actor-Critic achieve maximum performance without requiring additional improvements. The deterministic nature of the task allows these models to converge efficiently. Approaches like Soft Updates and Reward Shaping may introduce unneeded instability. However, in tasks with sparse rewards, like Acrobot and Mountain Car, the flaws of these methods are more evident. While Actor-Critic suffers from problems concerning variance and sample inefficiency, Pure DQN and Double DQN both suffer from instability induced by the delayed feedback problem. Reward Shaping now appears to be a plausible remedy for these challenges: domain-specific feedback aids exploration, driving the agent towards significant states. The key factor that has emerged is that the success of reward shaping lies in thorough design; inadequately structured feedback can block or hamper, rather than enhance, the learning process.

Empirical results also highlight the limitations of existing reinforcement learning methods in terms of solving sparse and delayed reward settings. While Double DQN outperforms Pure DQN remarkably on Acrobot, its inconsistent performance on Mountain Car strengthens the requirement for superior bias correction techniques. Similarly, the failure of Soft Updates in generalizing performance over multiple tasks further underlines the relevance of hyperparameter sensitivity, particularly to determine appropriate τ values. Results confirm that while current methodologies are effective in deterministic environments, they are still insufficient for more challenging tasks characterized by sparse rewards and delayed feedback. Such gaps can only be bridged by using a combination of improved exploration methods, bias correction techniques, and reward shaping for the task at hand.

5. Conclusion

In this work, we have compared Actor-Critic, DQN, Double DQN, Soft Updates, and Reward Shaping on the classical control tasks of CartPole, Acrobot, and Mountain Car. It turns out that the results are highly dependent on the task, including the reward structure, task complexity, and algorithmic design. Actor-Critic and DQN-based methods find optimal solutions in simple environments like CartPole but fail to perform well in handling sparse and delayed rewards in Acrobot and Mountain Car. Double DQN demonstrates substantial gains in Acrobot by overcoming overestimation bias, and Reward Shaping seems a promising approach for accelerating learning in sparse reward environments.

The results bring into light the wide challenges faced in the reinforcement learning field, especially in the context of delayed rewards and high-dimensional state-action spaces. Standard methods, while effective on simple tasks, are limited in their ability to handle sparse feedback and to perform meaningful exploration. Future research efforts will seek to address these limitations along the following lines:

1. Improved Exploration Techniques: Methods such as intrinsic motivation, curiosity-driven exploration, and entropy regularization can provide richer feedback signals, enabling agents to navigate sparse reward landscapes more effectively.

2. Stabilized Learning Frameworks: Enhancing methods like Soft Updates with adaptive τ tuning or integrating advanced policy gradient algorithms (e.g., PPO, SAC) can improve learning stability and convergence.

3. Generalized Reward Shaping: The development of systematic frameworks for task-agnostic reward shaping will make it possible for agents to exploit intermediate feedback while preserving the native reward structure. Hybrid methodologies combining value-based and policy-based techniques can exploit the merits brought by both paradigms to improve sample efficiency and learning stability.

4. Transfer Learning and Meta-RL: Investigating the transferability of learned policies across tasks and environments can reduce training time and improve generalization.

In summary, this study provides a wide review of reinforcement learning techniques in relation to different levels of task complexity, and it discusses their advantages and disadvantages. Addressing the challenges outlined can help upcoming research improve the resilience and applicability of reinforcement learning, therefore allowing for the development of more efficient solutions in complex and dynamic environments.

6. Contribution

- Faquan Wang: Responsible for the ACTOR-CRITIC portion of the experiment and related data collection. Discussion of some of the experimental data.
- Hao Yuan: The DQN and enhancement experiments, related experimental sections, and the overall structure and revisions of the paper.
- Yixuan Liu: Developed and trained a DQN-based Mountaineer model and wrote part of the paper.
- Yao Jia: Responsible for the ACTOR-CRITIC portion of the MountainCar experiment and related data collection. Discussion of some of the experimental data.

References

Bellman, R. *Dynamic Programming*. Princeton University Press, 1957.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1587–1596, 2018.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1861–1870, 2018.

Hu, Y., Wang, W., Jia, H., Wang, Y., Chen, Y., Hao, J., Wu, F., and Fan, C. Learning to utilize shaping rewards: A new approach of reward shaping. *CoRR*, abs/2011.02669, 2020. URL https://arxiv.org/abs/2011.02669.

Konda, V. R. and Tsitsiklis, J. N. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, pp. 1008–1014, 2000.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. Continuous control with deep reinforcement learning. *arXiv preprint*, arXiv:1509.02971, 2015. URL https://arxiv.org/abs/1509.02971.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013. URL https://arxiv.org/abs/1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

Ng, A. Y., Harada, D., and Russell, S. J. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287, 1999.

OpenAI. Dota 2 with large scale deep reinforcement learning. *arXiv preprint*, arXiv:1912.06680, 2019. URL https://arxiv.org/abs/1912.06680.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575: 350–354, 2019. doi: 10.1038/s41586-019-1724-z.

Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992. doi: 10.1007/BF00992696.