A Comparative Study of DQN-Family Algorithms on Gymnasium Control Tasks

Qi Deng [*1] **Zicheng Fan** [*1] **Kaiming Zhan** [*1]

Abstract

Deep Q-Networks (DQN) have become a cornerstone of value-based deep reinforcement learning, yet suffer from overestimation bias and inefficient value function representation. This paper presents a comparative empirical study of three DQN-family algorithms—vanilla DQN, Double DQN, and Dueling DQN—on canonical control benchmarks from the Gymnasium suite. We provide detailed algorithmic descriptions with pseudocode for each variant, highlighting their orthogonal design improvements: Double DQN decouples action selection and evaluation to mitigate overestimation, while Dueling DQN decomposes the Q-network into separate value and advantage streams to accelerate learning. Under matched hyperparameters and training protocols on CartPole-v3 and LunarLander-v3 tasks, our experiments demonstrate a consistent performance ranking where Dueling-DQN outperforms Double-DQN, which in turn outperforms vanilla DQN, in terms of stability, sample efficiency, and final return. These results validate the complementary benefits of both improvements and offer practical guidance for practitioners selecting value-based RL methods for discrete action spaces. Code is available at `https://github.com/Hyrsta/Q-learning`.

1. Introduction

Deep reinforcement learning (RL) has achieved remarkable success across diverse domains, from mastering complex games (Mnih et al., 2015) to robotic control (Levine et al., 2016) and autonomous decision-making. At the heart of many of these advances lies value-based learning (Sutton et al., 1998), where an agent learns to estimate the long-term expected return (value) of taking actions in given states and derives a policy by acting greedily with respect to these learned values. Among value-based methods, Deep Q-Networks (DQN) (Mnih et al., 2015) represent a landmark breakthrough, combining classical Q-learning with deep neural network function approximation and two key stabilization techniques: experience replay and target networks. Despite its success, vanilla DQN is known to suffer from overestimation bias (Thrun & Schwartz, 2014) in Q-value updates, which can degrade sample efficiency and policy quality. Subsequent variants—Double DQN (van Hasselt et al., 2015) and Dueling DQN (Wang et al., 2015)—address these limitations through orthogonal improvements: Double DQN mitigates overestimation by decoupling action selection and evaluation, while Dueling DQN redesigns the network architecture to separately model state values and action advantages.

This paper provides a comparative empirical study of these three DQN-family algorithms on canonical control benchmarks from the Gymnasium suite. Our contributions are threefold: (1) we present clear algorithmic descriptions and pseudocode for DQN, Double DQN, and Dueling DQN, highlighting their key differences and design rationales; (2) we evaluate all three methods under matched hyperparameters and training protocols to ensure fair comparison; and (3) we analyze their relative performance in terms of stability, sample efficiency, and final return, offering practical insights into when each variant provides tangible benefits. The remainder of this section describes the three algorithms in detail. Section 2 outlines our experimental setup, Section 3 presents the results, and Section 4 concludes with a discussion of the findings.

1.1. Deep Q-Network with Experience Replay

Q-learning is one of the earliest and most classical value-based methods in Reinforcement Learning (RL), originally proposed by Watkins and Dayan (Watkins & Dayan, 1992). It enables an agent to learn an optimal policy through trial-and-error interactions with the environment without requiring prior knowledge of the environment's transition probabilities. The algorithm is formulated under the framework of a Markov Decision Process (MDP) (Sutton et al., 1998), where decision-making scenarios are represented by states

[*]Equal contribution [1]School of Computing and Intelligence Innovation, Fudan University. Correspondence to: Qi Deng <25213050008@m.fudan.edu.cn>, Zicheng Fan <25213050157@m.fudan.edu.cn>, Kaiming Zhan <25213050555@m.fudan.edu.cn>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

Algorithm 1 DQN with Experience Replay

Input: environment Env, replay capacity N, batch size B, discount γ, target update period C, learning rate α, exploration schedule ϵ_t
Initialize replay buffer $\mathcal{D} \leftarrow \{\}$ of capacity N
Initialize online Q-network $Q(s, a; \theta)$ with random weights θ
Initialize target network weights $\theta^- \leftarrow \theta$
for episode = 1 **to** M **do**
  $s \leftarrow$ Env.reset()
  for t = 1 **to** T **do**
    With probability ϵ_t select a random action a, otherwise $a \leftarrow \arg\max_{a'} Q(s, a'; \theta)$
    Execute a, observe reward r and next state s'
    Store transition (s, a, r, s') in $\mathcal{D}$ (drop oldest if full)
    if $|\mathcal{D}| \geq B$ **then**
      Sample random batch $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^{B}$ from $\mathcal{D}$
      For each sample compute target:

$$y_j = \begin{cases} r_j & \text{if } s'_j \text{ is terminal} \\ r_j + \gamma \max_{a'} Q(s'_j, a'; \theta^-) & \text{otherwise} \end{cases}$$

      Perform a gradient step on loss $\mathcal{L}(\theta) = \frac{1}{B} \sum_j (y_j - Q(s_j, a_j; \theta))^2$
    end if
    Every C steps: $\theta^- \leftarrow \theta$
    $s \leftarrow s'$
  end for
end for

(s) and actions (a).

Algorithm 1 summarizes the core training loop of DQN with experience replay. The agent interacts with the environment by selecting actions via an ε-greedy policy: with probability ε_t it explores randomly, otherwise it exploits the current Q-network by choosing $\arg\max_{a'} Q(s, a'; \theta)$. Each observed transition (s, a, r, s') is stored in a replay buffer $\mathcal{D}$ of fixed capacity N; when the buffer is full, the oldest transitions are discarded. Once sufficient data has been collected ($|\mathcal{D}| \geq B$), the algorithm samples a mini-batch of transitions uniformly at random and computes target values y_j using a separate target network with frozen parameters θ^-. The online network parameters θ are then updated by minimizing the mean-squared temporal-difference error. Periodically (every C steps), the target network is synchronized with the online network to ensure stable bootstrapping. This decoupling of action selection and target evaluation, combined with experience replay to break temporal correlations, enables DQN to learn robust value functions even in complex, high-dimensional state spaces.

1.2. Double Deep Q-Network (Double DQN)

Double DQN (van Hasselt et al., 2015) reduces the overestimation bias of the original DQN by decoupling the action selection and action evaluation between the online network and the target network, building upon the earlier Double Q-learning algorithm (Hasselt, 2010). The pseudocode below shows this modification to the target computation while keeping the overall training loop and replay buffer logic identical to standard DQN.

Algorithm 2 presents the Double DQN variant, which addresses the overestimation bias inherent in standard DQN. The key difference lies in the target computation: whereas vanilla DQN uses $\max_{a'} Q(s'_j, a'; \theta^-)$ to both select and evaluate the best action, Double DQN decouples these two operations. Specifically, the online network $Q(s, a; \theta)$ is used to select the greedy action $\arg\max_a Q(s'_j, a; \theta)$, while the target network $Q(s, a; \theta^-)$ evaluates the value of that selected action. This decomposition prevents the max operator from consistently picking overestimated values, as the selection and evaluation are performed by different networks with different parameterizations. All other components—experience replay, ε-greedy exploration, and periodic target network updates—remain identical to standard DQN. Empirically, this simple modification yields more accurate Q-value estimates and improved sample efficiency, particularly in environments where overestimation can lead to suboptimal policies or training instabilities.

1.3. Dueling Deep Q-Network (Dueling DQN)

Dueling DQN (Wang et al., 2015) separates the representation of state-value and advantage for each action, which

Algorithm 2 Double DQN with Experience Replay

Input: environment Env, replay capacity N, batch size B, discount γ, target update period C, learning rate α, exploration schedule ϵ_t
Initialize replay buffer $\mathcal{D} \leftarrow \{\}$ of capacity N
Initialize online Q-network $Q(s, a; \theta)$ with random weights θ
Initialize target network weights $\theta^- \leftarrow \theta$
for episode = 1 **to** M **do**
  $s \leftarrow$ Env.reset()
  for t = 1 **to** T **do**
    With probability ϵ_t select a random action a, otherwise $a \leftarrow \arg\max_{a'} Q(s, a'; \theta)$
    Execute a, observe reward r and next state s'
    Store transition (s, a, r, s') in $\mathcal{D}$ (drop oldest if full)
    if $|\mathcal{D}| \geq B$ **then**
      Sample random batch $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^{B}$ from $\mathcal{D}$
      For each sample compute Double DQN target:

$$y_j = \begin{cases} r_j & \text{if } s'_j \text{ is terminal} \\ r_j + \gamma\, Q\big(s'_j, \arg\max_a Q(s'_j, a; \theta); \theta^-\big) & \text{otherwise} \end{cases}$$

      Perform a gradient step on loss $\mathcal{L}(\theta) = \frac{1}{B} \sum_j (y_j - Q(s_j, a_j; \theta))^2$
    end if
    Every C steps: $\theta^- \leftarrow \theta$
    $s \leftarrow s'$
  end for
end for

helps the agent learn which states are (or are not) valuable without having to learn the effect of each action for every state. The architecture splits the final layers of the Q-network into two streams that estimate a scalar state-value $V(s)$ and an advantage vector $A(s, a)$; these are combined to produce Q-values via $Q(s, a) = V(s) + A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a')$.

The pseudocode below shows a dueling variant that uses experience replay and a target network; it otherwise follows the same training loop as the DQN family (optionally usable together with Double DQN selection/evaluation).

Algorithm 3 introduces the Dueling DQN architecture, which represents a complementary improvement orthogonal to Double DQN's target-computation fix. Unlike the previous variants that modify the learning target, Dueling DQN redesigns the network architecture itself. The key innovation is the decomposition of the Q-network into two parallel streams: a value stream that estimates the state-value function $V(s; \theta)$ (measuring how good it is to be in a given state), and an advantage stream that estimates the advantage function $A(s, a; \theta)$ (measuring the relative benefit of each action over the average). These streams share a common convolutional or fully-connected trunk for feature extraction and are recombined via $Q(s, a; \theta) = V(s; \theta) + A(s, a; \theta) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta)$, where the mean-subtraction normalization ensures identifiability. This architectural change enables the network to learn which states are inherently valuable independently of the action-conditioned advantages, accelerating learning in scenarios where many actions yield similar outcomes. Importantly, the dueling architecture is compatible with both standard DQN and Double DQN target computations, as shown in the algorithm pseudocode: one can use either the vanilla max-based target or the Double DQN decoupled target. The gradient updates backpropagate through both value and advantage heads, jointly optimizing the shared representation. This modularity allows Dueling DQN to combine with Double DQN for cumulative benefits—mitigating overestimation while improving generalization across actions.

2. Experimental Setup

2.1. Implementation Details

All experiments were conducted on a single workstation equipped with an Intel Xeon Gold 6326 CPU (2.90 GHz), an NVIDIA RTX A6000 GPU (48 GB VRAM), and Ubuntu 20.04.4 LTS. To ensure reproducibility and eliminate potential performance interference, we ran all training jobs in isolation with no concurrent GPU workloads. Our implementation builds upon PyTorch (Paszke et al., 2019) and uses the Gymnasium API (Towers et al., 2024) (version 0.29.1) for environment interfacing.

Algorithm 3 Dueling DQN with Experience Replay

Input: environment Env, replay capacity N, batch size B, discount γ, target update period C, learning rate α, exploration schedule ϵ_t
Initialize replay buffer $\mathcal{D} \leftarrow \{\}$ of capacity N
Initialize online dueling Q-network: shared trunk; value head $V(s;\theta)$; advantage head $A(s,a;\theta)$; combine to $Q(s,a;\theta)$ via advantage normalization
Initialize target network weights $\theta^- \leftarrow \theta$
for episode = 1 **to** M **do**
  $s \leftarrow$ Env.reset()
  for t = 1 **to** T **do**
    With probability ϵ_t select a random action a, otherwise $a \leftarrow \arg\max_{a'} Q(s,a';\theta)$
    Execute a, observe reward r and next state s'
    Store transition (s,a,r,s') in $\mathcal{D}$ (drop oldest if full)
    if $|\mathcal{D}| \geq B$ **then**
      Sample random batch $\{(s_j,a_j,r_j,s'_j)\}_{j=1}^{B}$ from $\mathcal{D}$
      For each sample compute target y_j (use either DQN or Double DQN style target):

$$y_j = \begin{cases} r_j & \text{if } s'_j \text{ is terminal} \\ r_j + \gamma \max_{a'} Q(s'_j, a'; \theta^-) & \text{otherwise} \end{cases}$$

      Note: when combining with Double DQN, replace the $\max$ evaluation by

$$r_j + \gamma\, Q\big(s'_j, \arg\max_{a} Q(s'_j, a; \theta); \theta^-\big)$$

      Perform a gradient step on loss $\mathcal{L}(\theta) = \frac{1}{B}\sum_j (y_j - Q(s_j,a_j;\theta))^2$; gradients backpropagate through both value and advantage heads
    end if
    Every C steps: $\theta^- \leftarrow \theta$
    $s \leftarrow s'$
  end for
end for

2.2. Task Environments

We evaluate on two canonical control benchmarks using the Gymnasium API with ale-py:

CartPole-v3. A classic control task with a 4-dimensional continuous observation vector and a discrete 2-action space (push left/right). Episodes terminate or truncate according to the environment's default criteria; all default physics and termination settings are retained.

LunarLander-v3 (discrete). A 2-D lander with an 8-dimensional continuous observation vector and a discrete 4-action space (do nothing; left/right engine; main engine). Episodes terminate upon crash or successful landing; time-limit truncation follows the environment's default. We keep all default reward shaping and environment parameters in both tasks.

2.3. Training Configuration

We train value-based agents from the DQN family with standard experience replay and a target network. For the loss, we adopt the Huber loss in all settings, as it is empirically more robust to outliers and stabilizes Q-learning updates relative to mean-squared error. To ensure reproducibility and comparability, hyperparameters follow widely used community baselines, varying only where task-specific totals or memory/mini-batch sizes are customary. Exploration is ε-greedy with linear decay from the initial to the final value over the specified exploration fraction of total training steps.

2.4. Evaluation Protocol

We report learning curves as the episodic return obtained from deterministic evaluation rollouts ($\epsilon = 0$) at fixed training intervals. At each interval, we execute an evaluation rollout with the current policy and record its episodic return; the curve is the sequence of these values over training. No smoothing is applied and all results are from a single random seed unless stated otherwise.

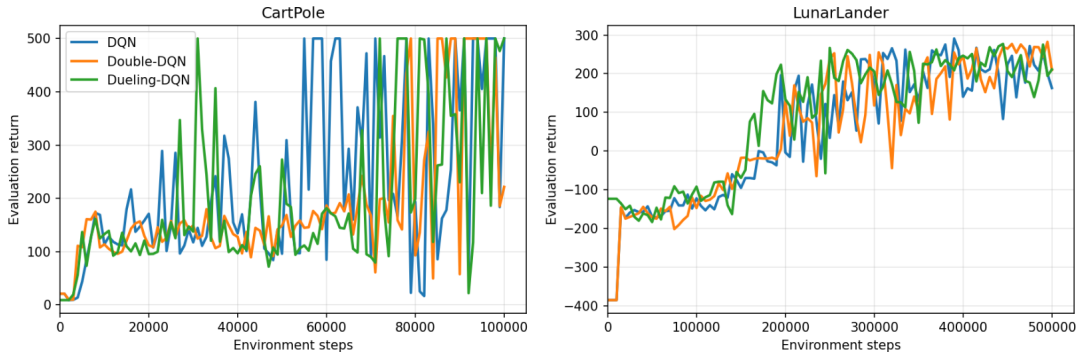

Figure 1. Learning curves on CartPole-v3 and LunarLander-v3 (discrete). Shaded regions indicate variability across random seeds.

Table 1. Training hyperparameters. CartPole-v3 and LunarLander-v3 (discrete) follow community baseline configurations; Huber loss is used throughout.

Hyperparameter	**CartPole-v3**	**LunarLander-v3 (discrete)**
`total_timesteps`	100000	500000
`learning_rate`	5×10^{-4}	5×10^{-4}
`buffer_size`	50000	200000
`batch_size`	64	128
`gamma`	0.99	0.99
`train_frequency`	1	4
`gradient_steps`	1	1
`target_update_interval`	1000	4000
`target_update_tau`	1.0	1.0
`learning_starts`	1000	10000
`exploration_fraction`	0.20	0.40
`exploration_initial_eps`	1.00	1.00
`exploration_final_eps`	0.05	0.02
`loss`	Huber	Huber

3. Results

Figure 1 shows the learning curve across both tasks. Under identical hyperparameters, our runs exhibit a consistent ranking DQN $<$ Double-DQN $<$ Dueling-DQN: DQN improves but displays noticeable oscillations and occasional regressions; Double-DQN dampens these fluctuations and is more sample-efficient; Dueling-DQN accelerates early learning and achieves the highest plateau with the most stable trajectory. The gap is most pronounced in the harder phases of training, where the variants sustain steadier progress toward convergence. For qualitative results (videos) from the optimized models for each algorithm and environment, please refer to the project repository: https://github.com/Hyrsta/Q-learning

4. Conclusion

We presented a systematic empirical comparison of three DQN-family algorithms—vanilla DQN, Double DQN, and Dueling DQN—on two canonical control tasks from the Gymnasium suite. Under identical hyperparameters and training protocols, our experiments reveal a consistent performance ordering: Dueling DQN outperforms Double DQN, which in turn surpasses vanilla DQN, across metrics of stability, sample efficiency, and final return.

These results provide empirical validation for the theoretical motivations underlying each algorithmic improvement. Double DQN's decoupled action selection and evaluation successfully mitigates the overestimation bias inherent in standard Q-learning, while Dueling DQN's architectural decomposition of state values and action advantages enables more efficient generalization across the action space. Importantly, our controlled experimental setup—using matched optimizers, Huber loss, and community-standard hyperparameters—ensures that observed performance gains reflect the core algorithmic innovations rather than confounding implementation details.

Future work could extend this analysis in several directions: evaluating the combined Dueling Double DQN architecture to assess whether the improvements compose additively; testing on more challenging domains with higher-dimensional observation spaces or larger action sets; incorporating additional improvements such as prioritized experience replay (Schaul et al., 2015) or the full Rainbow (Hessel et al., 2018) combination; and investigating the sensitivity of these rankings to hyperparameter choices and random seed variation through comprehensive ablation studies.

References

Hasselt, H. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end

training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Sutton, R. S., Barto, A. G., et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Thrun, S. and Schwartz, A. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 connectionist models summer school*, pp. 255–263. Psychology Press, 2014.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.

Wang, Z., de Freitas, N., and Lanctot, M. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015. URL http://arxiv.org/abs/1511.06581.

Watkins, C. J. C. H. and Dayan, P. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.