Submission and Formatting Instructions for International Conference on Machine Learning (ICML 2025)

Firstname1 Lastname1 [*1] **Firstname2 Lastname2** [*1 2] **Firstname3 Lastname3** [2] **Firstname4 Lastname4** [3]
Firstname5 Lastname5 [1] **Firstname6 Lastname6** [3 1 2] **Firstname7 Lastname7** [2] **Firstname8 Lastname8** [3]
Firstname8 Lastname8 [1 2]

Abstract

This document provides a basic paper template and submission guidelines. Abstracts must be a single paragraph, ideally between 4–6 sentences long. Gross violations will trigger corrections at the camera-ready phase.

1. Introduction

1.1. Deep Q-Network with Experience Replay

The following pseudocode gives a concise implementation flow for a Deep Q-Network (DQN) using an experience replay buffer and a target network. It is written in the standard `algorithm` / `algorithmic` environments and suitable as a high-level recipe for experiments.

Notes. Typical practical details: use RMSProp/Adam for optimization; clip rewards or normalize observations when appropriate; use a separate periodically-updated target network to stabilise learning; tune replay capacity N and batch size B; optionally use prioritized replay or Double DQN extensions.

1.2. Double Deep Q-Network (Double DQN)

Double DQN reduces the overestimation bias of the original DQN by decoupling the action selection and action evaluation between the online network and the target network. The pseudocode below shows this modification to the target computation while keeping the overall training loop and replay buffer logic identical to standard DQN.

[*]Equal contribution [1]Department of XXX, University of YYY, Location, Country [2]Company Name, Location, Country [3]School of ZZZ, Institute of WWW, Location, Country. Correspondence to: Firstname1 Lastname1 <first1.last1@xxx.edu>, Firstname2 Lastname2 <first2.last2@www.uk>.

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

Notes on Double DQN. Compared to vanilla DQN, Double DQN uses the online network θ to select the argmax action and the target network θ^- to evaluate its value, which empirically reduces overestimation and often improves stability. Other recommended improvements (e.g., prioritized replay, dueling architecture, n-step returns, Huber loss) are compatible with this change.

1.3. Dueling Deep Q-Network (Dueling DQN)

Dueling DQN separates the representation of state-value and advantage for each action, which 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).

Notes on Dueling DQN. The dueling architecture often speeds learning of state values and leads to more stable policies, particularly in environments where many actions have similar effects. Ensure advantage normalization (subtracting the mean advantage) when reconstructing Q-values to guarantee identifiability of V and A. Dueling is orthogonal to other improvements (Double DQN, prioritized replay, n-step returns) and can be combined with them.

2. Experimental Setup

2.1. Hardware and System

All experiments were executed on a single workstation with the following configuration:

- **CPU:** Intel Xeon Gold 6326 @ 2.90 GHz
- **GPU:** NVIDIA RTX A6000

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

- **OS:** Ubuntu 20.04.4 LTS

No other concurrent training jobs were scheduled on the GPU during runs.

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.

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

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

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.

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

This study evaluated value-based deep reinforcement learning algorithms; DQN, Double-DQN, and Dueling-DQN on CartPole-v3 and LunarLander-v3 (discrete) under matched hyperparameters and a common training protocol. Across both tasks, the results consistently rank DQN $<$ Double-DQN $<$ Dueling-DQN in terms of stability, sample efficiency, and attained return. These outcomes align with established explanations: Double-DQN mitigates overestimation bias, while the dueling architecture separates state value and advantage to improve value estimation efficiency. Using a unified setup (identical optimizers, Huber loss, and community-baseline hyperparameters) strengthens comparability and offers a clean empirical signal for the incremental benefits of these design choices.

References

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

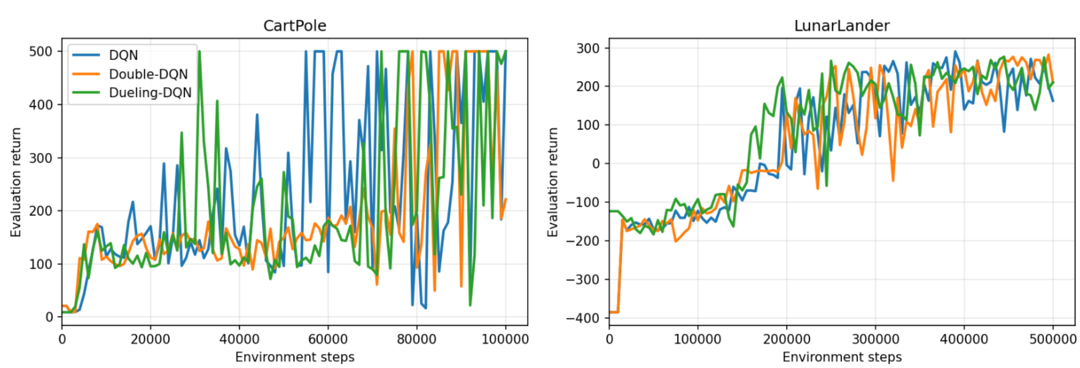

Figure 1. Learning curves on CartPole-v3 and LunarLander-v3 (discrete). Shaded regions indicate variability across random seeds.