Enhancing DQN Performance: Analyzing the Impact of Experience Replay and Target Networks in CartPole-v1

SaiKrishna Reddy Mulakkayala s4238206 [1]

Abstract

This study evaluates the effects of experience replay and target networks in Deep Q-Networks (DQN) for the CartPole-v1 environment by comparing four configurations: naive, only tn, only er, and tn & er. Training is conducted for 10^5 timesteps short of the 10^6 needed for the task, potentially affecting performance. A vectorized environment is used with epsilon-greedy exploration, and hyperparameter tuning is performed via grid search. The results, assessed using the mean and standard deviation of episode rewards over the last 100 episodes, provide insights into the effectiveness of different DQN variations. Additionally, Q-learning curves illustrate training stability and performance trends across configurations.

1. Introduction

Reinforcement Learning (RL) has become a key approach for solving sequential decision-making tasks, with Deep Q-Networks (DQN) proving effective in learning optimal policies from high-dimensional state spaces. However, standard DQN suffers from issues such as unstable learning, slow convergence, and inefficient sample usage due to correlated experiences and overestimation of Q-values. To address these challenges, two major improvements—experience replay (ER) and target networks (TN)—are commonly used to enhance training stability and performance. This study systematically evaluates the impact of these enhancements by comparing four DQN configurations in the CartPole-v1 environment: (1) naive (no ER or TN), (2) only tn(TN without ER), (3) only er(ER without TN), and (4) tn& er (both ER and TN).

A vectorized environment with several concurrent instances is employed to increase sampling efficiency and enable faster policy updates. Exploration and exploitation are balanced using an annealing schedule and an epsilon-greedy exploration approach. To optimize important hyperparameters like learning rate, epsilon decay, and batch size, a grid search is also carried out. Final performance results may be impacted since training is done for 10^5 timesteps rather than the 10^6 stipulated in the assignment due to computational limitations. The mean and standard deviation of episodic rewards from the previous 100 episodes are used to assess each configuration's efficacy. Additionally, Q-learning, the foundation of DQN, is implemented by iteratively updating Q-values using the Bellman equation. The learning process is enhanced with batch updates and target networks, preventing divergence and stabilizing training. Q-learning curves are analyzed to illustrate performance trends and convergence behavior across different configurations. This paper helps to design more stable and sample-efficient RL algorithms by shedding light on the effects of experience replay and target networks on DQN performance.

Understanding how architectural modifications, like the addition of target networks and experience replay, affect DQN algorithm performance would be greatly enhanced by the knowledge gathered from this study. In order to create more reliable, effective, and scalable RL agents that can handle more challenging situations, the results will contribute to the advancement of deep reinforcement learning in the future.

2. Theory

Reinforcement Learning (RL) is a learning paradigm in which an agent interacts with an environment to make decisions that maximize the total reward over time. One of the core algorithms in RL is Q-Learning, a model-free approach where the agent learns a value function, known as the Q-function. This function estimates the future rewards for each state-action pair. Through interactions with the environment, the agent receives feedback in the form of rewards, which is used to iteratively update the Q-function. The goal is for the agent to identify the optimal actions that lead to the highest cumulative rewards, which are typically represented as Q(s, a).

Q-Learning and the Bellman Equation

The fundamental component of Q-learning is the Bellman

[*]Equal contribution [1]LIACS, Universiteit Leiden, Leiden, The Netherlands. Correspondence to: SaiKrishna Reddy Mulakkayala <s4238206@vuw.leidenuniv.nl>.

equation, which gives the Q-value of a state-action pair a recursive definition. With the optimum course of action from the next state taken into account, the equation represents the Q-value as the current reward plus the deferred value of the future Q-values for that state:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a') \tag{1}$$

Where:

- $Q(s,a)$ is the expected return (future reward) from state s and action a,
- $r(s,a)$ is the immediate reward received after taking action a in state s,
- γ is the discount factor, which determines the importance of future rewards,
- $\max_{a'} Q(s',a')$ represents the maximum Q-value for the next state s', indicating the optimal action in the subsequent state.

The objective of Q-learning is to update the Q-values iteratively until an optimal policy is reached. Using the Q-learning update algorithm, which is based on the observed reward and the anticipated future Q-value from the next state, the Q-value for a state-action pair is updated.

2.1. Derivation of the Q-Learning Loss Function

The core idea behind Q-learning is to minimize the difference between the current Q-value estimate and the target Q-value, which is derived from the Bellman equation. This discrepancy is typically represented as a loss function, which is optimized using gradient-based methods. The loss function is created by comparing the current Q(s, a) estimate with the target value obtained through the Bellman equation, with the goal of reducing this gap over time. The target Q-value, denoted as y, is computed as follows:

$$y = r + \gamma \max_{a'} Q(s',a')$$

where:

- r is the immediate reward,
- $\gamma \max_{a'} Q(s',a')$ is the discounted maximum future reward the agent can obtain from state s'.

The update rule for Q-learning can be written as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(y - Q(s,a))$$

where the learning rate α determines the proportion of the difference between the current and target Q-values that are included in the updated Q-value. The loss function, which we define as the mean squared error (MSE) between the target and forecast Q-values, is used to reduce this difference over all state-action pairs:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')}\left[\left(Q(s,a) - \left(r + \gamma \max_{a'} Q(s',a')\right)\right)^2\right] \tag{2}$$

This loss function represents the squared error between the predicted Q-value $Q(s,a)$ and the target value, $r + \gamma \max_{a'} Q(s',a')$. Minimizing this loss function leads to updating the Q-values in a way that brings them closer to the true Q-values, as defined by the Bellman equation.

Use of Deep Q-Networks (DQN)

In conventional Q-learning, each state-action pair's Q-values are kept in a Q-table. However, it becomes impossible to store every state-action pair in a table in complicated situations with huge state spaces. Deep Q-Networks (DQN) use deep neural networks to approximate the Q-function in order to solve this problem. The DQN loss function is determined similarly to Q-learning, but with a neural network representing the Q-function and gradient descent updating the network's parameters (represented by θ). For DQN, the loss function is as follows:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')}\left[\left(Q_\theta(s,a) - \left(r + \gamma \max_{a'} Q_{\theta'}(s',a')\right)\right)^2\right] \tag{3}$$

where:

- $Q_\theta(s,a)$ is the Q-value predicted by the neural network with parameters θ,
- $Q_{\theta'}(s',a')$ is the Q-value predicted by a target network with parameters θ' (the target network is periodically updated with θ).

The network parameters are updated by minimizing this loss function using an optimization algorithm such as Stochastic Gradient Descent (SGD) or Adam.

In Deep Q-Networks (DQN), the tabular Q-learning update rule cannot be used because the Q-values are approximated by a neural network, requiring gradient-based optimization to update the network parameters instead of directly updating a Q-table.

2.2. Why Advanced Update Rules Enable Better Performance in Complex RL Tasks

The update rules from traditional tabular Q-learning cannot be directly applied to Deep Q-Networks (DQN) due to significant differences in how Q-values are updated and stored. In tabular Q-learning, each state-action pair is associated with a specific Q-value stored in a table, and the update rule is relatively simple: the Q-value is updated by using the maximum Q-value from the next state and the observed reward, given that the state space is typically discrete and small.

However, with DQN, the situation is quite different. In DQN, the state space is often continuous and high-dimensional, such as images or other complex data. Instead of using a table to approximate the Q-function, a neural network is employed, which introduces several challenges:

1. **Neural Network Approximation:** In DQN, a neural network that needs backpropagation for parameter updates predicts Q-values. There is no longer a direct index or table storage for the Q-values. Rather, they are learned by varying the neural network's weights according to the difference between the goal and forecast Q-values. Compared to the straightforward table lookup and update employed in tabular Q-learning, this is a far more complicated procedure.

2. **Training Instability:** While DQN uses stochastic gradient descent to update the neural network's parameters, the tabular update rule depends on instantaneous, deterministic updates to Q-values. Training may become unstable as a result, especially if correlated data is used and the Q-values are changed often. This problem does not arise with tabular Q-learning since each update is discrete and does not need gradient-based optimization.

3. **Experience Replay and Target Networks:** While tabular Q-learning does not have this issue, DQN uses replay buffers and target networks to break the correlation between successive samples and stabilize learning. In DQN, experience replay saves previous experiences and randomly samples them, but in tabular Q-learning, the Q-value for that state-action combination can be updated directly from each new sample. This lowers the possibility of overfitting to recent experiences and decouples the Q-value updates from the most recent state-action transitions.

4. **Batch Updates:** DQN employs batch updates as opposed to tabular Q-learning, which updates each state-action pair separately. Instead of updating a single Q-value at a time, the code uses the method `update_batch()` to analyze a batch of transitions sampled from the replay buffer. In order to update the neural network parameters in batch, gradients must be applied and the loss over all transitions must be calculated. The simple tabular update rule would not allow this batch technique, particularly when state spaces are huge.

2.3. Pseudo Code for DQN with ER and TN

Algorithm 1 Deep Q-Network (DQN) with Experience Replay (ER) and Target Network (TN)

```
1: Initialize environment $E$ with $num\_envs$ parallel environments
2: Initialize policy_net (neural network) with random weights
3: Initialize target_net (neural network) with the same weights as policy_net
4: Initialize experience replay buffer $D$ (size: buffer_capacity)
5: Set hyperparameters: learning_rate, discount_factor ($\gamma$), epsilon_start, epsilon_end, epsilon_decay, batch_size, target_update_freq, update_ratio
6: for each episode do
7:   Reset environment to get initial states $s_0$ for each parallel environment
8:   for each timestep $t$ do
9:     Select actions $a_t$ for each environment using epsilon-greedy policy based on states $s_t$
10:    if random.random() ¡ epsilon then
11:      $a_t$ = random_action() {Exploration}
12:    else
13:      $a_t = \arg\max$ (policy_net($s_t$)) {Exploitation}
14:    end if
15:    Take actions $a_t$ in the environment, observe rewards $r_t$, next states $s_{t+1}$, and done flags
16:    Store $(s_t, a_t, r_t, s_{t+1}, \text{done})$ in replay buffer $D$
17:    if $|D| >$ batch_size then
18:      Sample a batch of transitions $(s, a, r, s', \text{done})$ from $D$
19:      Compute target Q-values for each transition:
             $$y_t = r_t + \gamma \max_{a'} \text{target\_net}(s_{t+1}, a')$$
20:      Compute predicted Q-values: $Q_{\text{pred}} = \text{policy\_net}(s_t)[a_t]$
21:      Compute loss: loss = MSE($Q_{\text{pred}}, y_t$)
22:      Perform backpropagation and update policy_net using loss and optimizer
23:    end if
24:    if timestep is divisible by target_update_freq then
25:      Update target_net by copying weights from policy_net
26:    end if
27:    Decay epsilon according to epsilon_decay
28:  end for
29: end for
30: Return the list of rewards after each episode
```

3. Experiments

3.1. Experiment setup

Using the CartPole-v1 environment, the experiment aims to train a Deep Q-Network (DQN) while examining the effects of Experience Replay (ER) and Target Network (TN). Four variants are investigated: a version with only a target network, a version with only experience replay, a final version using both approaches, and a naive DQN without ER or TN. The objective is to assess the ways in which these elements affect learning stability and efficiency. To speed up training, several instances (`num_envs=24`) are run in parallel in a vectorized environment.

3.2. Training setup and Procedure

The agent chooses actions based on an epsilon-greedy strategy, where exploration starts off high but diminishes gradually over time. The policy network acquires knowledge from experiences kept in a replay buffer as needed, while the target network, if utilized, is refreshed at regular intervals by replicating weights from the policy network. The training procedure is organized so that in configurations with ER, updates are executed in batches, while in the naive method, updates happen through single transitions. A discount factor (γ) of 0.99 guarantees that long-term rewards are taken into account, while the learning rate, batch size, and epsilon decay settings are modified through a separate grid search to enhance performance.

The training commences with setting up the environment, the policy network, and, when relevant, the target network and replay buffer. The agent engages with the environment, choosing actions through an epsilon-greedy approach and saving transitions in the replay buffer. When sufficient samples exist, a batch is extracted to refresh the policy network, and when a target network is employed, it undergoes periodic updates to ensure stable learning. During training, epsilon is reduced to shift from exploration to exploitation. Rewards for each episode are recorded, and environments are reset after finishing. The effectiveness of various configurations is assessed using episode returns to identify which arrangement produces the optimal outcomes.

3.3. Hyperparameters Training

The experiment uses a batch size of 256, a replay buffer capacity of 10,000, a learning rate of 5e-4, and a discount factor of 0.99. In order to balance exploration and exploitation, the epsilon decay occurs on an exponential timetable spanning 10,000 steps. To increase stability, the target network if it existsis updated every 1,000 steps. In ER-based settings, the update ratio, which establishes how many updates are made for each environment step, is set to 5. To examine their impact on training effectiveness and performance, these hyperparameters are adjusted in a grid search.

3.4. Grid Search

The training process is optimized by evaluating various combinations of the hyperparameters (learning rate, epsilon decay, and batch size) using a grid search. The average episode rewards are determined by testing each setup over a number of trials. In order to determine the best-performing hyperparameters and provide a comprehensive evaluation of DQN configurations and their effects on learning stability and performance, the results are displayed using bar charts.

4. Results

4.1. Ablation study on Update Ratio

With this experiment, we sought to find the effect of update ratio on the performance and stability of DQNs with target networks and experience replay. Figure shows the result of Experiment.

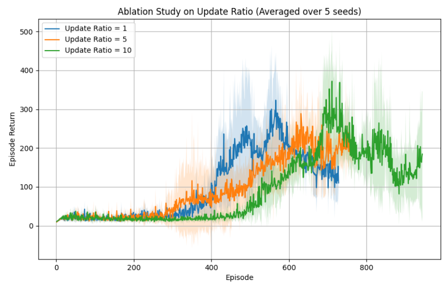

Figure 1. Ablation study with different update ratios

The effect of update ratio—policy network updates per environment step—on DQNs with target networks and experience replay is investigated in this study. Instability results from faster learning but larger volatility at higher update ratios (e.g., 10). Slower, more consistent, and less fluctuating progress is provided by smaller ratios (e.g., 1). A ratio of 5 offers more consistency than a ratio of 10 while providing faster improvement than a ratio of 1. The trade-off between stability and learning speed is highlighted by these results, indicating that moderate update ratios would be ideal for striking a decent balance between dependability and performance.

4.2. Q-learning Curve on CartPole-v1

With this experiment, we sought to find how the naive Q-learning algorithm performs in the CartPole-v1 environment without optimizations like experience replay or target networks. Figure 2 shows the result of Experiment. The naive Q-learning algorithm, tested on the CartPole-v1 en-

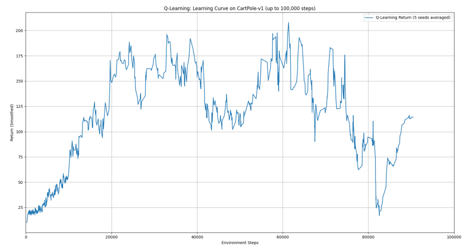

Figure 2. Learning curve showing the performance across the steps

vironment, showed initial exploration with low rewards, gradually improving to peak returns of over 200 around 40,000 to 60,000 steps. However, after this point, performance became unstable with sharp declines and only partial recovery, highlighting the limitations of Q-learning without optimizations like experience replay or target networks. This suggests that while Q-learning can make progress, it struggles with stability and long-term performance without additional techniques to stabilize learning.

4.3. Hyperparameter Grid Search

With this experiment, we sought to find the optimal hyperparameters for a reinforcement learning agent by comparing the smoothed learning curves of various combinations of epsilon values, batch sizes, and learning rates across training episodes.Figure **??** shows the result of Experiment. The graph depicts the results of a grid search aimed

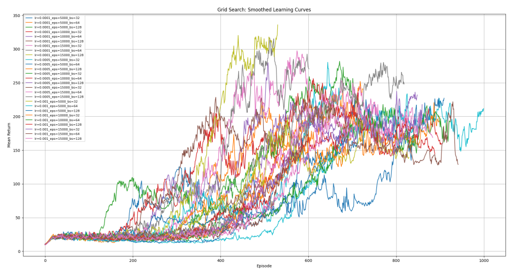

Figure 3. Graph with Hyperparameter Grid search

at optimizing hyperparameters for a reinforcement learning model, specifically focusing on the smoothed learning curves across various training episodes. The experiment explores combinations of learning rates (lr), epsilon decay values (eps), and batch sizes (bs), with each line representing a unique configuration's performance. The y-axis, labeled Mean Return,indicates the agent's performance, while the x-axis, Episode,represents the progression of training. The graph illustrates how different hyperparameter settings affect the learning trajectory, with some configurations leading to rapid initial improvement and others showing steadier, slower progress. The wide variance in performance highlights the sensitivity of reinforcement learning models to hyperparameter tuning and the importance of systematic exploration to identify optimal settings.

4.4. Baseline Configuration

With this experiment, we sought to find the impact of different DQN configurations—naive, target network only, experience replay only, and both target network and experience replay—on learning performance by comparing their mean episode returns over 5 seeds.Figure **??** shows the result of Experiment. The results highlight that while methods like

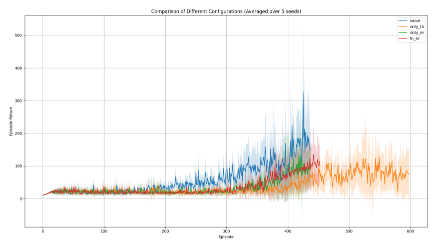

Figure 4. Performance of Baseline configuration with randomized seeds

the tn& er configuration, which combines target networks and experience replay, lead to steadier and slower progress, this approach ultimately results in more stable and reliable learning. The key takeaway is that prioritizing stability over rapid but inconsistent progress yields better long-term performance in DQN. Thus, a balance between stability and speed is crucial for achieving consistent success in reinforcement learning tasks.

5. Discussion

When DQN was implemented in the CartPole-v1 environment, it became clear how crucial stability is to reinforcement learning. The tn& er setup, which combined target networks and experience replay, produced the greatest steady performance, but at the expense of slower learning in the beginning than more straightforward configurations like naive and merely er. In DQN, the trade-off between stability and convergence speed is crucial, as achieving long-term success frequently necessitates compromising initial learning effectiveness. The trials also demonstrated the importance of hyperparameter adjustment, since configurations such as batch size, epsilon decay, and learning rate significantly affected the final result. In complicated ecosystems, for example, a slower epsilon decay maintained higher exploration and ultimately yielded superior outcomes. It was possible to have a better understanding of how various hyperparameters affected performance by conducting the grid search experiments. Grid search demonstrated the possibility of overfitting or sluggish convergence, especially when specific hyperparameters were poorly selected, even if it offered a methodical way to experiment with different settings. This demonstrates how crucial it is to select the ideal setup for every work in order to maximize learning effectiveness and performance.

5.1. Weakness

One significant disadvantage of the tn& er arrangement during the studies was its slower initial learning, which appeared to put stability before speed. Although this strategy produced more reliable long-term results, it came at the price of slower initial development. Furthermore, the addition of target networks and experience replay resulted in higher computational overhead, which impacted processing time and memory use and eventually slowed down the training process. Another drawback was the experience replay and target network's fixed update ratio, which may not have been the best strategy. An adjustment method that was more flexible or dynamic might have increased overall efficiency.

6. Conclusion

In conclusion, this study concludes by highlighting the importance of Target Networks (TN) and Experience Replay (ER) in enhancing the stability and functionality of Deep Q-Networks (DQN) in the CartPole-v1 environment. The tn&er configuration, which combines ER and TN, produced the most stable long-term performance, even though it showed slower initial learning. A more constant and dependable convergence was ensured by this configuration, which successfully addressed issues like associated experiences and Q-value overestimation. The study found certain trade-offs even though the results showed that hyperparameter tuning through grid search improved performance. In particular, despite offering stability, the tn&er combination resulted in slower learning in the beginning, and processing time was extended by the computational overhead of ER and TN. Additionally, it's possible that the fixed update ratio for ER and TN wasn't the best, indicating the possible advantages of looking into more flexible approaches. Overall, the results show how crucial it is to give stability precedence over quick learning while performing reinforcement learning tasks. In order to improve overall performance and training efficiency, future studies could look into adaptive techniques for ER and TN.

7. Future Work

In order to improve efficiency and convergence, my future research will look into adaptive techniques for Experience Replay (ER) and Target Networks (TN), changing their update rates based on the agent's performance. By experimenting with a more slow decrease of epsilon, I also hope to improve the exploration strategy by enabling more thorough exploration prior to exploitation. In order to address problems like Q-value overestimation and enhance learning stability, I will also investigate more complex DQN variations, such as Double DQN and Dueling DQN. Optimizing DQN for more effective reinforcement learning in complicated situations is the aim of these initiatives.