Exploring Reinforcement Learning: DQN, DDQN, and N-step DQN Algorithms

Panagiotis Konstantinos Kakkavas (03120866)

December 2023

Abstract

This report explores Reinforcement Learning (RL), focusing on Deep Q-Networks (DQN), Double DQN (DDQN), and N-step DQN algorithms. It aims to provide a comprehensive understanding of their mechanisms, implementations, and applications in the field of machine learning.

1 Introduction to Reinforcement Learning

Reinforcement Learning is a subset of machine learning where an agent learns to make decisions through trial and error interactions with an environment, aiming to maximize cumulative rewards.
The algorithms implemented in this project are off-policy meaning that the decision-making process during training (e-greedy) and during evaluation maximum Q value differ.

1.1 Key Concepts

- **Agent**: The learner or decision-maker.
- **Environment**: The system with which the agent interacts.
- **Action**: Decisions made by the agent.
- **State**: The current situation or context in the environment.
- **Reward**: Feedback from the environment in response to the agent's actions.

2 The Bellman Equation

The Bellman Equation is central to many RL algorithms, providing a recursive relationship for calculating the value of a state or a state-action pair.

2.1 State-Value Function

The value of a state under a policy π:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \tag{1}$$

2.2 Action-Value Function (Q-function)

The value of taking an action a in state s under policy π:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \tag{2}$$

2.3 Bellman Expectation Equation

For the action-value function:

$$Q^{\pi}(s,a) = \mathbb{E}\left[R_{t+1} + \gamma Q^{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\right] \tag{3}$$

3 Deep Q-Network (DQN)

DQN[1] integrates deep learning with Q-learning, using a neural network to approximate the Q-function.

3.1 Network Architecture

- **Input**: State representation.
- **Hidden Layers**: Feature extraction layers (often convolutional for image inputs).
- **Output**: Q-values for each action.

3.2 Learning Process

To train efficiently, we use the **Experience Replay Buffer** storing transitions to break the correlation between learning steps and to create batches. Replay Buffer types are either a simple array from which you simply sample or a more complex data structure that uses priority to sample experiences that will have a high impact on the learning process due to high Temporal Difference error.

[1] Mnih, V. et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533. `https://arxiv.org/abs/1312.5602`

[2] Then, in the new data structure using segment trees, we sample from the experiences sorted by rank. The probability of sampling transition i is given by

$$P(i) = \frac{p_i^a}{\sum_k p_k^a} \tag{4}$$

and $p_i > 0$ is the priority of transition i. The exponent α determines how much prioritization is used, with $\alpha = 0$ corresponding to the uniform case.
p_i has two proposed variants in the original paper: $p_i = |\delta_i| + \epsilon$, where ϵ is a small positive constant, and $p_i = \frac{1}{\text{rank}(i)}$, where rank$(i)$ is the rank of transition i when the replay memory is sorted according to $|\delta|$.
Priority Experience Replay buffer is implemented in the code provided yet not used in the notebook examples.

3.3 Loss Function

The loss function for DQN:

$$L(\theta) = \mathbb{E}\left[\left(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \theta) - Q(S_t, A_t; \theta)\right)^2\right] \tag{5}$$

where θ are the parameters of the main network

4 Double Deep Q-Network (DDQN)

DDQN[3] addresses the overestimation bias of DQN, caused from the use of maximum action value as an approximation for the maximum expected action value, by decoupling action selection and evaluation.

The benefit of Double Q-Learning becomes more pronounced as the number of possible actions increases. This is because with more actions, the likelihood of overestimating the Q-values also increases due to the max operation in the Q-value update step. By decoupling the action selection from its evaluation, Double Q-Learning reduces this overestimation, potentially leading to more stable and reliable learning, especially in environments with a large action space. This is also one of the reasons that in the CartPole experiment dqn and ddqn results are very similar.

4.1 Modification in DDQN

In DDQN, the action selection is done by the main network, while the evaluation is done by the target network and the Bellman equation becomes:

$$Q(s, a; \theta) = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_{a'} Q(S_{t+1}, a'; \theta); \theta^-) \tag{6}$$

[2] Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized Experience Replay. *arXiv preprint arXiv:1511.05952*. `https://arxiv.org/abs/1511.05952`

[3] Van Hasselt, H. et al. (2016). Deep Reinforcement Learning with Double Q-learning. *AAAI Conference on Artificial Intelligence*. `https://arxiv.org/abs/1509.06461`

The weights of the main network are updated normally, but the weights of the target net are updated less frequent with a copy of the weights of the main network's weights.

5 N-step DQN

N-step DQN[4] considers a sequence of n steps for updates, extending temporal-difference learning and leading to faster learning.

5.1 N-step Return

The N-step return:

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k R_{t+k+1} \tag{7}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_t^{(n)} + \gamma^n \max_{a'} Q(S_{t+n}, a') - Q(S_t, A_t) \right) \tag{8}$$

6 Reward Shaping

In this study, we utilized the default reward function provided by OpenAI Gymnasium in the CartPole environment, focusing on the duration for which the pole remains in an upright position. Given the established efficiency of classic control solutions in such contexts, it's inferred that an optimal reward structure exists that can guide the model towards a satisfactory policy. Specifically, a reward analogous to that of a PD controller, emphasizing penalties based on the pole's position and velocity, would align the model's learning process with that of a PD-controlled system.

To expedite convergence, starting with pre-established Q-values reflective of known control strategies, such as a positional controller, can be beneficial. While this approach reduces the initial exploration randomness, it potentially enhances the model's ability to balance exploitation with exploration, based on expert-defined initial conditions.

7 Conclusion

This report presents an analysis and implementation in python with PyTorch and open-ai-gymnasium of DQN, DDQN, and N-step DQN algorithms in reinforcement learning, highlighting their advancements and applications in simple CartPole environment.

The experimental results where:

[4] S. R. S. and B.A.G. Reinforcement Learning: An Introduction *(*2nd ed.). MIT Press. 2018 `http://incompleteideas.net/book/the-book-2nd.html`

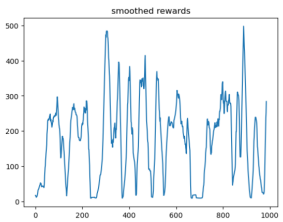

Figure 1: DQN

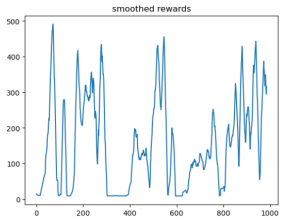

Figure 2: DDQN

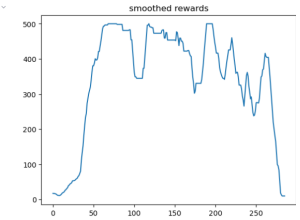

Figure 3: N-step DQN

The experiment yielded insightful findings regarding the model's rule-set in relation to the observation space. Notably, when evaluating combinations such as velocities with angle or position, the model predominantly prioritizes position and angle over velocities. This suggests a discernible correlation between position and angle, indicating a selective emphasis on these parameters within the decision-making process.

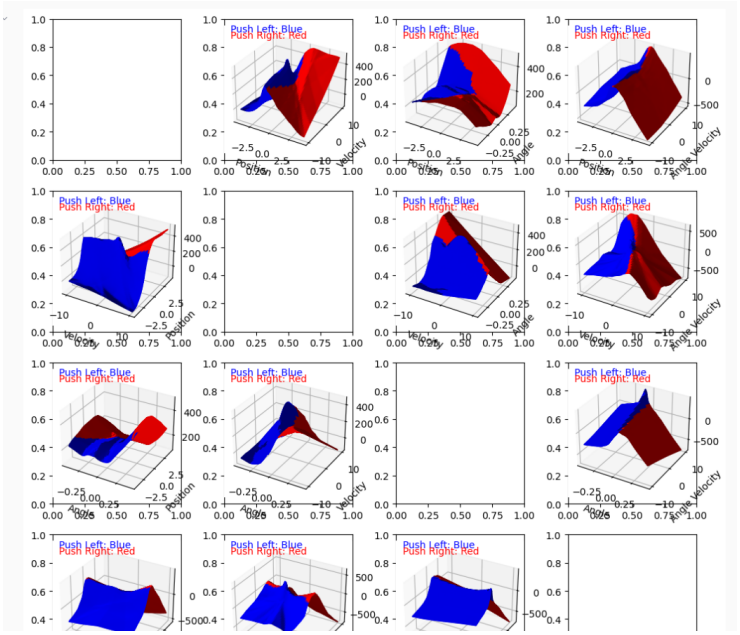

Figure 4: Q-values for pairs of observation space variable

The N-step Q learning algorithms severely overpassed the performance of the single step dqn and ddqn algorithms. Yet way more efficient algorithms can be implemented and combined with n-step to provide better results, like Prioritized Experience Replay, Dueling network architecture, noizy nets, distributional Q-

learning and many more.

You can find the code analytically implemented and commented in my GitHub repository: `https://github.com/ElGreKost/RL-SemesterProject.git`