SUS Second Global Assignment Report

Fabian Ozga

June 4, 2025

1 Model Architecture

For this project, we aimed to develop an agent capable of mastering the CartPole environment. We selected the Deep Q-Network (DQN) algorithm as our foundational model. Our choice was motivated by DQN's proven effectiveness in solving problems with discrete action spaces and its status as a landmark work in deep reinforcement learning, providing a strong baseline upon which to build and evaluate subsequent enhancements. We anticipated that while a basic DQN could achieve reasonable performance, incorporating more advanced techniques would be necessary to attain optimal and stable results.

1.1 Classic Deep Q-Network (DQN)

We implemented the Deep Q-Network (DQN) algorithm, as introduced by Mnih et al. (2014) [1], which utilizes a deep neural network to approximate the Q-function. This function estimates the expected return for performing a specific action in a given state. Our objective was to train the DQN to learn a policy that maximizes this expected return by accurately estimating Q-values and selecting actions with the highest corresponding estimates.

Our DQN implementation incorporated two key components: a Q-network and an experience replay buffer. The Q-network, a deep neural network, processes the environment's state as input and outputs the estimated Q-values for all possible actions. The experience replay buffer stores transitions encountered by the agent during its interaction with the environment. During training, we sampled mini-batches of experiences uniformly from this buffer to train the Q-network. This experience replay mechanism was crucial for decorrelating experiences and stabilizing the learning process, as mentioned in the original paper.

The Q-network was updated using the following loss function:

$$\mathcal{L}_\theta = \frac{1}{B}\sum_{i=1}^{B}\big(\text{TD }(s_i, a_i, s_i^{'})\big)^2$$

where the temporal difference (TD) error is defined as:

$$\text{TD }(s_i, a_i, s_i^{'}) = Q_\theta\ (s_i, a_i) - \big(r_{(s_i, a_i, s_i^{'})} + \gamma \max_{a_i^{'} \sim \bar{Q}_\theta} \bar{Q}_\theta\ (s_i^{'}, a_i^{'})\big)$$

Here, Q_θ represents the learned Q-network, and $\bar{Q}_\theta$ is the target Q-network. The target network, a periodically updated copy of the Q-network, was used to compute target Q-values, a technique known to enhance learning stability and performance.

For exploration, we implemented an ϵ-greedy strategy. In this strategy, the agent selects a random action with probability ϵ and the action with the highest Q-value estimate with probability $1 - \epsilon$. We designed the value of ϵ to anneal over time, encouraging broader exploration in the initial phases of training and gradually shifting towards exploitation as the agent's knowledge of the environment improved.

1.2 DQN Network Architecture

For the Q-value predicting network (DQN net), we adopted a classic architecture:

- An input layer corresponding to the state dimensions of the CartPole environment.
- One hidden layer with 256 neurons.

- An output layer providing Q-value estimates for each action.

We utilized the ReLU activation function for the hidden layer. The final output layer did not have an activation function, as its outputs were treated as the Q-values for each action.

2 Enhancements to DQN

2.1 Double DQN (DDQN)

Recognizing the tendency of the standard DQN to overestimate Q-values, which can negatively impact training, we implemented the Double DQN (DDQN) technique proposed by van Hasselt et al. (2015) [2]. In vanilla DQN, the same network is used to select the best action and then to estimate its Q-value, leading to potential overoptimism.

DDQN addresses this by decoupling the action selection from the value estimation in the target Q-value calculation. We modified the TD error calculation as follows:

$$\text{TD } (s_i, a_i, s_i^{'}) = Q_\theta \ (s_i, a_i) - \left(r_{(s_i, a_i, s_i^{'})} + \gamma \ \bar{Q}_\theta \ (s_i^{'}, \underset{a_i^{'} \sim Q_\theta}{\text{argmax}} \ Q_\theta(s_i^{'}, a_i^{'})\right)$$

In this formulation, the online network Q_θ is used to select the best action for the next state $s_i^{'}$, while the target network $\bar{Q}_\theta$ is used to evaluate the Q-value of that selected action. This modification, requiring minimal code changes, has been shown to lead to less inflated Q-value estimates, more stable learning, and ultimately, better policies. Although Q_θ and $\bar{Q}_\theta$ are not entirely decoupled, this approach provided a noticeable performance increase without introducing additional network parameters.

2.2 TD_n - N-step Q-value Estimation

To further enhance our model, we incorporated N-step TD Q-value estimation (TD_n), drawing from the work of Sutton (1988) [3]. Standard TD learning updates the Q-network based on a single-step reward plus the estimated Q-value of the subsequent state. In contrast, TD_n accumulates rewards over n steps and sums this accumulated reward with the estimated Q-value of the state encountered n steps later.

Our Double DQN TD_n loss was defined as:

$$\text{TD}_n(s_i, a_i, s_{i+n}^{'}) = Q_\theta \ (s_i, a_i) - \left(\sum_{k=0}^{n-1} \gamma^k \ r_{(s_{i+k}, a_{i+k}, s_{i+k}^{'})} + \gamma^n \underset{a_{i+n}^{'} \sim \bar{Q}_\theta}{\max} \ \bar{Q}_\theta \ (s_{i+n}^{'}, a_{i+n}^{'})\right)$$

Implementing TD_n necessitated modifications to our ExperienceBuffer class. We utilized the `deque` module from Python's `collections` library to temporarily store the n most recent transitions. This deque acted as an intermediary, processing transitions before they were stored in the main buffer. The main storage was adapted to hold n-step accumulated rewards and the state $s_{i+n}^{'}$ instead of single-step rewards and $s_i^{'}$.

3 Observations and Performance

Our initial implementation of classic DQN, using reasonable but unoptimized hyperparameters, readily achieved scores oscillating around 200 in the test CartPole environment. To leverage existing knowledge and given our limited computational budget, we adopted semi-optimized hyperparameters from the RL Baselines3 Zoo. With these parameters, the network occasionally reached the maximum possible score of 500. The trajectory of training Classic DQN is presented in Figure 1.

However, we observed significant variance across training runs, with some instances failing to achieve this peak performance. This highlighted the need for methods to reduce performance variance.

The introduction of DDQN significantly improved model stability and performance. Figure 2 shows that the DDQN model tends to learn faster and achieve higher scores more consistently than the classic DQN.

Further incorporating N-step TD alongside DDQN led to the best results, enabling the agent to consistently reach the maximum score in most training runs, as depicted in Figure 3. These figures suggest that adding DDQN and then N-step TD makes the model learn faster. While individual training runs (as shown in the figures) provide some indication of improved stability (e.g., smoother learning curves

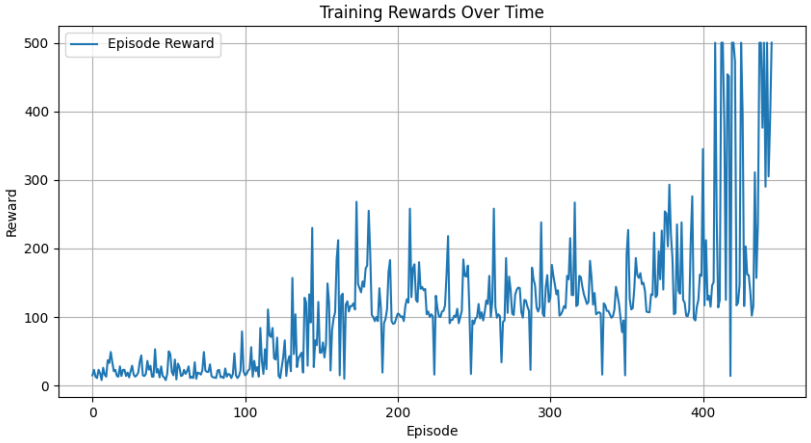

Figure 1: Training performance of the classic DQN model.

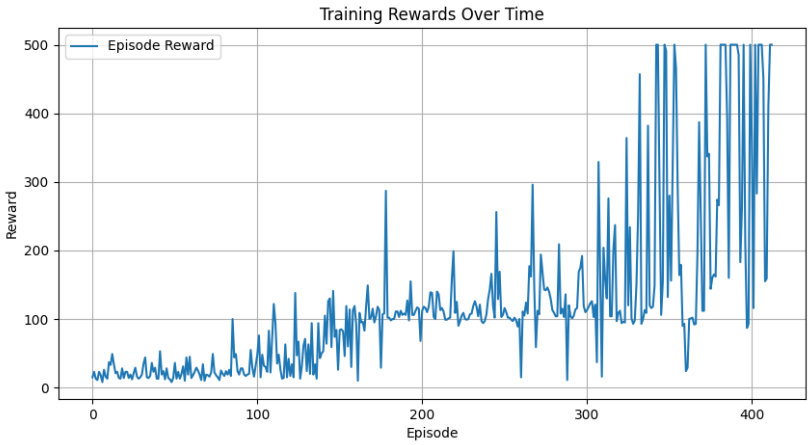

Figure 2: Training performance with the Double DQN (DDQN) enhancement.

or less variance in later stages), a full assessment of stability would require comparing results over many independent runs. Nevertheless, the literature, such as the RAINBOW paper (Hessel 2017) [4], widely acknowledges that these techniques, particularly N-step TD, generally yield significant improvements in both performance and learning stability.

During the development and refinement of our model, we made several key observations:

- The CartPole environment, while relatively simple, served as an effective testbed for demonstrating the capabilities of classic DQN methods.
- We directly encountered the well-known issue of instability in RL experiments. It is widely acknowledged in RL literature that experiments often require multiple runs across different random seeds, with results reported as averages accompanied by confidence intervals. This instability in RL experiments can stem from several factors:
  - Training data in RL is highly correlated, because future observations depend on current actions.
  - Small updates to the Q-network can lead to significant changes in the agent's policy, altering data distribution.
  - Correlation between action-values and target values, as the rargets are derived from current Q-value estimates.
  - Value function overestimation, especially in standard Q-learning.

Figure 3: Training performance with both Double DQN and N-step TD enhancements.

- High variance in gradient estimates, due to environmental stochasticity, delayed rewards, and exploration.
- Performance can be highly sensitive to hyperparameters; e.g., a high learning rate often causes instability.
- Exploration noise is necessary but can contribute to instability if not managed.

4 Conclusion and Learned Lessons

Our work on the CartPole environment demonstrated that while a classic DQN can learn the task, enhancements like Double DQN and N-step TD are crucial for achieving robust, high-performance, and stable learning. We successfully implemented these techniques, observing a marked improvement in the agent's ability to consistently solve the environment.

The primary obstacle encountered was the inherent instability of RL training. Our experience underscored the importance of techniques designed to mitigate this, such as experience replay, target networks, and the specific enhancements (DDQN, N-step TD) we implemented. We learned that careful hyperparameter tuning and architectural choices, even for seemingly simple environments, are critical. Furthermore, the process reinforced the understanding that results in RL often require aggregation over multiple trials to draw reliable conclusions due to this variance.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013. (Original NIPS 2013 deep learning workshop paper; later Nature paper in 2015: `https://arxiv.org/abs/1312.5602`)

[2] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016. (`https://arxiv.org/pdf/1509.06461.pdf`)

[3] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988. (`http://incompleteideas.net/papers/sutton-88-with-erratum.pdf`)

[4] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018. (`https://arxiv.org/pdf/1710.02298.pdf`)