Q-Learning and Actor-Critic Methods in the Cart Pole Environment

Nikita Baklazhenko, Alessandro Ciancetta and Alessandro Tenderini

June 27, 2023

1 Introduction

The aim of this project is to compare the performances of four different reinforcement learning algorithms in the cart pole environment by OpenAI Gym API. We consider both action-value and policy-gradient methods and implement them in Python. The code to reproduce the results is attached to this report.

2 Gymnasium CartPole Environment

The Cart Pole environment is part of the classic control environments in OpenAI's Gym library. The goal in this well-known vanilla environment is to balance a pole by applying forces to a cart moving along a track, with the pole attached to the cart. In the Cart Pole environment, the action space is discrete and made up of two values, 0 and 1, which correspond to the direction of the fixed force that is applied to the cart. Specifically, an action of 0 corresponds to pushing the cart to the left, while 1 corresponds to pushing the cart to the right. The velocity that is reduced or increased by the applied force is not fixed and depends on the angle at which the pole is pointing. The observation space is a four-dimensional vector that represents various aspects of the current state of the system:

- **Cart Position**: it can take values between (-4.8, 4.8). However, the episode terminates if the cart leaves the (-2.4, 2.4) range.
- **Cart Velocity**: it can take any value from negative infinity to positive infinity.
- **Pole Angle**: it can be observed between (-24°, +24°) degrees, but the episode terminates if the pole angle is not in the range (-12°, 12°).
- **Pole Angular Velocity**: it can take any value from negative infinity to positive infinity

The reward structure in this environment is quite straightforward. The aim is to keep the pole upright for as long as possible, so the environment provides a reward of +1 for every step taken. The initial state of the environment is defined by randomly assigning all state variables a value between -0.05 and 0.05. Then, as previously mentioned, an episode ends if any of the following conditions are met:

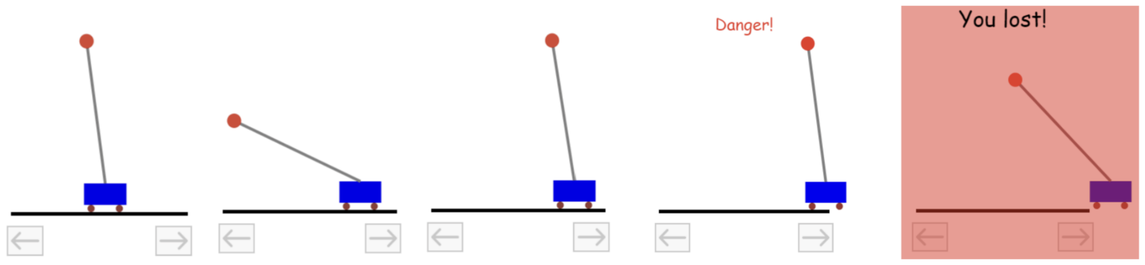


Figure 1: *The Cart Pole environment*

- Cart position is greater than ± 2.4
- Pole angle is greater than $\pm 12°$
- Episode length is greater than 500

The objective for the agent is to learn how to keep the pole upright by pushing the cart in the appropriate direction. Despite its apparent simplicity, this task proves to be non-trivial and it requires sophisticated learning algorithms to be solved effectively, thus making it a useful benchmark for comparing different reinforcement learning models.

3 Models and Implementation

We now briefly describe the algorithms that we use to solve the cart-pole problem. We consider both action-value and policy-gradient methods and we consider on- and off- policy algorithms. We denote by $\mathcal{A}, \mathcal{S}$ the action and state spaces respectively.

3.1. Action-value methods: Q-learning

We consider two variants of the Q-learning algorithms.

Discrete Q-learning To begin with, we implement a simple discretization of the environment and implement a classical Q-learning algorithm. As it is well-know, Q-learning consists in estimating a lookup table of returns associated to every state-action pair in the environment, in order to collect the best action for any given state in the estimated optimal policy. Even if in the cart-pole environment the action space still consists of only two possible actions, the state space consists in a four-dimensional continuous space that makes impossible to have a state-action table. Therefore, in order to apply plain-vanilla Q-learning we first discretize the state space by considering 30 bins for each of the dimensions involved (namely position, velocity, angle and angular velocity). The result is a lookup table of Q-values of dimension $30^4 \times 2$. The behaviour policy is initialized at random, while the Q values associated to the target policy are updated using the standard off-policy TD-control step based on the Bellman equations:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

The parameters α and γ are the learning rate and discount factor, respectively. α controls the extent to which the Q-values are updated at each iteration, while γ determines the importance of future rewards. To ensure adequate exploration of the state-action space, we implement an ε-greedy strategy. The agent mostly chooses the action with the highest Q-value (exploitation) but with probability ε selects uniformly at random one of the other actions in $\mathcal{A}$ (exploration). The balance between exploration and exploitation is controlled during training to favor the latter more and more, by decreasing ε over the episodes.

Deep Q-learning A main limitation of the method above lays in the arbitrary discretization of the state-space. Potentially, a finer grid of values could provide a sharper representation of the information available in the environment and could help the algorithm to converge to the actual optimal policy. Even if in this toy example it could be possible to refine the grid, in real-world applications with high-dimensional state-space is often impossible to obtain sufficiently fine grids without incurring in substantial loss of information. A natural solution to this problem is to turn to approximate methods. In particular, we choose to parametrize the Q function using a feedforward neural network with three layers, thus adopting a deep Q-learning approach. Instead of maintaining a full table of Q-values, a neural network is trained to estimate these values. This is done by minimizing the difference between the predicted Q-value of a state-action pair and the Q-value derived from the experienced reward and the predicted Q-value of the next state. The minimization is performed by applying an optimization algorithm, such as Adam, to adjust the neural network's weights. Moreover, we implement DeepMind's version of the algorithm, DQN, which introduces several improvements to stabilize the learning process. One is the use of a separate target network to generate the Q-values of the next state in the update rule, which prevents moving target problems. The target network is identical to the original network but with its weights updated less frequently, preventing rapid changes in the value estimates. Another major enhancement is the use of a replay buffer, which stores a history of state, action, reward, and next state transitions. During training, minibatches of these transitions are sampled from the buffer and used to train the network. This process breaks correlations in the observation sequence, thus reducing variance and making better use of past experiences. Furthermore, an ε-greedy strategy is used also in this case to balance exploration and exploitation during the agent's learning process to reduce the fraction of exploration actions.

3.2. Policy-gradient methods: actor-critic algorithm

We now explore methods based on parametrized policies. In general, this methods find ways to learn a parameter $\theta \in \mathbb{R}^d$ that links each state to an action. Updates of the policy occur via updates in θ, according to some policy-performance measure $J(\theta)$:

$$\theta_{t+1} \leftarrow \theta_t + \alpha \widehat{\nabla J(\theta_t)}$$

where the gradient can be computed using the representation of the policy gradient theorem.

Baseline actor-critic algorithm Contrarily to other policy-gradient algorithms like REINFORCE, the actor-critic algorithm offers a fully on-line way to policy optimization, by implementing both an "actor" (i.e. a policy parametrized by θ) and a "critic" (i.e. a state-value function parametrized by w, $\hat{v}(S;w)$) functions that allow to take and evaluate the undertaken action with an updated value function before the end of the episode. The policy is then updated as

$$\theta_{t+1} \leftarrow \theta_t + \alpha \delta_t \nabla \ln \pi(A|S;\theta_t),$$

where $\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1};w_t) - \hat{v}(S_t;w_t)$ is the estimated TD error and the parameter value function is updated as $w_{t+1} \leftarrow w_t + \alpha^{(w)} \delta_t \nabla \hat{v}(S_t;w_t)$. The two learning components (actor and critic) provide a natural balance for exploration and exploitation, as the critic guides the actor's learning process, avoiding the high variance problem often associated with policy gradient methods. We implement the actor and critic functions using 2-layers feedforward neural networks with ReLU activation functions and using the Adam optimizer. The output of the actor passes through a softmax output layer to obtain a probability distribution over $\mathcal{A}$.

Proximal policy optimization A problem of the baseline algorithm is that nothing ensures a smooth path in the policy update policies, meaning that the policy may change a lot in two subsequent iterations due to the high non-linearity in the parameters of the approximators. The literature focused on this problem, starting from algorithms like the natural policy gradient. An interesting strand consider algorithms that add a penalty factor in the update rule based on the KL divergence between new and previous iteration policies. One of the state-of-the-art algorithms coming after this literature is the Proximal Policy Optimization, an actor-critic method that restricts the update step by clipping the importance sampling ratio $\rho_t(\theta_t) = \frac{\pi_{\theta_t}(a_t|s_t)}{\pi_{\theta_{t-1}}(a_t|s_t)}$ between the new and previous policy. The importance-sampling ratio enters the objective function to weight the "advantage" function $J(\pi_{\theta+\Delta\theta}) - J(\pi_\theta)$, which is here estimated using Generalized Advantage Estimation. Samples outside the clipped (trusted) region are discarded, thus excluding overly large updates. As before, we can simply use the Adam optimizer to perform the updates for the same network parametrization used for the baseline algorithm.

4 Results and Analysis

The Cart Pole problem serves as an excellent benchmark for testing these models as it requires the models to maintain balance, making it a complex yet relatable task for understanding their functionalities. We also include a random model that takes random action to have a random benchmark. It is important to note that each model takes a different number of episodes to train and obtain improved performance. One immediate observation from the analysis was that all four models significantly outperformed a non-learning, random simulation. The rewards achieved by each of the models were at least ten times greater than those obtained by randomly selecting actions, shown in Figure 2, clearly demonstrating the power of the considered model-free learning algorithms to discern patterns and make optimal decisions. In comparing the learning rates of these models, the Discrete Q-Learning algorithm demonstrated a higher speed of learning and greater robustness. This made it a strong contender for the most reliable model, particularly when fast and stable results are the main criterion. However, not everything went smoothly for the models. A common trait observed across all models was the occurrence of a performance plateau, indicating a performance ceiling that they struggled to exceed. This could be a result of inherent limitations of the models or the constraints of the Cart Pole problem itself. In cases where the models managed to break free from this plateau, they started to overfit, leading to a noticeable drop in performance despite attempts to mitigate this through regularizations and reducing mutation rates. Despite these challenges, there were instances where after a long plateau period, the models would suddenly deliver a significant, consistent improvement. The models that handled continuous variables showed a distinct behavior compared to the discrete Q-Learning model. Although they took longer to learn and were less consistent in their performance, they provided the opportunity for superior results in the long term, as long as one had some patience and a bit of luck. To better understand the performance of each model over time, we plotted the learning curves for each model in Figure 3. In each plot of the left column, the x-axis represents the number of episodes, and the y-axis represents the reward of each episode. A line representing the moving average of the rewards across 100 episodes was also added to provide a clearer view of the overall performance trend of each model. We also plot only the moving average to allow a better visualization in the right column of the same Figure. Diving deeper into the performances of each model, the DQN emerged as a strong performer. It was able to achieve a moving average of rewards close to 2000 after only 900 episodes, outperforming his peers. The Discrete Q-Learning model showed resilience and consistency by reaching a moving average of 200 after 2500 episodes and maintaining that level for the long run. In contrast, the Actor-Critic Model demonstrated a high degree of volatility, with its moving average fluctuating between 100 and 800. The PPO model also showed volatility, though to a lesser extent, with the moving average ranging between 150 and 350. The moving average for the random strategy, as a comparison, was observed to be in the range of 10 to 25, once again emphasizing the effectiveness of the reinforcement learning models over random strategies. In conclusion, these findings provide a clear demonstration of the capabilities and constraints of these reinforcement learning models. Each model presents its own strengths and weaknesses - the Discrete Q-Learning model for its speed and robustness, the DQN for its high reward achievements, and the PPO and Actor-Critic Models for their potential to achieve significant improvements, although with increased volatility. These characteristics make the choice of model heavily reliant on the specific requirements of the problem at hand, the desired speed of learning, and the acceptable level of performance volatility.

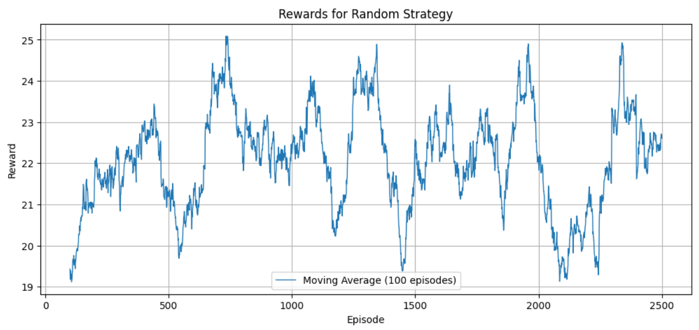


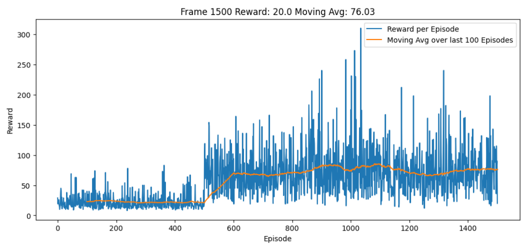
Figure 2: *Moving Average - Learning curve of Random streategy*

4.1. State-Action analysis

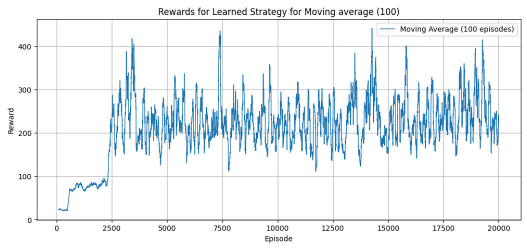
Extending our analysis, we turned our attention to the state-action distributions. This was achieved through the generation of scatter plots of the Discrete Q-Learning model during different episodes, shown in Figures 4 and 5. The scatter plots enabled us to observe the changes in the range of positions the cart-pole system could reach without failure (where failure is defined as the pole falling or the cart moving out of the acceptable range). Initially, in the episodes ranging from 0 to 500, the position ranges were smaller, indicating a more cautious strategy employed by the learning agent. However, as the agent continued to learn and adjust its policy, a noticeable increase in the position ranges was observed in the episodes from 500 to 1000. This was an indication of the agent's improved ability to control the cart-pole system and prevent failure even in more extreme positions, evidencing the effectiveness of the Discrete Q-Learning model. To further our understanding of the state-action dynamics, we generated histograms for the state-action pairs in the Discrete Q-Learning model. These were generated for the 0-500 episodes, in Figue 6, and the 500-1000 episodes range in Figure 7. The histograms revealed a pattern where, with training, the model encountered more state values closer to zero for the angle and angular velocity. This suggests that as the learning progresses, the system manages to maintain a more upright pole (lower angles) and with less rotational movement (lower angular velocities), reflecting a more stable control strategy. The histograms also highlighted interesting distinctions between the actions the agent chose based on the current state. The choice of action varied more significantly when the angular velocity and cart velocity values were larger or smaller than zero. This differentiation in action selection makes sense given that the direction of the cart's movement and the pole's rotation significantly affect the necessary control action. However, for the position value, the action distribution was relatively uniform, likely because the position of the cart alone doesn't significantly influence the required action. The pole angle also showed a distinct action distribution, indicating its importance in the decision-making process. When comparing these results to those obtained from the Deep Q-Network model in Figure 8, similarities could be seen, particularly to the histograms generated for the Discrete Q-Learning model in the 500-1000 episodes range. However, a peculiar result was observed. For the Deep Q-Network model, while there were clear differences in action selection based on the cart and pole velocities, the action distribution for different pole angles was surprisingly uniform. This uniform distribution implies that the Deep Q-Network model may not be considering the pole angle as heavily in its decision-making process.

5 Conclusion

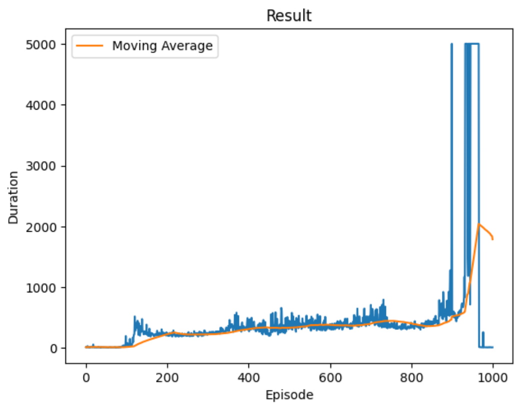
The results of the simulations show clearly different behaviours of the three algorithm in the cart pole environment. The discrete Q-learning algorithm, despite showing consistent results, is unable to achieve substantial gains in performances even after many episodes. PPO also shows quite stable results which are consistently higher than discrete Q-learning, so that it may be preferrable if stability is the focus. On the other hand, if performance is the main concern and computational constraints for runnning many episodes are not binding, deep Q-learning appears to be the most promising solution.



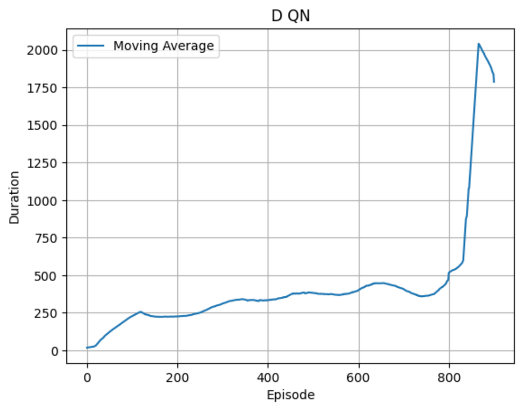
(a) *Learning curve of Discrete Q-Learning*



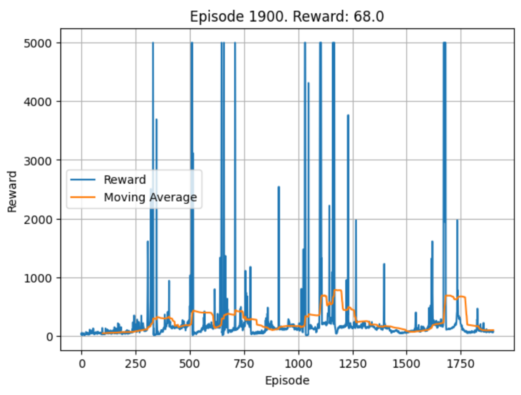
(b) *Moving Average - Discrete Q-Learning*



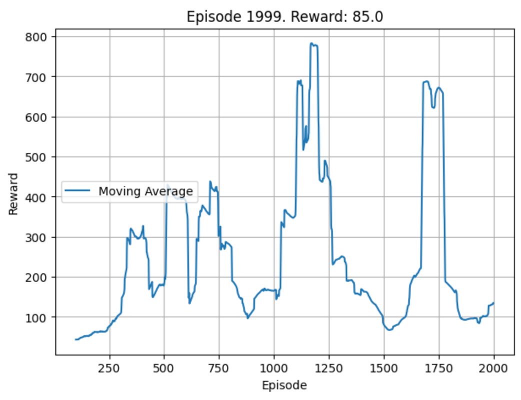
(c) *Learning curve of Deep Q model*



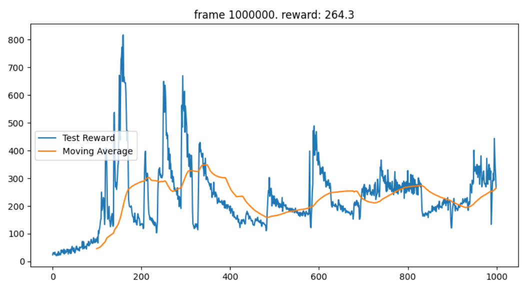
(d) *Moving Average - Deep Q model*



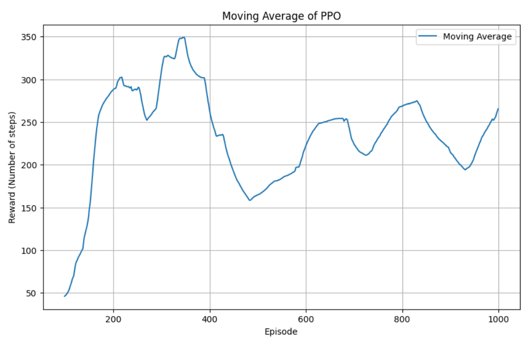
(e) *Learning curve of Actor-Critic model*



(f) *Moving Average - Actor-Critic model*



(g) *Learning curve of PPO*



(h) *Moving Average - PPO*

Figure 3: *Learning curves and moving averages of different models*

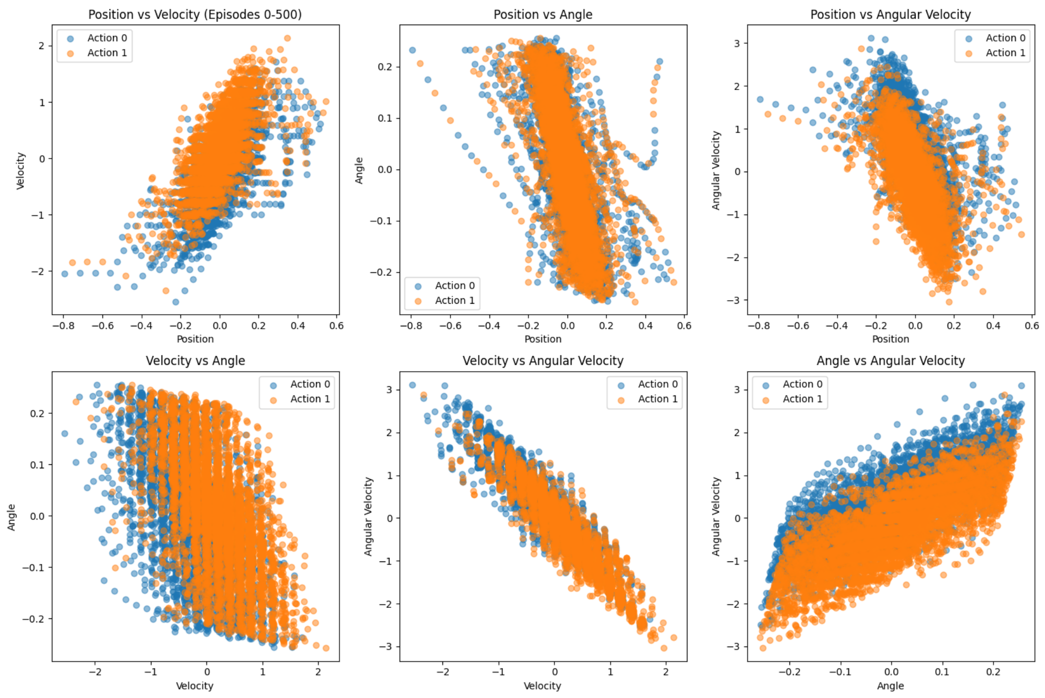


Figure 4: *State-action scatter plot of Discrete Q-Learning in episode 0-500*

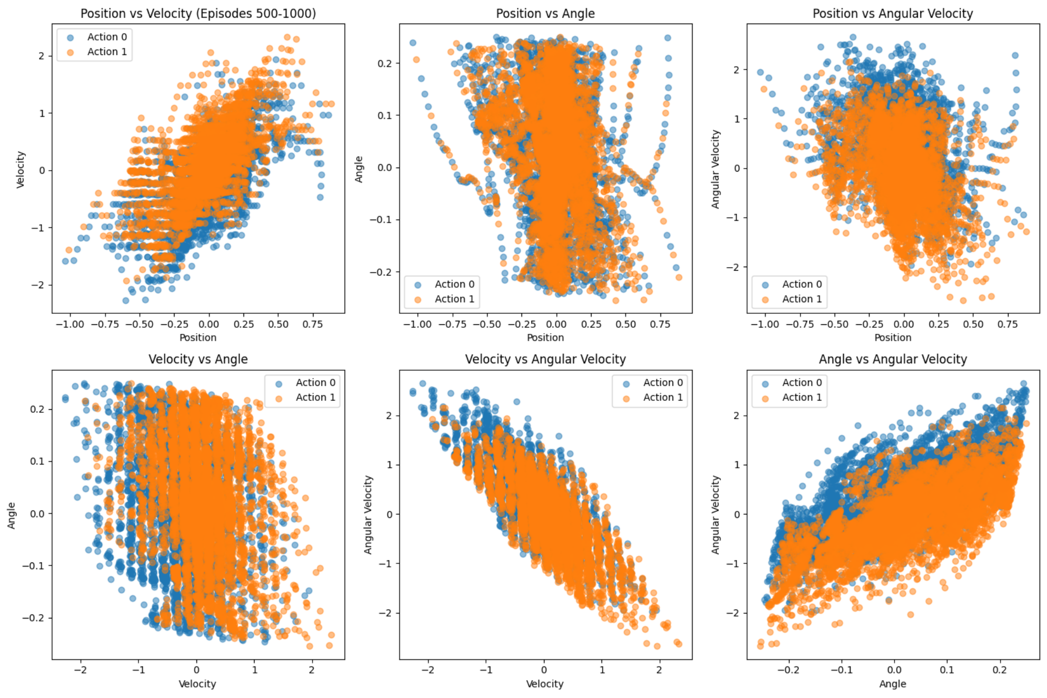


Figure 5: *State-action scatter plot of Discrete Q-Learning in episode 500-1000*

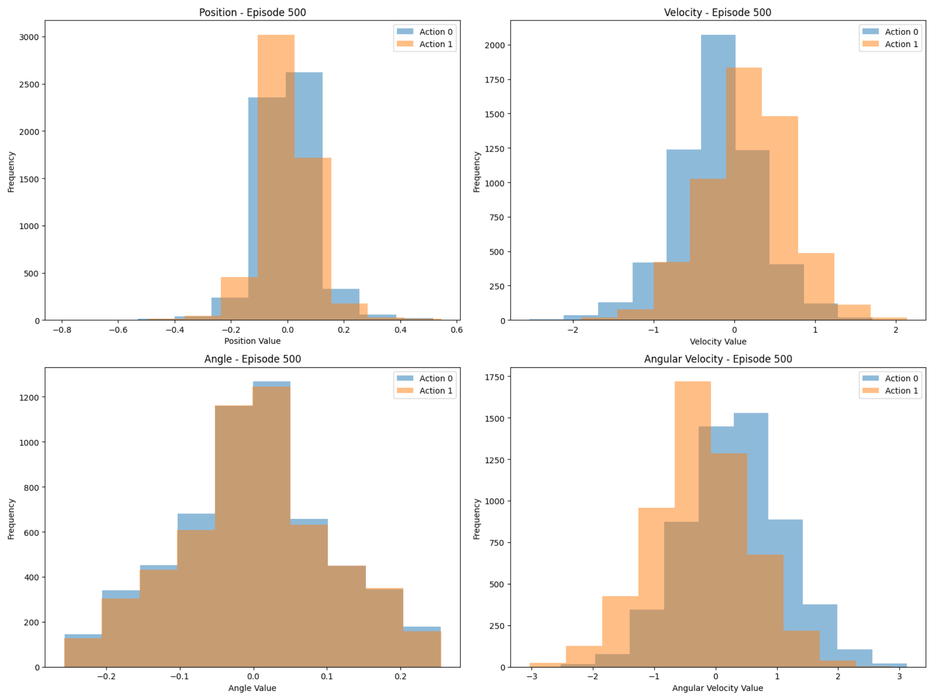


Figure 6: *State-action histogram of Discrete Q-Learning in episode 0-500*

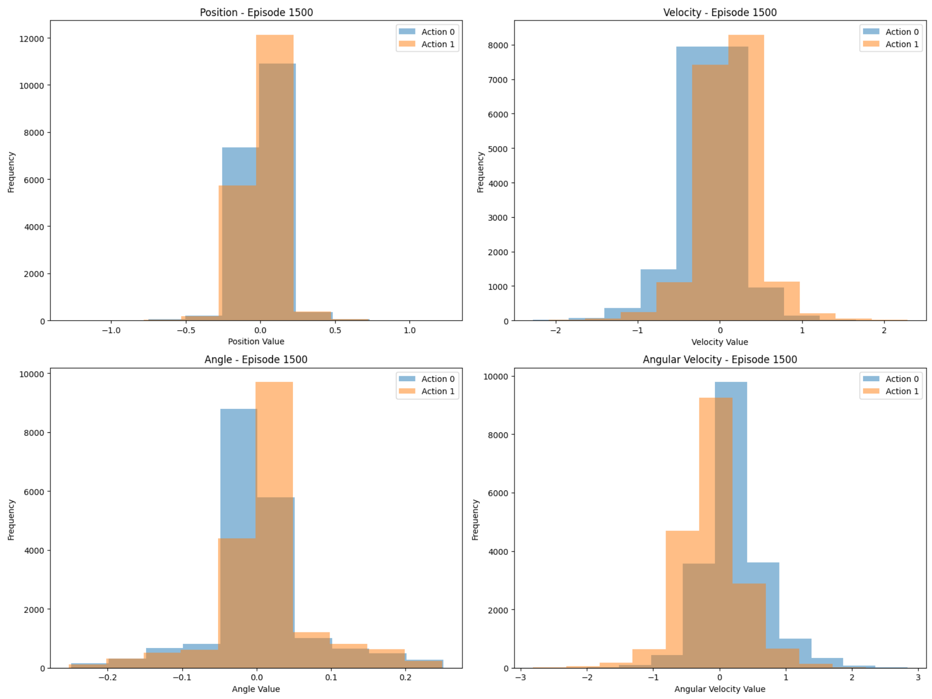


Figure 7: *State-action histogram of Discrete Q-Learning in episode 500-1000*

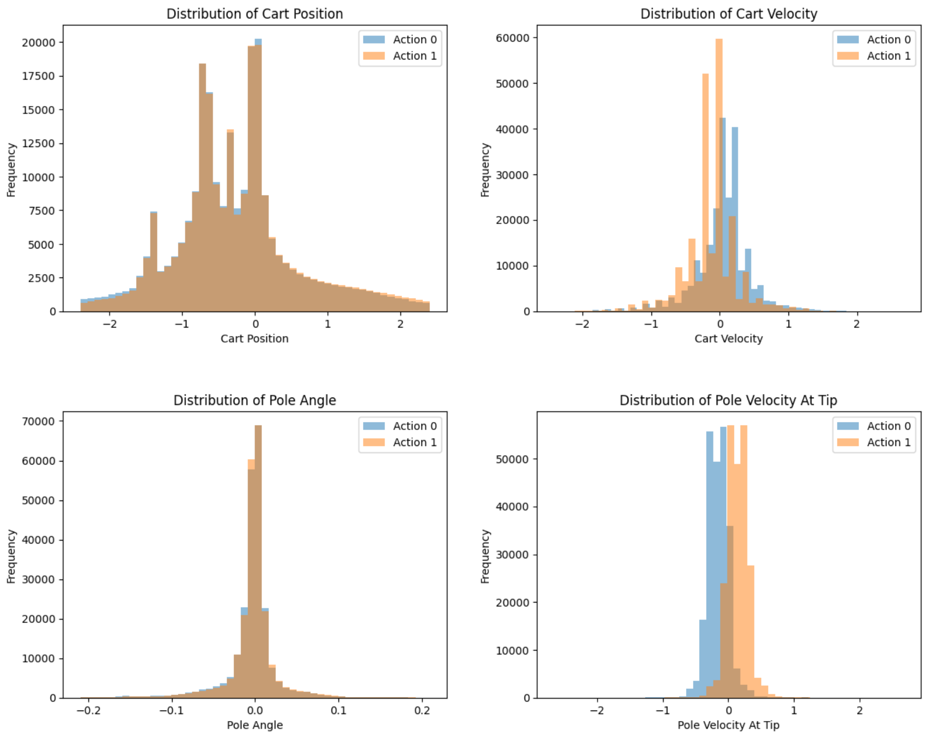


Figure 8: *State-action histograms of Deep Q model*