Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning

Clément Bernard & Aladji Faye
Department of Computer Engineering and Software Engineering
Artificial Intelligence : INF8225
Polytechnique Montreal
Github link : https://github.com/clementbernardd/Count-Based-Exploration

1 Introduction

Exploration is an important aspect of reinforcement learning. It enables a model to explore actions that couldn't be considered by the model but could lead to future high rewards. The simplest method is the famous ϵ-greedy where the agent explores with probability ϵ and exploit with probability $1-\epsilon$. Nevertheless, such a method has some disadvantages : it doesn't take into account what is learnt or the visited states of the agent. Another method, which is the main focus of our work, is the *count-based exploration*. This method is said to be very efficient for tabular RL methods. The aim of our work is to prove that it can also be applied to continuous and high-dimension environment. The idea of the method is to map a continuous state into a hash code in order to count the occurences, and then modify the rewards in order to add a bonus for the states not frequently visited. Our work aims to explore this method and to compare it with the ϵ-greedy exploration with both tabular and deep RL algorithms such as Q-learning, Sarsa, Deep Q-learning, Double DQN, Dueling DQN.

2 Related Work

In this section, we first present the principle of reinforcement learning, and classical machine learning methods that are commonly used. We will then use these methods in the exploration scheme.

Reinforcement learning (Sutton & Barto (1998)) defines the artificial intelligence methods allowing an agent to learn through a loop of actions and rewards (hence "reinforcement"). Reinforcement learning methods have been implemented in the AI world for several decades now and are still thoroughly studied as of today, for instance in video games. There are many ways to implement reinforcement learning, some of which will be presented as follows. First, the tabular methods for discrete states space : Q-learning algorithm that uses a *Q function*, which is at each step updated and used to choose the best actions. (Sutton & Barto (1998)). Then, the SARSA which is similar to Q learning, the difference being in the choice of the action and the update of the Q Value at each step. (Sutton & Barto (1998)).
But the tabular methods doesn't work when the states space is continuous, which is the case in most of RL environments. To bypass this, one can use deep reinforcement learning ((Agostinelli et al. (2018))), which is, as its name suggests, "simply" the combination of reinforcement learning and deep learning (LeCun et al. (2015)). It enables to both deal with continuous states and to approximate Q-function for instance. Deep reinforcement learning can be found in AlphaGo, the famous agent that beat the Go world champion in 2014. One of the famous deep RL algorithm is the extention of the Q-learning algorithm : the DQN or *Deep Q-Learning*. The neural network approximates the Q-table in order to pass as inputs continuous states. Other algorithms exist, that tend to improve the classic DQN like the Double DQN (that deals with the unstable training by overestimating rewards). Basically, the update of the Q function is done using *two Q functions* (van Hasselt (2010)). Another algorithm would be the Dueling DQN where the Q function is decomposed between a *Value function* and an *Advantage function*. Thus, it gives numerical values to, respectively, the reward of being in a precise state and how good choosing an action is compared to other actions. (Wang et al. (2016)).
One of the main bottleneck of RL is the exploration. It enables the agent to discover states that its current policy doesn't allow, and can lead to surprising discovery. Some methods that worked well were to use uniform sampling (Mnih (2015)) and Gaussian noise (Lillicrap & Wierstra (2015)). Some of the classic justified exploration are based on state-action counts and add a bonus reward.

The UCB algorithm (Lai & Robbins (1985)) for instance chooses the action a_t that maximises the reward $r(\hat{a_t}) + \sqrt{\frac{2logt}{n(a_t)}}$ where $r(\hat{a_t})$ is the estimated reward and $n(a_t)$ the number of times the action a_t has been chosen. Nonetheless, there are some situations where the rewards are quite sparse and simple exploration isn't as efficient as it should. Recently, some deep RL algorithms use explorations methods that enable them to perform well, like Boostrapped DQN (Osband & Van Roy (2016)). Therefore this work aims to implement a state-count based exploration that can be extended to deep RL with the help of a static hashing, like the SimHash from Charikar (2002).

3 Theory

First of all, we consider the classic reinforcement learning notations. We assume a finite discounted Markov Decision Process (MDP) with the following notation : (S, A, P, r, p_0, γ , T) with S the state space, A the action space, P a probability distribution, r : S x A $\rightarrow$ R, p_0 an initial state distribution, $\gamma \in$ [0,1] a discount factor and T the horizon.

3.1 Count-based exploration

The approach of the article Haoran Tang (2017) is to discretize the spaces and add a term into the reward function. It uses a hashing function $\phi : S \rightarrow Z$, where the given hashing function is explained just after. After having discretize the state, a count function is incremented by one each time the state is visited. This count function is then used to add a reward to the classic reward function : $r^+ : S \rightarrow R$ with the given formula :

$$r^+(s) = \frac{\beta}{\sqrt{n(\phi(s))}}$$

with $\beta \geq 0$ the bonus coefficient. At each time-step, the given state s_t visited will increase the value $n(\phi(s))$. After that, the current agent is trained with the new reward $(r + r^+)$, but the evaluation of the training process is done with only the classic reward. Indeed, the article wants to see the effects of the exploration on the classic reward to see if the decisions made by the help of exploration led to better rewards.
The article used a state-count base, but they mention the fact that a state-action count base didn't work well. Therefore, we didn't try the state-action count base, but only explore the state-count based.
The main bottleneck of this exploration method is the choice of the hashing function. Indeed, it could lead to a bad discretization and map non close state in the continuous space into same representation in the discrete space. It will then lead to a bad representation of the count of the states, and therefore the new reward added won't have a good estimation of state count.

3.2 Static Hashing

The hashing that maps continuous states into discrete states is the main factor of this exploration method. As the deep RL aims to take advantage of continuous states in order to get the maximum information available, the hashing will do the contrary : map the states which are close in continuous space into the same discrete representation in order to compress the states.
The methods that are used in the paper are either the static hashing or the learning hashing. The first one won't evolve during the training process, whereas the learning hashing aims to learn the hashing itself during training. We consider only the static hashing for our work, as the learning hashing is used for ATARI games that are very long and expensive to train.
The hash function that we implemented aims to merge similar states in continuous space and make distant states counted separately. We could have added a prior over the environment in order to improve the ϕ function, but we didn't do it.
The hashing function used retrieves a binary code for a given state $s \in S$:

$$\phi(s) = sgn(Ag(s)) \in \{-1, 1\}^k$$

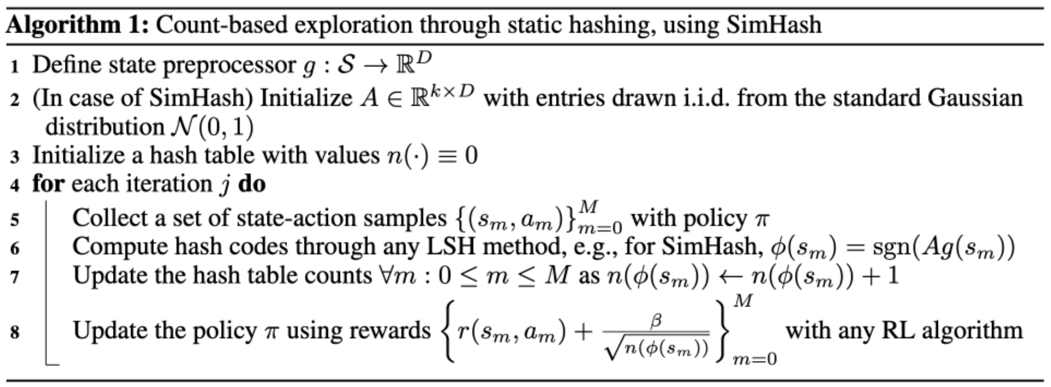
with g : $S \rightarrow R^D$ a pre-processing function (which can take into account prior over the states), and A a $k \times D$ matrix with entries drawn from a standard Gaussian Distribution $\mathcal{N}(0, 1)$. The k factor is a hyperparameter of the algorithm that deals with the granularity : the higher the value of k, the fewer the collisions occur between states. There remains a trade-off : we want to have enough granularity to differentiate the distant states, but not too much granularity to merge the close states. For our experiences we didn't use a pre-processing function, which means our static hashing was the following one :

$$\phi(s) = sgn(As) \in \{-1, 1\}^k$$

We didn't get very well if the given state that is feed to the ϕ function was a state after few linear layers, or if it was the original state found in the observation space. Therefore, we tried both and we kept the state that can be found directly in the observation space by openAI gym libraries.

4 IMPLEMENTATION PLAN

The main algorithm that we used for our work is shown in Figure 1



```
Algorithm 1: Count-based exploration through static hashing, using SimHash
1 Define state preprocessor $g : \mathcal{S} \to \mathbb{R}^D$
2 (In case of SimHash) Initialize $A \in \mathbb{R}^{k \times D}$ with entries drawn i.i.d. from the standard Gaussian
    distribution $\mathcal{N}(0, 1)$
3 Initialize a hash table with values $n(\cdot) \equiv 0$
4 for each iteration $j$ do
5     Collect a set of state-action samples $\{(s_m, a_m)\}_{m=0}^{M}$ with policy $\pi$
6     Compute hash codes through any LSH method, e.g., for SimHash, $\phi(s_m) = \mathrm{sgn}(Ag(s_m))$
7     Update the hash table counts $\forall m : 0 \leq m \leq M$ as $n(\phi(s_m)) \leftarrow n(\phi(s_m)) + 1$
8     Update the policy $\pi$ using rewards $\left\{ r(s_m, a_m) + \frac{\beta}{\sqrt{n(\phi(s_m))}} \right\}_{m=0}^{M}$ with any RL algorithm
```

Figure 1: Count-based exploration algorithm

It leads freedom of the choice of the deep RL algorithm. The paper uses the Trust Region Policy Optimisation (TRPO). Nonetheless, we found quite hard to use this algorithm as we didn't know this one and the tutorials on the internet aren't abundant. We decided to use another algorithms : Q-learning and SARSA for the tabular methods, and DQN, Double DQN and Dueling DQN for the deep RL implementations. Our goal is the compare the epsilon-greedy exploration policy used in all these algorithm with the count-based exploration of the article.
The environments that we used was the basic ones of the openAI gym library. Here is few explanation of each of those :

1. **Acrobot** : task that includes two joints and two links and where the aim is to swing the end of the lower link up to a given height. The states are continuous of size 6, and there are 3 actions to perform.
2. **Cartpole** : task where a joint is attached to a car. The system is controled by given a force of +1 or -1 to the car, and the aim is to keep the pendulum upright. The episode ends when the pole is more than 15 degrees from vertical or the cart moves more than 2.4 units from the center. The state size is continuous of size 4, whereas there are 2 actions available.
3. **Mountain Car** : task where a car is positioned between two mountains. The aim is to drive up the mountain on the right, and the car isn't strong enough to drive in a single pass. There are 3 actions available and the states are continuous of size 2.

We used both Q-learning and SARSA for taxi and acrobot environments (where we discretize by hand the acrobot states uniformly), and DQN, DDQN and Dueling DQN for acrobot, cartpole and mountain algorithms. For each experimentations, we used both the ϵ-greedy exploration with $\epsilon = 0.1$ and the count-based exploration method. We don't report the results with the tabular algorithms as the results aren't interesting : the two exploration methods work the same.

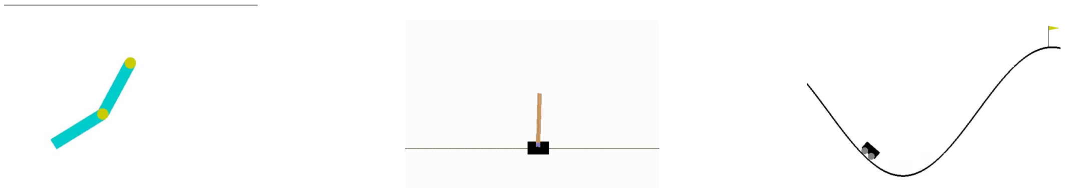


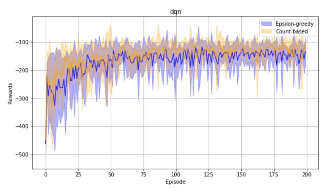
Figure 2: OpenAI environments used : Acrobot, Cartpole and Mountain Car

5 Results

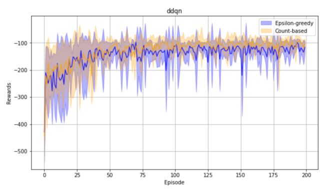
To compare the different models with the exploration methods, we used the same hyperparameters for both ϵ-greedy and count-based exploration. The hyperparameters for the different environments are in the Appendix. For all the environments, we used the same hyperparamters for 10 different seeds : [77,7,17,75,73,2,11,89,99,6]. We stored the rewards for all the 10 experiments, and then plot in plain line the average, and in background the variance for the experiments.

5.1 Acrobot

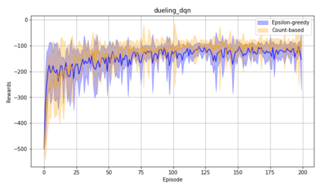
This environment has continuous states, which means that classic tabular methods doesn't work. We used deep RL algorithms such as DQN, DDQN and Dueling DQN, with hyperparameters that are in the Appendix.



(a) DQN



(b) DDQN



(c) Dueling DQN

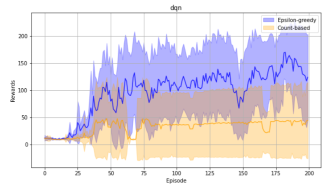
Figure 3: Rewards for the acrobot environment

For this method, the two different exploration methods have the same behaviour. We can see that for DQN and DDQN, the count-based method is slightly better than the ϵ-greedy method. It means that this method also works for the continuous state space, which is the expected behaviour.
The mean rewards for 10 instances of the environment is on Figure 9. It shows that the best model is the Dueling DQN with count-based exploration. It means that the exploration with the count-based method is efficient to solve the given environment. Note that the model to test was the model with SEED 77, which means that this is possible that for this SEED the training has been less efficient than the other SEEDS. It therefore explains the behaviour of the DQN count based model for this graph.

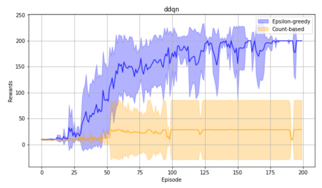
5.2 Cartpole

For this environment, the states are still continuous. The game ends quite quickly. The rewards are shown on Figure 4.

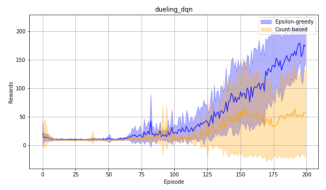
We see from the reward curves that the count-based exploration doesn't learn well for this environment. For all the models, the ϵ-greedy exploration outperforms the count-based method such as the count-based method doesn't solve the environment. To better understand this phenomenon, let's have a look to the count states histogram on Figure **??**. Unfortunately, we don't see any weird distribution. As the histogram is with logarithm scale, it hides the fact that some states are a lot more visited than other ones. One reason for the bad learning could be the static hashing that is used



(a) DQN



(b) DDQN



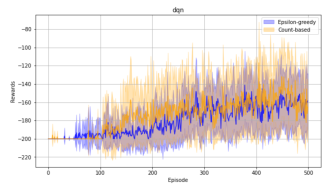
(c) Dueling DQN

Figure 4: Rewards for the cartpole environment

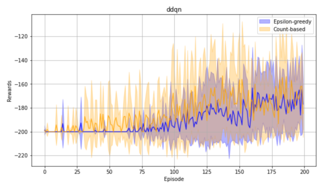
: this is the main bottleneck of this method. Nonetheless, we didn't find obvious reasons with the distribution plot. The results on the mean rewards infers the same conclusions (on Figure 9).

5.3 Mountain car

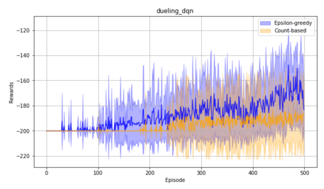
The main challenges of this environment is to explore well the states in order to find the states that don't have immediate high rewards, but enable the car to gain speed in order to climb the hill.
The reward plot is on Figure 5.



(a) DQN



(b) DDQN



(c) Dueling DQN

Figure 5: Rewards for the mountain car environment

The behaviour is unexepected : the count-based method outperforms the baseline for both DQN and DDQN (it learns better and faster), but not with the dueling DQN algorithm.
The histograms of the state count (on Figure 8) shows that the dimension of the discretize state is 4 times more with the Dueling DQN method than with the DQN and DDQN. It could explain the fact the the Dueling DQN method isn't as high-performance as both the DQN and DDQN. Indeed, the static hashing could here be too sparse and don't map well the close states in the continuous space. Nonetheless we tried with lower value of k but it didn't learn well too.
The results on the mean rewards plot (Figure 9) shows that the best model is the DQN with ϵ-greedy, followed by the DDQN count based method. It means that for this environment, it wasn't quite obvious if an exploration method is better than the other one.

6 Conclusion

In this work, we explored a tabular exploration method that could be extended for continuous state space. The count-based exploration aims to explore by adding a reward that considers the number of times the agent has visited a given state. Such a state that has been mapped with a hashing function in order to convert it from continuous space into discrete space. The static hashing is the main bottleneck of this exploration method : it should compress enough in order to map close states in the continuous space into the same discrete representation. On the other hand, it should be sparse enough to keep far away distance states. We achieved to implement such exploration and succeeded in outperforming ϵ-greedy exploration for the acrobot and mountain car environments, but we didn't succeed with cartpole and the Dueling DQN algorithm with mountain car environment. Such conclusion leads to the fact that the choice of hashing function is really important. A good improvement could be the learning hashing mentioned in the paper, which is much more challenging as the hashing is changing during the training process. Another limitation of our work is the fact that we didn't implement the same deep RL algorithm than the paper (TRPO). Therefore, the comparison with the papers are limited, added to the fact that we didn't implement all the environments that are too expensive to train.

References

Forest Agostinelli, Guillaume Hocquet, Sameer Singh, and Pierre Baldi. From reinforcement learning to deepreinforcement learning: An overview. *Lecture Notes in Computer Sciences*, 2018.

Moses S. Charikar. Similarity estimation techniques from rounding algorithms. 2002. URL `https://www.cs.princeton.edu/courses/archive/spr04/cos598B/bib/CharikarEstim.pdf`.

Davis Foote Adam Stooke Xi Chen Yan Duan John Schulman Filip De Turck Pieter Abbeel Haoran Tang, Rein Houthooft. exploration: A study of count-based exploration for deep reinforcement learning. 2017. URL `https://arxiv.org/pdf/1611.04717v3.pdf`.

Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules.advances in applied mathematics. 1985.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

Hunt Jonathan J Pritzel Alexander Heess Nicolas Erez Tom Tassa Yuval Silver David Lillicrap, Timothy P and Daan Wierstra. Continuous control with deep reinforcement learning. 2015. URL `https://arxiv.org/pdf/1509.02971.pdf`.

Kavukcuoglu Koray Silver David Rusu Andrei A Veness Joel Bellemare Marc G Graves Alex Riedmiller Martin Fidjeland Andreas K Ostrovski Georg et al. Mnih, Volodymyr. Human-level control through deep reinforcement learning. 2015. URL `https://storage.googleapis.com/deepmind-data/assets/papers/DeepMindNature14236Paper.pdf`.

Blundell Charles Pritzel Alexander Osband, Ian and Benjamin Van Roy. Deep exploration via bootstrapped dqn. 2016. URL `https://arxiv.org/pdf/1602.04621.pdf`.

Richard S. Sutton and Andrew G. Barto. Reinforcement learning : an introduction. *MIT Press*, 1998.

Hado van Hasselt. Double q-learning. *NeurIPS 2010*, 2010.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. *ICML'16: Proceedings of the 33rd International Conference on International Conference on Machine Learning*, 2016.

7 Appendix

7.1 Hyperparameters

Here are the hyperparameters used for the training of the models for the Acrobot environment. The hyperparameters for the ϵ-greedy exploration policy are the same (without the count-based hyperparameters) with $\epsilon = 0.1$. Note that for all the environments the hyperparameters for the ϵ-greedy exploration method are the same with just $\epsilon = 0.1$.

The hyperparameters for the cartpole environment are summarized in Table 2

The hyperparameters for the mountain car environment are summarized in Table 3

7.2 Histogram of count-based states

Here are the histograms of the count-based states for the exploration method of the paper.

The histograms of the count-based states for the cartpole environment is shown on Figure 7.

The histograms of the count-based states for the mountain car environment is shown on Figure 8.

The mean rewards for 10 instances of the environment is shown on the following Figure :

Model	Q-learning	SARSA	DQN	DDQN	Dueling DQN
Learning rate α	0.01		1e-3		
Discount factor γ	0.99				
Buffer size			500		
Hidden units			(64,64)		
Batch size			32		64
Optimizer			Adam		
Update every				10	
Count-based β	0.1				
Granularity k			16		

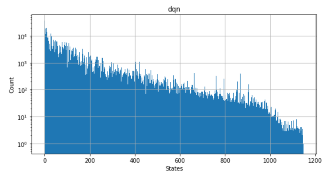
Table 1: Hyperparameters for acrobot environment

Model	DQN	DDQN	Dueling DQN
Learning rate	1e-3		
Discount factor γ	0.99		
Buffer size	500	1000	5000
Hidden units	(64,128)	(64,64)	(128,64)
Batch size	64	32	64
Optimizer	Adam		
Update every		50	10
Count-based β	0.1		0.9
Granularity k	32		

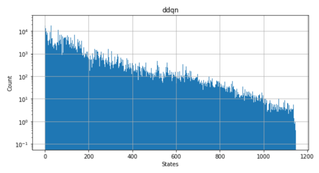
Table 2: Hyperparameters for cartpole environment

Model	DQN	DDQN	Dueling DQN
Learning rate	1e-3		
Discount factor γ	0.99		
Buffer size	1000	5000	10000
Hidden units	(64,64)		(64,128)
Batch size	64	32	64
Optimizer	Adam		
Update every		50	10
Count-based β	0.1	0.9	
Granularity k	16	32	64

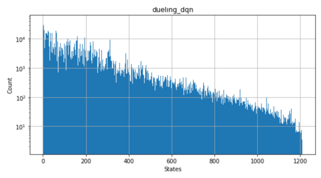
Table 3: Hyperparameters for Mountain car environment



(a) DQN

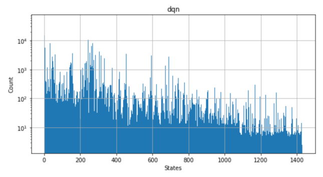


(b) DDQN

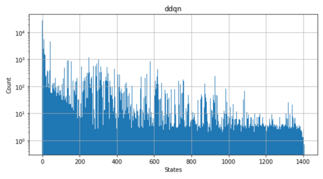


(c) Dueling DQN

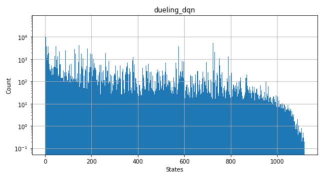
Figure 6: Count state distribution for the acrobot environment



(a) DQN

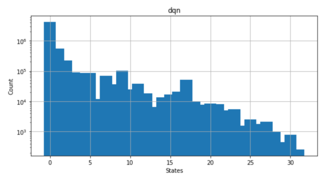


(b) DDQN

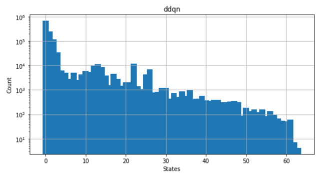


(c) Dueling DQN

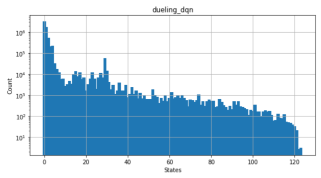
Figure 7: Count state distribution for the cartpole environment



(a) DQN

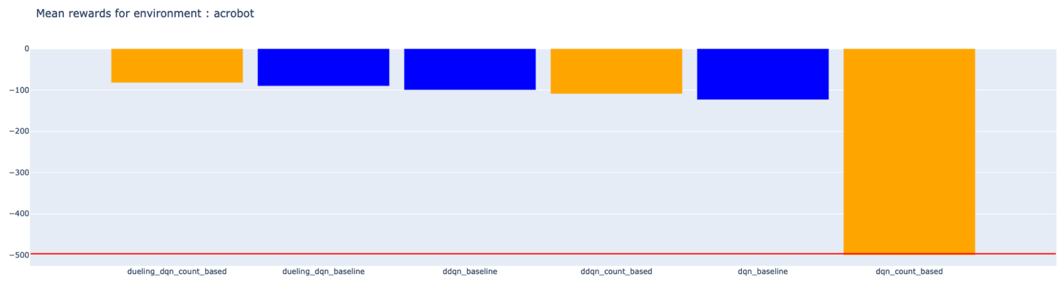


(b) DDQN

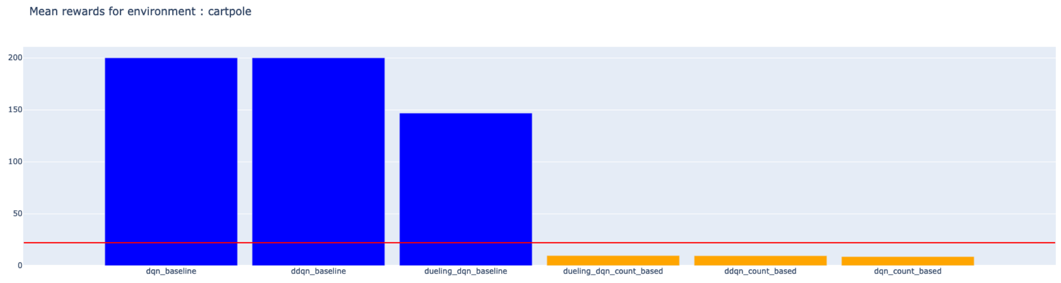


(c) Dueling DQN

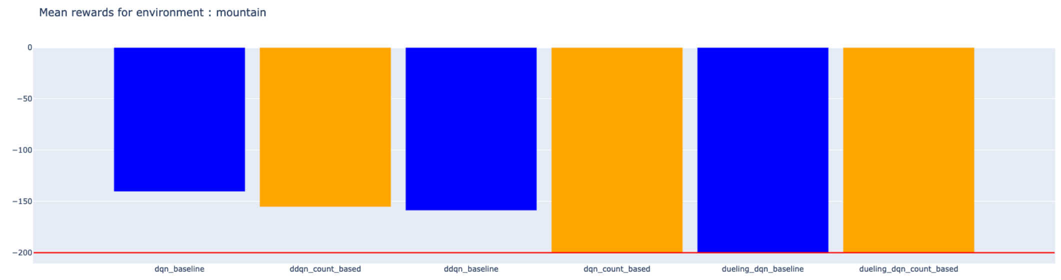
Figure 8: Count state distribution for the mountain car environment



(a) Acrobot



(b) Cartpole



(c) Mountain Car

Figure 9: Mean rewards for 10 instances of environment for all the models of SEED 77 (blue for -greedy and orange for count-based exploration)