Navigation Project

A Deep Reinforment Learning case study : Unity Banana environment with DQN

Gabriel Colas

2022

Abstract

The pairing of deep neural network and Reinforcment Learning and its improvements has outperformed the historical methods on a large panel of environment. In this work we applied several state-value based techniques borrowed from this framework as the famous DQN, Dueling DQN and Double DQN to the Unity game environment Banana as benchmark. We have demonstrated great performances under the same hyper-parameter. Indeed, the game environment is considered solved for each method tested with a fast learning curve.

1 Introduction

Reinforcment learning is a framework that allows a smart agent to act with a provided environment and find the optimal policy within the environment. Since we don't know the intrinsic dynamic of the Markov Decision Process we discover the policy by sampling its dynamic. Thus, the historical idea derived from dynamic programming is based on the famous recursive Bellman Equation that led to a variety of algorithms commonly used for a learning agent [O'Donoghue et al., 2018]. In [Mnih et al., 2013], there has been a huge progress for dealing with large continuous states and actions. Combining a deep neural network and the usual Q-learning algorithm has considerably enhanced the benchmark scores on the suite of controlling ATARI games and outperformed significantly human-based performance. Since the kick-start of this new Deep-RL field many improvement and variant have been suggested and outperformed the DQN algorithm.
In this case study, we will try several state-value techniques based on the DQN algorithm on the Banana Unity environment and we will show their performance.

2 Theory

During this work, we made several attempts to find the optimal policy for the Banana environment. We describe here the different algorithms and underlying theory we used.
We consider a smart agent that take an action A_t in an environment at each time step t. The agent interact with this environment that return an observation O_t, which we consider for simplification to be its internal states S_t. The environment is formally described by a Markov Decision Process where the current state S_t completely characterize the process. From the Recursive Bellman Equation for the action-value $Q(s, a)$ we derived the off-policy learning algorithm Q-Learning. Originally adapted for Sequential Learning of tabulars environment where the states are discrete we approximate in our case the algorithm for continuous state. Thus, we use function approximation where we consider

$Q(s, a)$ to be a differentiable non-linear function. We approach the continuous states with the help of a Neural Network architecture that maps the states to action. We learn its weights by the action-in procedure with a Q-learning weight update as :

$$\Delta w \propto (R_{t+1} + \gamma max q_w(S_{t+1}, a) - q_w(S_t, A_t) \nabla_w q_w(S_t, A_t)$$

Then we apply the few popular tricks as a fixed Q-target q_{w-} and experience replay to improve the stability and convergence of the algorithm. The technique described here, called DQN in [Mnih et al., 2013] have create great success in this field outperforming the previous methods.

Also, in this work we tried a different implementation of the DQN algorithm, the Double DQN algorithm. Its goal is to tackle the instrisic overestimation problem relative to to the DQN algorithm. In practice we have seen that this new algorithm perform better and is a step closer to the optimal policy we are looking for. So we change the Q-learning weight update equation by :

$$\Delta w \propto (R_{t+1} + \gamma q_{w-}(S_{t+1}, argmax q_w(S_{t+1}, a)) - q_w(S_t, A_t) \nabla_w q_w(S_t, A_t)$$

We have substitute the max operation by an argmax operation and change the q update by a q-update.

Finally we tried the Dueling DQN with a different network architecture [Wang et al., 2016]. The idea between this alternative is to separate the actions mapping estimations in two separate network. The first one is used to estimate the action-value and second one the state-action anomaly value. This algorithm as shown great performances in the papers.

3 Methods

For these experiments the program run in a laptop with a GPU GTX 1050. We learned our neural networks with Torch and the GPU for python 3.6. We give open access to the python code in the Git-Hub [student, 2022].
The environment provides a continuous vector observation of space size 37 as input of our simple neural network architecture. In the output 4 discrete actions are possible. For each algorithm we have 3 commons perceptrons layers with 64 units for the first one and the second with RELU activation. The last one has 4 neurons to correspond to the 4 discrete actions with a softmax activation.
For the dueling architecture, the network has two commons layers with the last architecture. Then two branch separate with the value estimate branch composed of two hidden layer, one of 16 units and one of one unit and an advantage function estimate branch composed of two hidden layer, one of 16 units and one with 4 units and 4 outputs.

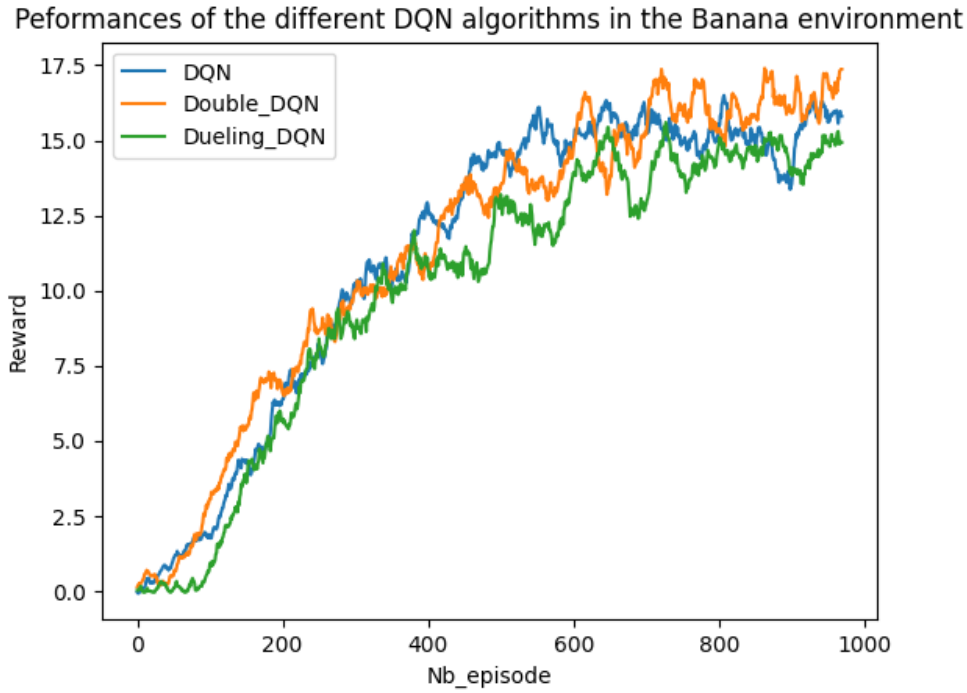
After several attempts we have kept the different hyper-parameters for the different algorithms as following :

	γ	Batch size	Replay-buffer size	α	Network update step	Soft update target network τ
DQN	0.99	64	$10e5$	$10e-4$	4	$10e-3$
Double DQN	0.99	64	$10e5$	$10e-4$	4	$10e-3$
Dueling DQN	0.99	64	$10e5$	$10e-4$	4	$10e-3$

So we got a large replay buffer with $10e5$ samples with the most oldest samples automatically discarded when the buffer is full. From this buffer we update our local network parameters each 4 step with a small learning step of value $\alpha = 10e - 4$ taken on a 64 sample batch size. Finally, the parallel target network parameters are updated by a soft update, a mixture of target network parameters and local network parameters with a $10e - 3$ weight at each 4 time step for the target network parameters.

4 Results and Analysis

We present the performances of our optimal value-based algorithms according to the parameters drawn from the table last section. We see that for every algorithm, the environment is considered as solved (It needed a reward > 13), and the final performances are quite similar. Double DQN seems to have a small advantage over the algorithms but it remains anecdotal. Also the learning curves are quite look alike. We notice a small delay for the dueling network that seems to have more difficulty at the beginning to begin its learning.



5 Conclusion

We have tried several state of the art deep-RL algorithm based on the literature for continuous states value-based method. Through value approximation with a deep neural network, we have solved the environment and have shown similar performances for the different algorithms we tried in this work.

References

Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. In *International Conference on Machine Learning*, pages 3836–3845, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

Gabriel PHD student. Project navigation. https://github.com/GabrielLinear/RL_Learning, 2022.