Training Reinforcement Learning Agents with Double Deep Q Networks for Super Mario Bros

Manh Hung Nguyen and Julio Candela Cáceres
CentraleSupélec

1 INTRODUCTION

Naturally, the learning process implies discovering what to do in a specific situation based on the positive and negative feedback obtained by interacting with the environment. Reinforcement Learning emulates this idea into an agent which automatically learn what actions to take to maximize its reward [11]. One of the most popular algorithms is Q-Learning which obtains an optimal Q-function based on the rewards and the next states [12]. However, the learning task becomes complex for Q-learning in real-world environments due to the high dimensionality of inputs and actions [19].

Deep Q Networks (DQN), which combines deep learning and Q Learning, emerged as a solution to build a flexible artificial agent that can learn from complex input representations. One of the most frequent use cases in Deep Reinforcement Learning is the set of Atari 2600 games. Since it involves learning a policy from image frames of a video game, Convolutional Neural Networks (CNN) appear as a perfect option to represent the input states in this type of challenges [8][19][12]. Nevertheless, the training of the neural networks, including CNN, was not accurate due to the instability and divergence of these non-linear models in learning tasks[8].

Both Q-Learning and DQN used to overestimate the value function, especially, in larger problems with uncertain rewards as proved by Hasselt et al [15] in later experiments. The overestimations increased with the number of iterations yielding to a deterioration of the policy and, in consequence, worse rewards. Thus, Hasselt et al [15] proposed Double Deep Q Networks (Double DQN) to deal with the overestimations and find better policies. The Double DQN was implemented as the DQN with some minor variations in the update. Basically, it consists of an online network to select the best action and a target network to evaluate the new value function based on the selected action. This approach, indeed, reduced the overestimation and increased the reward and stability of the learning surpassing DQN and human results

The purpose of this work is to implement one of the current state of the art algorithms, Double DQN, and evaluate its performance in the famous game Super Mario Bros. We want to show the benefits of using CNN in complex and high-dimensional problems in Reinforcement Learning. Then, the Double DQN will be presented to deal with some of the deficiencies in DQN so that the agent can learn better policies and maximize the rewards.

2 RELATED WORKS

Our work focuses on the research of Hasselt et al [15] who proposed a Double DQN to deal with the overestimations of DQN. In fact, the Double DQN agent surpassed the results of simple DQN and humans in most of the tests. Previously, Minh et al [8] had also proposed to incorporate an experience replay and a target network improving the results of DQN.

Ziyu Wang et al proposed Dueling DQN - a new architecture with two networks/estimators, one for the state value function and the other one for the state-dependent action advantage function [16]. Schaul et al improved DQN by introducing prioritized experience replay which tries to increase the replay probability of important experiences during training the agents [10].

Other studies suggests the use of actor critic methods on the Atari domain. Mnih et al [7] proposed an asynchronous variant of actor-critic which consisted of parallel actor-learners providing an stabilizing effect on the training of neural network controllers. Furthermore, Gruslys et al [3] also proposed a Retrace actor-critic architecture for off-policy learning which improves the real-time performance. Wu et al [18] propose the combination of both actor-critic methods and Dueling networks to mitigate the learning problems in stochastic environments.

3 BACKGROUND

3.1 Q-Learning

Q-learning [17] is one of the most popular reinforcement algorithm. In a sequential decision making problem, agents need to learn how good it is to choose a specific action to interact with the environment. They can learn it by evaluating the expected sum of future rewards R_i when taking that action and following the optimal policy thereafter. To be more specific, the action value of an action a in a state s under a specific policy π could be defined as follows:

$$Q_\pi(s,a) = E[R_1 + \gamma R_2 + ...|S_0 = s, A_0 = a, \pi] \ ,$$

where $\gamma \in [0,1]$ is the discount factor. This hyper-parameter specifies how important delayed rewards are. The update of Q-learning is:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha(R_t + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t))$$

where $Q_t(s,a)$ is the action value of a in state s at the time step t. R_t is the reward the agent receives when go from state s to state s_{t+1} by taking action a. The discount factor γ can be used not only to weight immediate rewards more heavily than later rewards, but also to guarantee the convergence. The learning rate $\alpha \in [0,1]$ specifies how much the agents should update the action values in each iteration.

3.2 Deep Q Networks

Real-life problems often have extremely large state space and action space, which makes it almost impossible to learn all the action values due to limited computational resources. One approach is to learn a parameterized function $Q(s, a; \theta_t)$ to estimate action values. Neural networks are well-known with a powerful ability of approximating a function. This leads to an approach of using multi-layered Neural Networks to approximate the Q-function [9], which is called Deep Q Networks (DQN). For a given state s, the DQN outputs a vector of action values. The training data are sampled uniformly from a experience buffer updated during many episodes. The idea of using this buffer is to de-correlate observation sequence. A second network called target network is used to have a fixed target instead of a moving target. This target network has its parameters copied periodically from the online network to avoid divergence.

The Figure 1 shows an example of a DQN with convolution neural networks. The input of this DQN is state representation of raw pixels from Atari 2600 games and the outputs correspond to the estimated Q-values for each possible actions starting from the input state. The method was able to learn good policies in a wide variety of games without any prior knowledge and based only on the visual images.

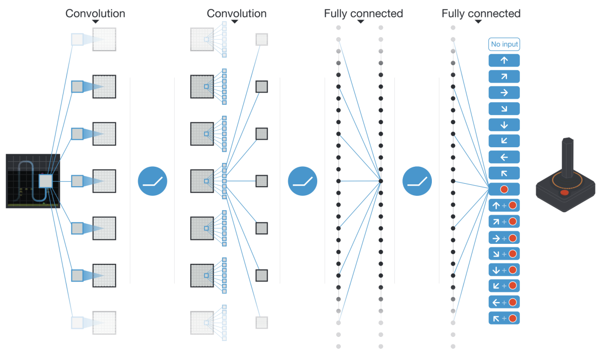

Figure 1: DQN with convolutional neural networks [9]

3.3 Double DQN

Thrun and Schwartz were the first ones who studied Q-learning's overestimations [13]. They have showed the evidences of overestimations lead to sub-optimal policies. In many cases, Q-learning tends to overestimate the action values due to using the max operator for the same action values to select and to evaluate actions. This results in overoptimistic value estimates. The same happens in DQN. In each update of DQN, the target is defined as:

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a, \theta_t); \theta_t)$$

Hado Van Hasselt et al proposed an idea of using two esimators to decouple the selection from the evaluation [14] as a solution to overestimations. Two different sets of weights θ and θ' are learnt based on experience randomly. These set of weights can be updated symmetrically by switching their roles. In each update, one set of weights is used to choose the action to be evaluated and the other set of weights is used for fairly evaluating its action value as follows:

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a, \theta_t); \theta_t')$$

The idea of Double Q-Learning has been implemented using Neural Networks called Double Deep Q Networks (Double DQN) by Hado Van Hasselt et al to take advantage of both Double Q-Learning and Neural Networks [15]. Two networks trained with experience replay are used for action selection and action evaluation separately.

The Figure 2 shows the experiments done by Hado Van Hasselt et al to show overestimations do happen in real-life cases and they negatively affect the resulting policies. DQN and Double DQN were trained with same conditions in Atari games. We can clearly see that in some games, values estimated by DQN (in red) are higher than ones estimated by Double DQN (in blue) over the training steps. In the bottom row, the score of these DQN and Double DQN agents in two game *Wizard of Wor* and *Asterix* have been shown. The moment the DQN agent starts overestimating action values, its score also starts dropping, which means overestimation actually lead to poor policies.

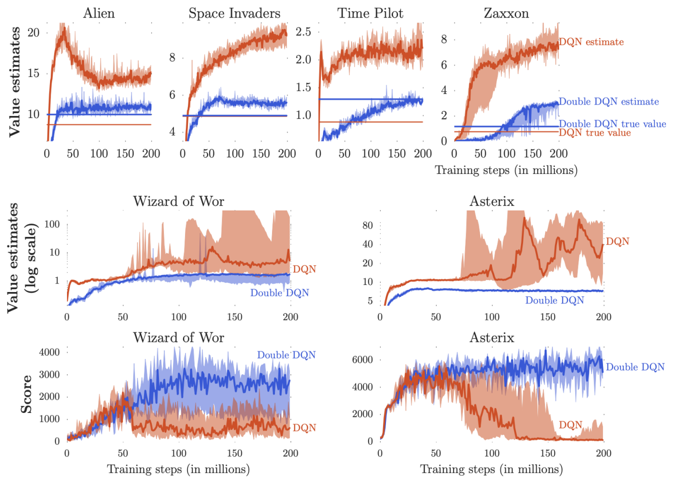

Figure 2: Overestimations in DQN [15]

4 DOUBLE DQN FOR SUPER MARIO BROS

4.1 OpenAI Gym Environment

OpenAI Gym [1] is a python library for developing and comparing reinforcement learning (RL) algorithms. It provides us interfaces to environments of many reinforcement learning problems. RL agents are supposed to act on these environments and observe the feedbacks to compute the next actions.

4.2 Super Mario Bros (NES)

In this project, we are going to train RL agents with Double DQN to play the first level of Super Mario Bros (NES). We will be using the *gym-super-mario-bros* environment [4], which is built on top of the OpenAI gym library. A state of the game is represented by a list of 4 consecutive frames of $240 \times 256 \times 3$ (height $\times$ width $\times$ 3 channels RGB). The size of the action space in this game is 256 which corresponds to 256 possible actions of the character.

Figure 3: An image captured from the Super Mario Bros

The motivation to switch from using a Q-table to building neural networks is because the size of the combinations of observations and actions is massive in this game setting. If we were to use the tabular format for Q-learning, the table would need to store $256 \times 256^{240\times256\times4}$ values, which is not feasible. Therefore, using deep neural networks to approximate the Q function would be clearly a better option since they provide flexible function approximation. The goal here is to map each state to its action values, and the optimal policy is derived from these values.

4.3 Pre-Processing

The input are raw pixels streamed from the Mario Bros game while the agent is playing. We have performed several pre-processing steps to make the training faster and increase the quality of the agents:

(1) The joy-pad is always set to RIGHT to reduce the size of the action space
(2) Resize the frames to 84x84 gray-scale images
(3) Normalize pixel values to [0,1]
(4) Use the same action for 4 consecutive steps
(5) Return the max of 2 most recent observations

5 EXPERIMENTS

5.1 Methodology

In this section we are going to define the methodology of our experimentation. We have selected 4 CNN architectures to be trained with both Double DQN and DQN under the same learning parameters to provide a fair comparison. Firstly, we are going to show how Double DQN performs compared to DQN over a number of training episodes. Then, we are going to test different CNN architectures and compare their average final rewards after 5000 episodes so that we can select the best architecture. At the end, we are going to tune the parameters of the best architecture with the objective of increasing the reward.

5.2 Parameters

Besides the CNN parameters, Deep Reinforcement Learning also introduces some parameters which may determine the success of the learning process. These parameters, shown in Table 1, remained the same during the first 2 experiments among all the CNN architectures and they have been tuned for the last experiment.

Parameter	Definition	Default
gamma	Discount factor	0.9
memory size	Buffer of experience	10000
batch size	Sample of experience to train	32
eps max	Initial exploration rate	1
eps min	Min exploration rate	0.02
eps decay	Decay exploration rate	0.99
lr	Learning rate	0.00025
copy model	Frequency to update networks	5000

Table 1: Parameters of the experiments

5.3 CNN Architectures

Figure 4 shows the different CNN architectures used for the experimentation. The objective is to analyze what architectures work better for this learning environment based on the concepts provided by [5] . The Classic CNN (a) contains 3 convolution layers and 1 linear hidden layer with RELU activation both. The Deeper CNN (b) adds 2 more convolutional layers to (a). The Wider CNN (c) doubles the number of filters in the convolutional layers of (a). Lastly, the MaxPooling CNN (d) adds max pooling after the RELU activation of each convolutional layers in (a).

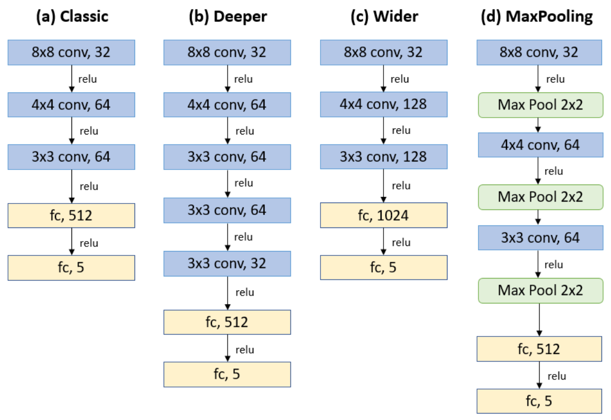

Figure 4: CNN Architectures in our experiments

5.4 Results

Figure 5 shows the evolution of the average rewards over training episodes . Double DQN outperforms the results of DQN for all the architectures trained under the same parameters. Double DQN also provides stability and an increasing trend of reward gain when increasing the number of episodes. On the other hand, DQN tends to decrease the rewards with more training thereby deteriorating the quality of the policy as a consequence of the overestimation as proved by [15]

The next experiment aims to select the best architecture for Super Mario Bros video game. Figure 6 shows the final average reward after running 50 experiments at the end of the training for each architecture. The Classic Double DQN architecture possess the highest average reward accompanied by a high standard

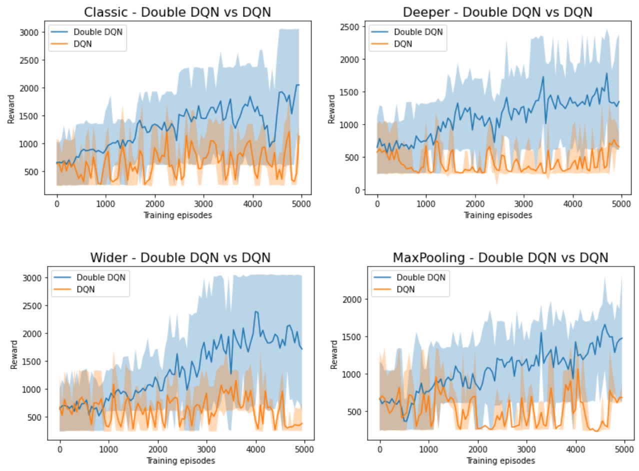

Figure 5: Double DQN vs DQN in different architectures. Evolution of average rewards over training episodes. The shaded area represents the 10% and 90% quantiles. The blue line (for Double DQN) and the red line (for DQN) are calculated each 50 episodes during the training

deviation (Reward: 2284 ± 929). Other architectures such as the Double DQN Wider (Reward: 1497 ± 435) and Double DQN Deeper (Reward: 1178 ± 380) achieved good results with greater stability. Based on the parameters of the experiments, we have selected the Classic Double DQN architecture as the architecture for the hyper parameter tuning section of the experiments.

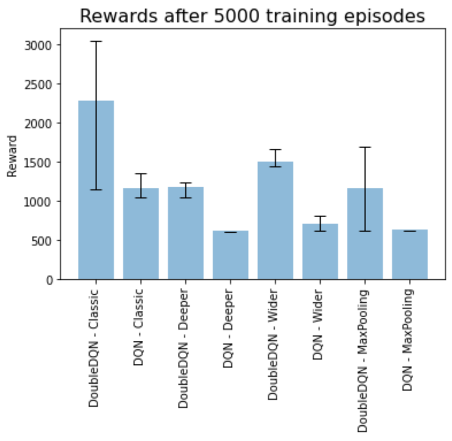

Figure 6: Performance of all architectures. Final rewards after 5000 training episodes. The confidence interval represents the 10% and 90% quantiles after running 50 experiments at the end of the training.

5.5 Hyperparameter tuning

The last part of the experiments involves tuning the hyper parameters of the learning process with the selected architecture. François-Lavet et al [2] highlights the importance of tuning the gamma parameter in order to reduce the number of learning steps. Kim et al [6] also suggests that a slow update of the copy model parameter can slow down the learning due to a delayed update. Training parameters such as the learning rate and the batch size can also impact in the final result.

As shown in Table 2, the experiments with low batch size tend to have better average rewards. In fact, it was said by [8] that training with small samples from all the memory buffer removes correlations between the sequences of frames which it is important when dealing with sequence of frames such as video games. Other important parameter is the frequency of copying parameters between the networks. A faster update of the network tends to speed up the selection/evaluation step in the Double DQN agent. Moreover, the learning rate has a great influence in the quality of the results; low learning rates slow down the training. The winning agent (Reward: 2810 ± 490) was trained with the following parameters: gamma=0.9, batch size=32, lr=0.00025 and copy model=1000 which significantly outperforms the initial Classic DoubleDQN result (Reward: 2284 ± 929) and provides more stability.

gamma	batch size	lr	copy model	Avg reward ± std
0.9	32	0.00025	5000	2284 ± 929
0.9	32	0.00025	1000	2810 ± 490
0.9	32	0.0001	1000	2468 ± 785
0.9	128	0.0001	5000	879 ± 592
0.99	32	0.0001	5000	1720 ± 444
0.99	128	0.0001	1000	1861 ± 793
0.99	128	0.0001	5000	1911 ± 462
0.99	128	0.00025	1000	2425 ± 556

Table 2: Hyperparameter tuning of Double DQN Classic architecture

6 CONCLUSION

This work highlights the importance of using Neural Networks to train agents in real-world environments which require complex value functions. During the experimentation, we have shown how the overestimations of DQN diminish the result of the learning process for different CNN architectures. We have also demonstrated that the implementation of agents with Double DQN provides a higher performance in stability and quality of the learnt policy thereby leading to better rewards. In fact, Double DQN resulted in an increasing reward with more training episodes while DQN tends to be unstable and poor with more episodes when applied to the Super Mario Bros video game. By comparing different architectures, we have shown that wider and deeper architectures slow down the learning, but they have a more stable learning which can be beneficial if they are trained with more episodes and GPU. At the end, we have shown the importance of tuning the hyper parameters of the agent. The role of training with small samples from all the memory buffer removes correlations between the sequences of frames leading to a better learning. Likewise, updating the target network parameters with the online network ones more frequently speeds up the learning task and increases the final reward significantly.

REFERENCES

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:cs.LG/1606.01540

[2] Vincent François-Lavet, Raphael Fonteneau, and Damien Ernst. 2016. How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies. arXiv:cs.LG/1512.02011

[3] Audrunas Gruslys, Will Dabney, Mohammad Gheshlaghi Azar, Bilal Piot, Marc Bellemare, and Remi Munos. 2018. The Reactor: A fast and sample-efficient Actor-Critic agent for Reinforcement Learning. arXiv:cs.AI/1704.04651

[4] Christian Kauten. 2018. Super Mario Bros for OpenAI Gym. GitHub. https://github.com/Kautenja/gym-super-mario-bros

[5] A. Khan, A. Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. 2020. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review* (2020), 1 – 62.

[6] Seungchan Kim, Kavosh Asadi, Michael Littman, and George Konidaris. 2019. DeepMellow: Removing the Need for a Target Network in Deep Q-Learning. 2733–2739. https://doi.org/10.24963/ijcai.2019/379

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 1928–1937.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (02 2015), 529–33. https://doi.org/10.1038/nature14236

[9] V. Mnih, K. Kavukcuoglu, D. Silver, Andrei A. Rusu, J. Veness, Marc G. Bellemare, A. Graves, Martin A. Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, S. Petersen, C. Beattie, A. Sadik, Ioannis Antonoglou, H. King, D. Kumaran, Daan Wierstra, S. Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (2015), 529–533.

[10] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized Experience Replay. (11 2015).

[11] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA.

[12] Fuxiao Tan and Pengfei Yan. 2017. Deep Reinforcement Learning: From Q-Learning to Deep Q-Learning. 475–483. https://doi.org/10.1007/978-3-319-70093-9_50

[13] Sebastian Thrun and Anton Schwartz. 1993. Issues in Using Function Approximation for Reinforcement Learning. In *Proceedings of the 1993 Connectionist Models Summer School*, Michael Mozer, Paul Smolensky, David Touretzky, Jeffrey Elman, and Andreas Weigend (Eds.). Lawrence Erlbaum, 255–263. http://www.ri.cmu.edu/pub_files/pub1/thrun_sebastian_1993_1/thrun_sebastian_1993_1.pdf

[14] Hado Van Hasselt. 2010. Double Q-learning. 2613–2621.

[15] Hado van Hasselt, Arthur Guez, and David Silver. 2015. Deep Reinforcement Learning with Double Q-learning. arXiv:cs.LG/1509.06461

[16] Ziyu Wang, Nando Freitas, and Marc Lanctot. 2015. Dueling Network Architectures for Deep Reinforcement Learning. (11 2015).

[17] Christopher Watkins. 1989. Learning From Delayed Rewards. (01 1989).

[18] Menghao Wu, Yanbin Gao, A. Jung, Qiang Zhang, and S. Du. 2019. The Actor-Dueling-Critic Method for Reinforcement Learning. *Sensors (Basel, Switzerland)* 19 (2019).

[19] Jianwei Zhai, Quan Liu, Zongzhang Zhang, Shan Zhong, Haijun Zhu, Peng Zhang, and Cijia Sun. 2016. Deep Q-Learning with Prioritized Sampling. In *Neural Information Processing*, Akira Hirose, Seiichi Ozawa, Kenji Doya, Kazushi Ikeda, Minho Lee, and Derong Liu (Eds.). Springer International Publishing, Cham, 13–22.