Deep Quality-Value (DQV) Learning

Matthia Sabatelli
Montefiore Institute
Université de Liège, Belgium
m.sabatelli@uliege.be

Gilles Louppe
Montefiore Institute
Université de Liège, Belgium
g.louppe@uliege.be

Pierre Geurts
Montefiore Institute
Université de Liège, Belgium
p.geurts@uliege.be

Marco A. Wiering
Bernoulli Institute for Mathematics, Computer Science
and Artificial Intelligence
University of Groningen, The Netherlands
m.a.wiering@rug.nl

Abstract

We introduce a novel Deep Reinforcement Learning (DRL) algorithm called Deep Quality-Value (DQV) Learning. DQV uses temporal-difference learning to train a Value neural network and uses this network for training a second Quality-value network that learns to estimate state-action values. We first test DQV's update rules with Multilayer Perceptrons as function approximators on two classic RL problems, and then extend DQV with the use of Deep Convolutional Neural Networks, 'Experience Replay' and 'Target Neural Networks' for tackling four games of the Atari Arcade Learning environment. Our results show that DQV learns significantly faster and better than Deep Q-Learning and Double Deep Q-Learning, suggesting that our algorithm can potentially be a better performing synchronous temporal difference algorithm than what is currently present in DRL.

1 Introduction

In Reinforcement Learning (RL), Temporal-Difference (TD) learning has become a design choice which is shared among the most successful algorithms that are present in the field [2]. Whether it is used in a Tabular-RL setting [11, 33], or in combination with a function approximator [28, 30], TD methods aim to learn a Value function, V, by directly bootstrapping their own experiences at different time-steps t. This is done with respect to a discount factor, γ, and a reward, r, which allow the computation of the TD-errors, $r_t + \gamma V(s_{t+1}) - V(s)$. Since the TD-errors can be computed directly while interacting with the environment, TD learning algorithms turn out to be a much faster and better performing alternative, when compared to other RL approaches such as Monte Carlo (MC) learning. In fact, the latter methods require the RL episodes to end before being able to estimate how to predict future rewards.

The recent successful marriage between the most popular RL algorithms, and the use of Deep Neural Networks [16], has highlighted the power of TD-learning methods. Despite having to deal with the instabilities of a non-linear function approximator, such algorithms have led to remarkable results, which have made Deep Reinforcement Learning (DRL) one of the most popular fields within Machine Learning. Temporal-difference learning algorithms have been shown to be very efficient, both when applied on learning an action-value function [10, 21, 22, 31, 36], and when used in combination with policy gradient methods [6, 17, 20].

In this work we exploit the powerful capabilities of TD learning by extending a Tabular-RL algorithm, called QV(λ) learning [34], so that its update rules can be successfully used in combination with

Deep Artificial Neural Networks. Furthermore, we take advantage of the most popular strategies which are known to ensure stability when training DRL algorithms, in order to construct our novel DRL algorithm called Deep Quality-Value learning.

The structure of this paper is as follows: in Section 2 we present the most important theoretical concepts of the field of RL, which serve as preliminary knowledge for our novel algorithm presented in Section 3. We then present the methodological details about the experimental setup of this work in Section 4 and the results that have been obtained in Section 5. An analysis of the performances of our algorithm, together with a discussion about the relevance of our work, concludes the paper in Section 6.

2   Preliminaries

Reinforcement Learning (RL) is the branch of Machine Learning in which artificial agents have to learn an optimal behavior while interacting with a particular environment [27]. An RL problem can be formalized as a Markov Decision Process (MDP) consisting of a set of possible states, $\mathscr{S}$, and a set of possible actions $\mathscr{A}$. By choosing an action $a \in \mathscr{A}$, the agent performs a state transition from state s_t at time t to s_{t+1} that is defined by a transition probability distribution $p(s_{t+1}|s_t, a_t)$. Associated to this transition probability there is an immediate reward function, $\Re(s_t, a_t, s_{t+1})$, that specifies the reward r_t of being in state s_{t+1} based on the action a_t that the agent has taken in state s_t. The agent chooses which actions to perform based on its policy, which maps every state to a particular action $\pi : s \to a$. If a policy is followed, it is possible to compute for each state its Value (V):

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t = s\right],$$ (1)

which corresponds to the expected cumulative reward that an agent will gain while being in state s, and by following policy π. The discount factor γ is set between $[0, 1]$, when its value is close to 0 the agent will only consider the rewards that are obtained in an immediate future, while if its value is close to 1, the agent will also aim to maximize the rewards obtained in a more distant future. Training an RL agent consists in finding an optimal policy, π^*, which maximizes the expected future reward from all possible states such that

$$\pi^*(s) = \underset{\pi}{argmax}\, V^{\pi}(s).$$ (2)

In addition to the V function, there is a second function, $Q^{\pi}(s, a)$, which indicates how good or bad it is to select a particular action a in state s while following policy π. The goal is to find a policy that maximizes Q, which is known to correspond to satisfying the Bellman equation:

$$Q^{\pi}(s_t, a_t) = \sum_{s_{t+1} \in \mathscr{S}} p(s_{t+1}|s_t, a_t)\left(\Re(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1} \in \mathscr{A}} Q^{\pi}(s_{t+1}, a_{t+1})\right).$$ (3)

Once Q is learned, the optimal policy can be easily found by choosing the action that maximizes Q in each state.

Whether the goal is to learn the V or the Q function, Tabular based RL aims to formulate the different RL update rules similar to Dynamic Programming algorithms, which makes it possible to exploit the Markov properties of such rules. However, it is well known that this approach is computationally very demanding and is only suitable for problems which have a limited state-action space. In order to deal with more complicated RL problems non parametric function approximators can be used [5]. While several types of regression algorithms can be used to do so, such as Support Vector Machines [19] or Tree-based approaches [7], a large popularity has been gained with the use of Deep Artificial Neural Networks. The combination of RL and Deep Learning has recently made it possible to master RL problems of the most various domains, ranging from videogames [8, 15, 20, 21, 32], to boardgames [14, 23, 24] and complex robotics control problems [1, 9, 17].

3 Deep Quality-Value (DQV) Learning

We now introduce our novel DRL algorithm called Deep Quality-Value (DQV) learning. We first present the online RL algorithm QV(λ) together with its update rules, and then extend these update rules to objective functions that can be used to train Artificial Neural Networks (ANNs) for solving complex RL problems.

3.1 QV(λ) Learning

QV(λ) has been presented in [34] and essentially consists of a value function RL algorithm that keeps track of both the Q function and the V function. Its main idea consists of learning the V function through the use of temporal difference TD(λ) learning [25], and using these estimates to learn the Q function with an update rule similar to one step Q-Learning [33]. The main benefit of this approach is that the V function might converge faster than the Q function, since it is not directly dependent on the actions that are taken by the RL agent. After a transition,$\langle s_t, a_t, r_t, s_{t+1} \rangle$, QV($\lambda$) uses the TD($\lambda$) learning rule [25] to update the V function for all states:

$$V(s) := V(s) + \alpha \big[r_t + \gamma V(s_{t+1}) - V(s_t) \big] e_t(s), \tag{4}$$

where α stands for the learning rate and γ is again the discount factor. It is worth noting how the V function is updated according to a similar learning rule as standard Q-Learning [33] to update the Q function:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \big[r_t + \gamma \max_{a_{t+1} \in \mathscr{A}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \big]. \tag{5}$$

Besides this TD update, QV(λ) also makes use of eligibility traces, defined as $e_t(s)$, that are necessary to keep track if a particular state has occurred before a certain time-step or not. These are updated for all states as follows:

$$e_t(s) = \gamma \lambda e_{t-1}(s) + \eta_t(s), \tag{6}$$

where $\eta_t(s)$ is an indicator function that returns a value of 1 whether a particular state occurred at time t and 0 otherwise. Before updating the V function, QV(λ) updates the Q function first, and does this via the following update rule:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \big[r_t + \gamma V(s_{t+1}) - Q(s_t, a_t) \big]. \tag{7}$$

In [34] it is shown that QV(λ) outperforms different offline and online RL algorithms in Sutton's Dyna maze environment. However, this algorithm has so far only been used with tabular representations and Shallow Neural Networks [35] and never in combination with Deep Artificial Neural Networks. Thus we now present its extension: Deep QV Learning. We show how to transform the update rules 4 and 7 as objective functions to train ANNs, and how to make the training procedure stable with the use of 'Experience Replay' [18] and 'Target Networks' [21].

3.2 DQV Learning

Since the aim is to approximate both the V and the Q function, we train two ANNs with two distinct objective functions. We thus define the parametrized V neural network as Φ and the Q neural network as θ. In order to build the two objective functions it is possible to simply express QV-Learning's update rules in Mean Squared Error terms similarly to how DQN addresses the Q-Learning update rule 5:

$$L_\theta = \mathbb{E} \big[(r_t + \gamma \max_{a_{t+1} \in \mathscr{A}} Q(s_{t+1}, a_{t+1}, \theta) - Q(s_t, a_t, \theta))^2 \big]. \tag{8}$$

Hence we obtain the following objective function when aiming to train the V function:

$$L_\Phi = \mathbb{E}\big[(r_t + \gamma V(s_{t+1}, \Phi) - V(s_t, \Phi))^2\big], \tag{9}$$

while the following one can be used to train the Q function:

$$L_\theta = \mathbb{E}\big[(r_t + \gamma V(s_{t+1}, \Phi) - Q(s_t, a_t, \theta))^2\big]. \tag{10}$$

By expressing the update rules 4 and 7 as such it becomes possible to minimize the just presented objective functions by gradient descent.

QV-Learning is technically an online reinforcement learning algorithm since it assumes that its update rules get computed each time an agent has performed an action and has observed its relative reward. However, when it comes to more complex control problems (like the games of the Arcade Learning Environment, ALE, presented in [21]), training a Deep Convolutional Neural Network (DCNN) in an online setting is computationally not suitable. This would in fact make each experience usable for training exactly one time, and as a result, a very large set of experiences will have to be collected to properly train the ANNs. To overcome this issue it is possible to make use of 'Experience Replay' [17, 21, 36], a technique which allows to learn from past episodes multiple times and that has proven to be extremely beneficial when tackling RL problems with DCNNs.

Experience Replay essentially consists of a memory buffer, D, of size N, in which experiences are stored in the form $\langle s_t, a_t, r_t, s_{t+1} \rangle$. Once this memory buffer is filled with a large set of these quadruples, $\mathcal{N}$, it becomes possible to randomly sample batches of such experiences for training the model. By doing so the RL agent can learn from previous experiences multiple times and does not have to rely only on the current $\langle s_t, a_t, r_t, s_{t+1} \rangle$ quadruple for training. In our experiments we use an Experience Replay buffer that stores the most recent 400,000 frames that come from the ALE, we uniformly sample batches of 32 experiences $\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)$, and use them for optimizing the loss functions 9 and 10 each time the agent has performed an action. Training starts as soon as 50,000 frames have been collected in the buffer.

Besides making it possible to exploit past experiences multiple times, training from Experience Replay is also known for improving the stability of the training procedure. In fact, since the trajectories that are used for optimizing the networks get randomly sampled from the memory buffer, this makes the samples used for training much less correlated to each other, which yields more robust training. A second idea that serves the same purpose when it comes to TD algorithms has been proposed in [21], and is known as the 'Target Neural Network'.

Target Neural Network: it consists of a separate ANN that is specifically designed for estimating the targets (y_t) that are necessary for computing the TD errors. This ANN has the exact same structure as the one which is getting optimized, but its weights do not change each time RL experiences are sampled from the Experience Replay buffer to train the value function. On the contrary, the weights of the target network are temporally frozen, and only periodically get updated with the parameters of the main network. In the classic DQN scenario the target network is parametrized as θ^-, while the Deep Q-Network uses θ, leading to a slight modification of the loss function presented in Equation 8:

$$L_\theta = \mathbb{E}\big[(r_t + \gamma \max_{a_{t+1} \in \mathscr{A}} Q(s_{t+1}, a_{t+1}, \theta^-) - Q(s_t, a_t, \theta))^2\big]. \tag{11}$$

Since DQV also computes TD errors we include a specifically designed Value-Target-Network to our algorithm, which, similarly to what happens in DQN, slightly modifies the original loss functions presented in Equations 9 and 10 to:

$$L_\Phi = \mathbb{E}\big[(r_t + \gamma V(s_{t+1}, \Phi^-) - V(s_t, \Phi))^2\big] \tag{12}$$

and

$$L_\theta = \mathbb{E}\big[(r_t + \gamma V(s_{t+1}, \Phi^-) - Q(s_t, a_t, \theta))^2\big]. \tag{13}$$

4

For our experiments we update the Value-Target-Network Φ^- with the weights of our original Value Network Φ every 10,000 actions as defined by the hyperparameter c.

We have now defined all elements of our novel DRL algorithm which is summarized by the pseudocode presented in Algorithm 1.

Algorithm 1 DQV Learning

Require: Experience Replay Stack D with size N
Require: Q network with parameters θ
Require: V networks with parameters Φ and Φ^-
Require: total_a = 0
Require: total_e = 0
Require: c = 10000
Require: $\mathcal{N} = 50000$
 1: **for** $e \in$ EPISODES **do**
 2:     **while** $e \neg$ over **do**
 3:         observe s_t
 4:         select $a_t \in \mathcal{A}$ for s_t with policy π
 5:         get r_t and s_{t+1}
 6:         store e as $\langle s_t, a_t, r_t, s_{t+1} \rangle$ in D
 7:         total_e += 1
 8:         **if** total_e $\geq \mathcal{N}$ **then**
 9:             sample minibatch of 32 e from D
10:             **if** $s_{t+1} \in e$ is over **then**
11:                 $y_t := r_t$
12:             **else**
13:                 $y_t := r_t + \gamma V(s_{t+1}, \Phi^-)$
14:             **end if**
15:             $\theta := \arg\min_{\theta} \mathbb{E}[(y_t - Q(s_t, a_t, \theta))^2]$
16:             $\Phi := \arg\min_{\Phi} \mathbb{E}[(y_t - V(s_t, \Phi))^2]$
17:             total_a += 1
18:             **if** total_a = c **then**
19:                 $\Phi^- := \Phi$
20:                 total_a := 0
21:             **end if**
22:         **end if**
23:     **end while**
24: **end for**

---

4 Experimental Setup

For our experiments we use two groups of environments that are both provided by the Open-AI Gym package [4]. The first group is intended to explore DQV's performance on a set of RL tasks that do not rely on computationally expensive GPU support and make it possible to quickly fine tune all the necessary hyperparameters of the algorithm. It consists of two environments that simulate two classic control problems that are well known in RL literature: *Acrobot* [26] and *Cartpole* [3]. The second group of environments is more challenging and consists of several Atari 2600 games. In particular we investigate DQV's performances on the games *Pong*, *Enduro*, *Boxing* and *Ice-Hockey*. A visualization of such environments can be seen in Figure 1. In all the experiments we compare the performance of DQV with two other well known self implemented TD DRL algorithms: DQN [21] and DDQN [31].

We use a two hidden layer Multilayer Perceptron (MLP) activated by a ReLU non linearity ($f(x) = max(0,x)$) when we test the DRL algorithms on the first two environments. While a three hidden layer DCNN, followed by a fully connected layer as originally presented in [21], when tackling the Atari 2600 games. For the latter set of environments we use the `Deterministic-v4` versions of the games, which, as proposed in [21], use 'Frame-Skipping', a design choice which lets the

agent select an action every 4th frame instead of every single one. Furthermore, we use the standard Atari-preprocessing scheme in order to resize each frame to an 84×84 gray-scaled matrix.

The Adam optimizer [13] has been used for optimizing the MLPs while the RMSprop optimizer [29] has been used for the DCNNs. As DQV's exploration strategy we use the well known epsilon-greedy approach with an initial ε value set to 0.5 which gets linearly decreased to 0.1. The discount factor γ is set to 0.99. Lastly, since the scale of the rewards changes from game to game, we 'clip' all the rewards to the $[-1, 1]$ range.

Figure 1: A visualization of the six RL environments that have been used in our experiments as provided by the Open-AI Gym package [4]. From left to right: *Acrobot, Cartpole, Pong, Enduro, Boxing, Ice-Hockey*.

5   Results

We first evaluate the performance that DQV has obtained on the *Acrobot* and *Cartpole* environments where we have used MLPs as function approximators. Given the relative simplicity of these problems we did not integrate DQV with the Value-Target-Network. In fact, we have only optimized the two different MLPs according to the objective functions 9 and 10 presented in Section 3.2. Regarding the Experience Replay buffer we did not include it in the *Acrobot* environment while it is used in the *Cartpole* one. However its size is far smaller than the one which has been presented in Algorithm 1, in fact, since the problem tackled is much simpler, we only store 200 experiences which get randomly sampled with a batch size of 16.

We then present the results obtained on the Atari 2600 games, where instead of MLPs, we have used DCNNs and have integrated the algorithm with a larger Experience Replay buffer and the Value-Target-Network. The algorithm used is thus the one presented in Algorithm 1. We report on the x axis of all figures the amount of RL episodes that have been used to train the different algorithms, and on the y axis the cumulative reward that has been obtained by the agents in each episode. All results report the average cumulative reward which has been obtained over 5 different simulations with 5 different random seeds. Please note that we smoothed the reward function with a 'Savgol Filter' to improve the interpretability of the plots.

The results obtained on *Acrobot* and *Cartpole* show how promising the update rules 9 and 10 can be when used in combination with an MLP. In fact as shown in Figure 2 it is possible to see that DQV is able to significantly outperform DQN and DDQN in both environments. It is however worth noting that the problems which have been tackled in this set of experiments are not particularly complex. Hence, in order to concretely establish the performance of our novel algorithm, the Atari 2600 games serve as an ideal testbed. The results presented in Figure 3 show how DQV is again able to systematically outperform DQN and DDQN in all the experiments we have performed. When DQV is tested on the games *Pong* and *Boxing* it can be seen that it learns significantly faster when compared to the other two algorithms. This is particularly true for the game *Pong* which gets solved in less than 400 episodes, making DQV more than two times faster than DQN and DDQN. Similar results have been obtained on the *Boxing* environment, where DQV approaches the maximum possible cumulative reward in the game (≈ 80) almost 300 episodes before the other two algorithms. On the *Enduro* and *Ice-Hockey* games we can see that DQV does not only learn faster, but also obtains much higher cumulative rewards during the game. These rewards are almost twice as high in the *Enduro* environment, while the results also show that DQV is the only DRL algorithm which is able to optimize the policy over 800 episodes when it is tested on the *Ice-Hockey* game. The latter results are particularly interesting since they correspond to the first and only case so far, in which we have observed that DQN and DDQN are not able to improve the policy during the time that is required by DQV to do so.
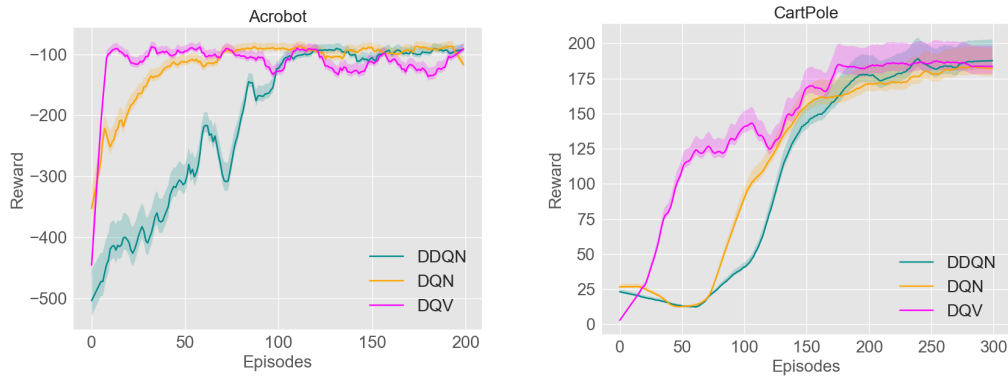
6

Figure 2: The results that have been obtained when using DQV in combination with an MLP on two classic RL problems (Acrobot and Cartpole). DQV learns significantly faster when compared with DQN and DDQN.
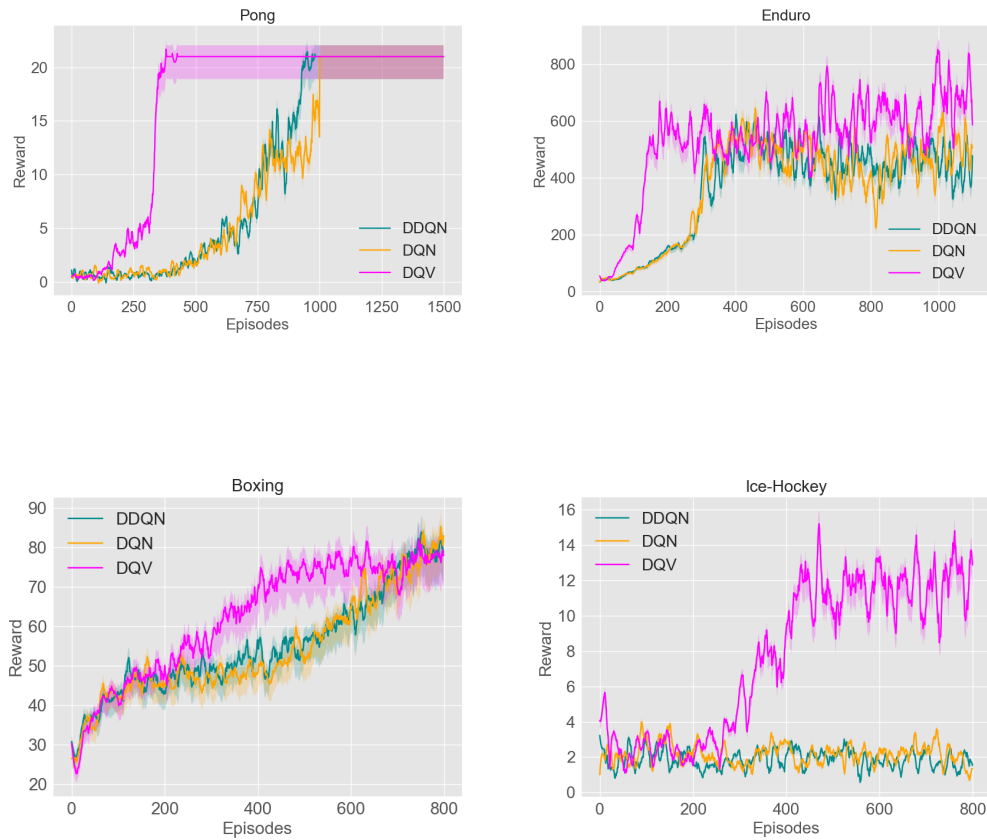

Figure 3: The results which have been obtained when using DQV in combination with Deep Convolutional Neural Networks, Experience Replay and Target Neural Networks on four Atari games. It can be seen that DQV learns significantly faster on the games *Pong* and *Boxing* while it also yields better results on the games *Enduro* and *Ice-Hockey*.

6 Discussion and Conclusion

Besides introducing a novel DRL algorithm, we believe that the methodologies and results proposed in this paper contribute to the field of DRL in several important ways. We showed how it is possible to build upon existing Tabular-RL work, in order to successfully extend the set of DRL algorithms which are currently present in the field. So far, the results obtained by DQV-Learning are very promising and definitely suggest that our algorithm is a better alternative if compared to the well known DQN and DDQN algorithms. Furthermore, DQV also reaffirms the benefits of TD-Learning. More specifically, it is particularly interesting to see how effective the use of the V neural networks is, when learning an action-value function with the Q network. When QV(λ) has been initially introduced, one of the main intuitions behind the algorithm relied on the fact that the V function could converge faster than the Q function, which can make the algorithm learn faster. The results obtained in this paper support this idea, with DQV being the fastest algorithm in all the experiments that we have performed.

Moreover, we also noticed how important it is to integrate a DRL algorithm with a specifically designed Target-Network that is explicitly built for estimating the target values while learning. It is worth noting however, that unlike DQN and DDQN, this Target-Network is required for the V function and not the Q function, because the V function is used to update both value functions. Lastly, we also remark the importance of including an Experience-Replay buffer in the algorithm when tackling complex RL problems as the ones provided by the Arcade Learning Environment. We aim to research more in detail the role of the Target-Network. In fact, there is to the best of our knowledge no real formal theory that motivates the need for this additional neural architecture. Nevertheless, it is a design choice which is well known to be useful in the field. With DQV there is a new algorithm that makes use of it, and which could help for gaining a better understanding of this additional neural architecture.

The main strength of DQV certainly relies on the speed that our algorithm obtains for learning. However, we are aware that more sophisticated approaches such as the Asynchronous-Advantage-Actor-Critic (A3C) algorithm have already been proposed in the field for dealing with long RL training times. As future work we want to work on a multi-threaded version of DQV, and investigate if, in case its update rules are used for training multiple agents in parallel, our algorithm could perform better and faster than A3C. Furthermore, by estimating both the V function and the Q function, DQV-Learning allows for a novel method to estimate the Advantage function, and we want to study if this can help algorithms such as A3C to perform better. Finally, we want to take into account the several contributions that have been proposed in the literature for making DQN more data efficient and faster [12]. We will perform a similar ablation study on DQV, and investigate whether such improvements will be as successful when applied to our algorithm and result in state-of-the-art performance on many different Atari 2600 games.

References

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pages 1–8, 2007.

[2] Artemij Amiranashvili, Alexey Dosovitskiy, Vladlen Koltun, and Thomas Brox. TD or not TD: Analyzing the role of temporal differencing in deep reinforcement learning. *arXiv preprint arXiv:1806.01175*, 2018.

[3] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2010.

[6] Dotan D Castro, Dmitry Volkinshtein, and Ron Meir. Temporal difference based actor critic learning-convergence and neural implementation. In *Advances in neural information processing systems*, pages 385–392, 2009.

[7] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

[8] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*, 2017.

[9] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

[10] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[11] Hado van Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[12] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Matthew Lai. Giraffe: Using deep reinforcement learning to play chess. *arXiv preprint arXiv:1509.01549*, 2015.

[15] Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. In *AAAI*, pages 2140–2146, 2017.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[17] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[18] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[19] Mario Martin. On-line support vector machine regression. In *European Conference on Machine Learning*, pages 282–294. Springer, 2002.

[20] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[22] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep RL for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

[23] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[24] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[25] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[26] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

[27] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[28] Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000. ACM, 2009.

[29] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[30] John N Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

[31] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

[32] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[33] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[34] Marco A Wiering. QV (lambda)-learning: A new on-policy reinforcement learning algorithm. In *Proceedings of the 7th European Workshop on Reinforcement Learning*, pages 17–18, 2005.

[35] Marco A Wiering and Hado van Hasselt. The QV family compared to other reinforcement learning algorithms. In *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*, pages 101–108. IEEE, 2009.

[36] Dongbin Zhao, Haitao Wang, Kun Shao, and Yuanheng Zhu. Deep reinforcement learning with experience replay based on Sarsa. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–6. IEEE, 2016.