

# The Deep Quality-Value Family of Deep Reinforcement Learning Algorithms

1<sup>st</sup> Matthia Sabatelli  
Montefiore Institute  
Liège, Belgium  
m.sabatelli@uliege.be

2<sup>nd</sup> Gilles Louppe  
Montefiore Institute  
Liège, Belgium  
g.louppe@uliege.be

3<sup>rd</sup> Pierre Geurts  
Montefiore Institute  
Liège, Belgium  
p.geurts@uliege.be

4<sup>th</sup> Marco A. Wiering  
University of Groningen  
Groningen, The Netherlands  
m.a.wiering@rug.nl

**Abstract**—We present a novel approach for learning an approximation of the optimal state-action value function ( $Q$ ) in model-free Deep Reinforcement Learning (DRL). We propose to learn this approximation while simultaneously learning an approximation of the state-value function ( $V$ ). We introduce two new DRL algorithms, called DQV-Learning and DQV-Max Learning, which follow this specific learning dynamic. In short, both algorithms use two neural networks for separately learning the  $V$  function and the  $Q$  function. We validate the effectiveness of this training scheme by thoroughly comparing our algorithms to DRL methods which only learn an approximation of the  $Q$  function, namely DQN and DDQN. Our results show that DQV and DQV-Max present several important benefits: they converge significantly faster, can achieve super-human performance on DRL testbeds on which DQN and DDQN failed to do so, and suffer less from the overestimation bias of the  $Q$  function.

**Index Terms**—model-free deep reinforcement learning, temporal-difference learning, DQV, DQV-Max-Learning

## I. INTRODUCTION

In value-based Reinforcement Learning (RL) the aim is to construct algorithms which learn *value functions* that are either able to estimate how good or bad it is for an agent to be in a particular state, or how good it is for an agent to perform a particular action in a given state. Such functions are respectively denoted as the state-value function  $V(s)$ , and the state-action value function  $Q(s, a)$  [1]. In Deep Reinforcement Learning (DRL) the aim is to approximate these value functions with e.g. deep convolutional neural networks [2], which can serve as universal function approximators and powerful feature extractors. Classic model-free RL algorithms like Q-Learning [3], Double Q-Learning [4] and SARSA [5] have all led to the development of a “deep” version of themselves in which the original RL update rules are expressed as objective functions that can be minimized by gradient descent [6]–[8]. Despite their successful applications [9], the aforementioned algorithms only aim at approximating the  $Q$  function, while completely ignoring the  $V$  function. This approach, however, is prone to issues that go back to standard RL literature. The DQN algorithm [6] is known to overestimate the values of the  $Q$  function [7] and requires an additional target network to not diverge (which role is not yet fully understood [10]). These overestimations can partially be corrected by the DDQN [7] algorithm, which, despite yielding stability improvements, does not always prevent its  $Q$  networks from diverging [11]

and sometimes even underestimating the  $Q$  function. Furthermore, DRL algorithms are also extremely slow to train. In this work, we introduce a new family of DRL algorithms, which simultaneously learn the  $V$  function and the  $Q$  function for faster, more robust and better model-free Deep Reinforcement Learning.

## II. PRELIMINARIES

We formally define the RL setting as a Markov Decision Process (MDP) where the main components are a finite set of states  $\mathcal{S} = \{s^1, s^2, \dots, s^n\}$ , a finite set of actions  $\mathcal{A}$  and a time-counter variable  $t$ . In each state  $s_t \in \mathcal{S}$ , the RL agent can perform an action  $a_t \in \mathcal{A}(s_t)$  and transit to the next state as defined by a transition probability distribution  $p(s_{t+1}|s_t, a_t)$ . When moving from  $s_t$  to a successor state  $s_{t+1}$  the agent receives a reward signal  $r_t$  coming from the reward function  $\mathfrak{R}(s_t, a_t, s_{t+1})$ . The actions of the agent are selected based on its policy  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  that maps each state to a particular action. For every state  $s \in \mathcal{S}$ , under policy  $\pi$  its *value function*  $V^\pi$  is defined as:

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

which denotes the expected cumulative discounted reward that the agent will get when starting in state  $s$  and by following policy  $\pi$  thereafter. Similarly, we can also define the *state-action value function*  $Q$  for denoting the value of taking action  $a$  in state  $s$  based on policy  $\pi$  as:

$$Q^\pi(s, a) = \mathbb{E} \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi \right]. \quad (2)$$

Both functions are computed with respect to the discount factor  $\gamma \in [0, 1]$  which controls the trade-off between immediate and long term rewards. The goal of an RL agent is to find a policy  $\pi^*$  that realizes the optimal expected return:

$$V^*(s) = \max_{\pi} V^\pi(s), \text{ for all } s \in \mathcal{S} \quad (3)$$

and the optimal  $Q$  value function:

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a) \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \quad (4)$$

It is well-known that optimal value functions satisfy the Bellman optimality equation as given by

$$V^*(s_t) = \max_a \sum_{s_{t+1}} p(s_{t+1}|s_t, a) \left[ \mathfrak{R}(s_t, a, s_{t+1}) + \gamma V^*(s_{t+1}) \right] \quad (5)$$

for the state-value function, and by

$$Q^*(s_t, a_t) = \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \left[ \mathfrak{R}(s_t, a_t, s_{t+1}) + \gamma \max_a Q^*(s_{t+1}, a) \right], \quad (6)$$

for the state-action value function. Both functions can either be learned via Monte Carlo methods or by Temporal-Difference (TD) learning [12], with the latter approach being so far the most popular choice among model-free RL algorithms [3]–[5].

### III. RELATED WORK

The contribution which has established the potential of DRL can certainly be identified with the Deep-Q-Network (DQN), the first algorithm which uses a convolutional neural network for successfully learning an approximation of the  $Q$  function from high dimensional inputs [6]. This approximation is learned by reshaping the popular Q-Learning algorithm [3] to an objective function which can be minimized by gradient descent. The original Q-Learning algorithm learns the state-action value function as follows

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \left[ r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t) \right] \quad (7)$$

where  $\alpha$  corresponds to the learning rate. The DQN algorithm adapts this update rule to a differentiable loss function which can be used for training a neural network that is parametrized by  $\theta$ . This objective function comes in the following form:

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)} \left[ \left( r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta) \right)^2 \right]. \quad (8)$$

Within this loss there are two components which ensure stable training. The first one is the Experience-Replay memory buffer ( $D$ ), a buffer coming in the form of a queue which stores RL experiences  $\langle s_t, a_t, r_t, s_{t+1} \rangle$ . When it comes to the popular Atari Arcade Learning (ALE) [13] benchmark, the DQN algorithm uniformly samples mini-batches of 32 experiences for minimizing Eq. 8, a procedure which starts as soon as at least 50.000 experiences are stored within the queue. Furthermore, there is a second component which ensures stable training denoted as the target-network. DQN learns an approximation of the  $Q$  function via TD-Learning, meaning that the approximated  $Q$ -function is regressed towards TD-targets which are computed by the approximated  $Q$  function itself. The TD-target, defined as  $y_t^{DQN}$ , is expressed as follows:

$$y_t^{DQN} = r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-), \quad (9)$$

and is computed by the target network  $\theta^-$  instead from the online Q-network  $\theta$ . The online network, and its target counterpart, have the exact same structure, with the main difference being that the parameters of the latter do not get optimized each time a mini-batch of experiences is sampled from the memory buffer. On the contrary its weights are temporally frozen and are only periodically updated with the  $\theta$  weights (as defined by an appropriate hyperparameter). Given a training iteration  $i$ , differentiating the objective function of Eq. 8 with respect to  $\theta$  gives the following gradient:

$$\nabla_{\theta_i} y_t^{DQN}(\theta_i) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)} \left[ \left( r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta_{i-1}^-) - Q(s_t, a_t; \theta_i) \right) \nabla_{\theta_i} Q(s_t, a_t; \theta_i) \right]. \quad (10)$$

Despite yielding super-human performance on most games coming from the ALE, DQN has shown to be suffering from the same issues which characterize the Q-Learning algorithm [4]. Among these issues we mention the overestimation bias of the Q-function [4]. In short, DQN is prone to learn overestimated Q-values because the same values are used both for selecting an action ( $\max_{a \in \mathcal{A}}$ ) and for evaluating it ( $Q(s_{t+1}, a; \theta^-)$ ). As originally presented in [7] this becomes clearer when re-writing Eq. 9 as:

$$y_t^{DQN} = r_t + \gamma Q(s_{t+1}, \arg \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta); \theta^-). \quad (11)$$

As a result, DQN tends to approximate the expected maximum value of a state, instead of its maximum expected value. To solve this problem the DDQN algorithm untangles the selection of an action from its evaluation by taking advantage of the target network  $\theta^-$ . DDQN's target is the same as DQN's with the main difference being that the selection of an action, given by the online Q-network  $\theta$ , and the evaluation of the resulting policy, given by  $\theta^-$ , can get unbiased by symmetrically updating the two sets of weights ( $\theta$  and  $\theta^-$ ). This can be achieved by regularly switching their roles during training.

Several extensions of DQN and DDQN have been proposed over the years, to make these algorithms learn faster and more data-efficient. We refer the reader to [9] for a more in-depth review of these contributions. Within this paper, we are only interested in synchronous DRL algorithms which learn an approximation of a value function. This is achieved by following the same experimental setups that have been used for DQN and DDQN, and that will be reviewed in Sec. V-A of this paper. Therefore, in this contribution, we will aim at comparing our novel DRL algorithms to DQN and DDQN only, while leaving their potential integration within more sophisticated DRL techniques as future work.

### IV. DQV AND DQV-MAX LEARNING

Just as much as DQN and DDQN are based on two tabular RL algorithms, so are the main contributions presented in this

work. More specifically we extend two RL algorithms which were introduced in [14] and [15] to the use of deep neural networks that serve as function approximators. Training these algorithms robustly is done by taking advantage of some of the techniques which have been reviewed in the previous section.

### A. DQV-Learning

Our first contribution is the Deep Quality-Value (DQV) Learning algorithm, a novel DRL algorithm which aims at jointly approximating the  $V$  function alongside the  $Q$  function in an *on-policy* learning setting. This algorithm is based on the QV( $\lambda$ ) algorithm [14], a tabular RL algorithm which learns the  $V$  function via the simplest form of TD-Learning [12], and uses the estimates that are learned by this value function to update the  $Q$  function in a Q-Learning resembling way. We take inspiration from this specific learning dynamic and aim at learning an approximation of both the  $V$  function, and the  $Q$  function, with two neural networks that are respectively parametrized by  $\Phi$  and  $\theta$ . The objective function which is used by DQV for learning the state-value function is given by the following equation:

$$L(\Phi) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)} \left[ \left( r_t + \gamma V(s_{t+1}; \Phi^-) - V(s_t; \Phi) \right)^2 \right], \quad (12)$$

while the following loss is minimized for learning the  $Q$  function:

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)} \left[ \left( r_t + \gamma V(s_{t+1}; \Phi^-) - Q(s_t, a_t; \theta) \right)^2 \right], \quad (13)$$

where  $D$  is again the Experience-Replay memory buffer, used for uniformly sampling batches of RL trajectories  $\langle s_t, a_t, r_t, s_{t+1} \rangle$ , and  $\Phi^-$  is the target-network used for the construction of the TD-errors. Please note that the role of this target network is different from its role within the DQN algorithm. In DQV this network corresponds to a copy of the network which approximates the state-value function and not the state-action value function. It is also worth noting that both networks learn from the same TD-target which comes in the following form:

$$y_t^{DQV} = r_t + \gamma V(s_{t+1}; \Phi^-). \quad (14)$$

The gradient with respect to both loss functions can be easily expressed similarly as done in Eq. 10 for the DQN algorithm.

### B. DQV-Max Learning

Our second contribution is the Deep Quality-Value-Max algorithm, a novel DRL algorithm which reshapes some of the ideas that characterize DQV. Similarly as done for DQV, we still aim at jointly learning an approximation of the  $V$  function and the  $Q$  function, but in this case, the goal is to do this with an *off-policy* learning scheme. To construct this

algorithm we take inspiration from the QV-Max RL algorithm introduced in [15]. The key component of QV-Max is the use of the  $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$  operator, which makes RL algorithms learn *off-policy*. We use this operator when approximating the  $V$  function and for computing TD-errors which correspond to the ones that are also used by the DQN algorithm. However, within DQV-Max, these TD-errors are used by the state-value network and not by the state-action value network. This results in the following loss which is used for learning the  $V$  function:

$$L(\Phi) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)} \left[ \left( r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - V(s_t; \Phi) \right)^2 \right]. \quad (15)$$

In this case the target network  $\theta^-$  corresponds to the same target network that is used by DQN. The TD-error  $r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-)$  is however only used for learning the  $V$  function. When it comes to the  $Q$  function we use the same update rule that is presented in Eq. 13 with the only difference being that in this case no  $\Phi^-$  target network is used. Despite requiring the computation of two different targets for learning, we noticed that DQV-Max did not benefit from using two distinct target networks, therefore its loss function for approximating the  $Q$  function is simply:

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)} \left[ \left( r_t + \gamma V(s_{t+1}; \Phi) - Q(s_t, a_t; \theta) \right)^2 \right]. \quad (16)$$

The pseudocode of both DQV and DQV-Max is presented at the end of this paper in Algorithm 1. The pseudocode is an adaptation of a standard DRL training loop which corresponds to what is usually presented within the literature [6]. We just make explicit use of the hyperparameters `total_a` and `c` which ensure that enough actions have been performed by the agent before updating the weights of the target network. We also ensure via the hyperparameter `total_e`, that enough episodes are stored within the memory buffer (which has capacity  $\mathcal{N}$ ) before starting to optimize the neural networks.

## V. EMPIRICAL RESULTS

We now present a set of empirical results that help us characterize the performance of DQV and DQV-Max. We start by evaluating the performance of our algorithms on the common `Atari-2600` benchmark [13] in Sec. V-A. We then investigate the time that is required by both algorithms to converge to a successful policy in Sec. V-B. We finally study the quality of the learned value functions in Sec. V-C.

### A. Global Evaluation

We evaluate the performance of DQV and DQV-Max on a subset of 15 games coming from the popular `Atari-2600` benchmark [13]. Our newly introduced algorithms are compared against DQN and DDQN. To keep all the comparisons as fair as possible we follow the same experimental setup

and evaluation protocol which was used in [6] and [7]. The only difference between DQV and DQV-Max, and DQN and DDQN is the exploration schedule which is used. Differently from the latter two algorithms, which use an epsilon-greedy strategy which has an  $\epsilon$  starting value of 1.0, DQV and DQV-Max’s exploration policy starts with an initial  $\epsilon$  value of 0.5. All other hyperparameters, ranging from the size of the Experience-Replay memory buffer to the architectures of the neural networks, are kept the same among all algorithms. We refer the reader to the original DQN paper [6] for an in-depth overview of all these hyperparameters. The performance of the algorithms is tested based on the popular `no-op` action evaluation regime. At the end of the training, the learned policies are tested over a series of episodes for a total amount of 5 minutes of emulator time. All testing episodes start by executing a set of partially random actions to test the level of generalization of the learned policies. We present our results in Table I where the best performing algorithm is reported in a green cell while the second-best performing algorithm is reported in a yellow cell. As is common within the DRL literature, the table also reports the scores which would be obtained by an expert human player and by a random policy. When the scores over games are equivalent, we report in the green and yellow cells the fastest and second fastest algorithm with respect to its convergence time. We can start by observing that DQV and DQV-Max successfully master all the environments on which they have been tested, with the only exception being the *Montezuma’s Revenge* game. It is well-known that this game requires more sophisticated exploration strategies than the epsilon-greedy one [16], and was also not mastered by DQN and DDQN when these algorithms were introduced. We can also observe that there is no algorithm which performs best on all the tested environments even though, as highlighted by the green and yellow cells, the algorithms of the DQV-family seem to generally perform better than DQN and DDQN, with DQV-Max being the overall best performing algorithm in our set of experiments. When either DQV or DQV-Max are not the best performing algorithm (see for example the *Boxing* and *CrazyClimber* environments), we can still observe that our algorithms managed to converge to a policy which is not significantly worse than the one learned by DQN and DDQN. There is however one exception being the *RoadRunner* environment. In fact, in this game, DDQN significantly outperforms DQV and DQV-Max. It is also worth noting the results on the *BankHeist* and *Enduro* environments. Both DQN and DDQN failed to achieve super-human performance on these games, while DQV and DQV-Max successfully managed to obtain a significantly higher score than the one obtained by a professional human player.

### B. Convergence Time

While DRL algorithms have certainly obtained impressive results on the *Atari-2600* benchmark, it is also true that the amount of training time which is required by these algorithms can be very long. Over the years, several techniques, ranging from Prioritized Experience Replay (PER) [17] to the Rainbow

extensions [18], have been proposed to reduce the training time of DRL algorithms. It is therefore natural to investigate whether jointly approximating the  $V$  function alongside the  $Q$  function can lead to significant benefits in this behalf. Unlike the  $Q$  function, the state-value function is not dependent on the set of possible actions that the agent may take, and therefore requires fewer parameters to converge. Since DQV and DQV-Max use the estimates of the  $V$  network to train the  $Q$  function, it is possible that the  $Q$  function could directly benefit from these estimates and as a result converge faster than when regressed towards itself (as happens in DQN).

We use two self-implemented versions of DQN and DDQN for comparing the convergence time that is required during training by all the tested algorithms on three increasingly complex *Atari* games: *Boxing*, *Pong* and *Enduro*. Our results, reported in Fig. 1, show that DQV and DQV-Max converge significantly faster than DQN and DDQN, and highlight the benefits of jointly approximating two value functions instead of one when it comes to the overall convergence time that is required by the algorithms. Even though, as presented in Table I, DQV and DQV-Max do not always significantly outperform DQN and DDQN in terms of the final cumulative reward which is obtained, it is worth noting that these algorithms require significantly less training episodes to converge on all tested games. This benefit makes our two novel algorithms faster alternatives within model-free DRL.

### C. Quality of the Learned Value Functions

It is well-known that the combination of RL algorithms with function approximators can yield DRL algorithms that diverge. The popular Q-Learning algorithm is known to result in unstable learning both if linear [19] and non-linear functions are used when approximating the  $Q$  function [11]. This divergence is caused by the interplay of three elements that are known as the ‘*Deadly Triad*’ of DRL [1]. The elements of this triad are:

- *a function approximator*: which is used for learning an approximation of a value function that could not be learned in the tabular RL setting due to a too large state-action space.
- *bootstrapping*: when the algorithms use a future estimated value for learning the same kind of estimate.
- *off-policy learning*: when a future estimated value is different from the one which would be computed by the policy the agent is following.

Recent work [11] has shown that the ‘*Deadly Triad*’ is responsible for enhancing one of the most popular biases that characterize the Q-Learning algorithm: the overestimation bias of the  $Q$  function [4]. It is therefore natural to study how DQV and DQV-Max relate to the ‘*Deadly Triad*’ of DRL, and to investigate up to what extent these algorithms suffer from the overestimation bias of the  $Q$  function. To do this we monitor the estimates that are given by the network that is responsible for approximating the  $Q$  function. More specifically, at training time, we compute the averaged  $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$  over a set ( $n$ ) of full evaluation episodes

TABLE I: The results obtained by DQV and DQV-Max on a subset of 15 Atari games. We can see that our newly introduced algorithms have a comparable, and often even better performance than DQN and DDQN. As highlighted by the green cells the overall best performing algorithm in our set of experiments is DQV-Max while the second-best performing algorithm is DQV (as reported by the yellow cells). Specific attention should be given to the games BankHeist and Enduro where DQV and DQV-Max are the only algorithms which can master the game with a final super-human performance.

Environment	Random	Human	DQN [6]	DDQN [7]	DQV	DQV-Max
Asteroids	719.10	13156.70	1629.33	930.60	1445.40	1846.08
Bank Heist	14.20	734.40	429.67	728.30	1236.50	1118.28
Boxing	0.10	4.30	71.83	81.70	78.66	80.15
Crazy Climber	10780.50	35410.50	114103.33	101874.00	108600.00	1000131.00
Enduro	0.00	309.60	301.77	319.50	829.33	875.64
Fishing Derby	-91.70	5.50	-0.80	20.30	1.12	20.42
Frostbite	65.20	4334.70	328.33	241.50	271.86	281.36
Gopher	257.60	2321.00	8520.00	8215.40	8230.30	7940.00
Ice Hockey	-11.20	0.90	-1.60	-2.40	-1.88	-1.12
James Bond	29.00	406.70	576.67	438.00	372.41	440.80
Montezuma's Revenge	0.00	4366.70	0.00	0.00	0.00	0.00
Ms.Pacman	307.30	15693.40	2311.00	3210.00	3590.00	3390.00
Pong	-20.70	9.30	18.90	21.00	21.00	21.00
Road Runner	11.50	7845.00	18256.67	48377.00	39290.00	20700.00
Zaxxon	32.50	9173.30	4976.67	10182.00	10950.00	8487.00

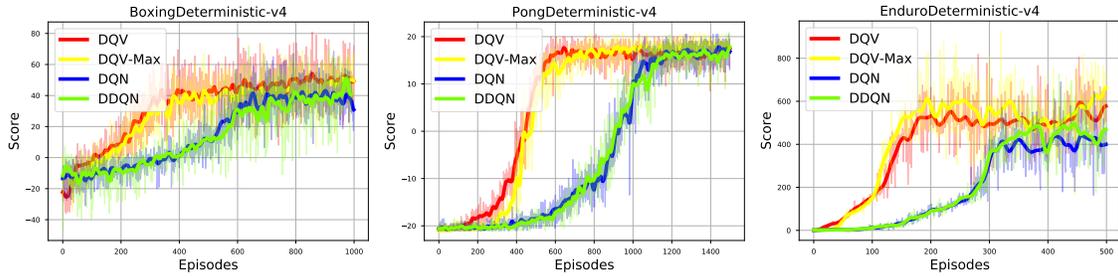


Fig. 1: Learning curves obtained during training on three different Atari games by DQV and DQV-Max, and DQN and DDQN. We can observe that on these games both DQV and DQV-Max converge significantly faster than DQN and DDQN and that they obtain higher cumulative rewards on the Enduro environment. The shaded areas correspond to  $\pm 1$  standard deviation obtained over 5 different simulation rounds.

as defined by  $\frac{1}{n} \sum_{t=1}^n \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta)$ . As suggested in [7] these estimates can then be compared to the averaged discounted return of all visited states that comes from an agent that has already concluded training. By analyzing whether the  $Q$  values which are estimated while training differ from the ones which should be predicted by the end of it, it is possible to quantitatively characterize the level of divergence of DRL algorithms. We report our results in Fig. 2, where the full lines correspond to the value estimates that come from each algorithm at training time, while the dashed lines correspond to the actual averaged discounted return that is given by an already trained agent.

We can start by observing that the values denoting the averaged discounted return obtained by each algorithm differ among agents. This is especially the case when it comes to the Enduro environment, and is a result which is in line with what has been presented in Table I: DQV and DQV-Max lead to better final policies than DQN and DDQN. Furthermore, when we compare these baseline values to the value estimates that are obtained during training, we can observe that the ones obtained by the DQN algorithm significantly diverge from the ones which should be predicted by the end of training.

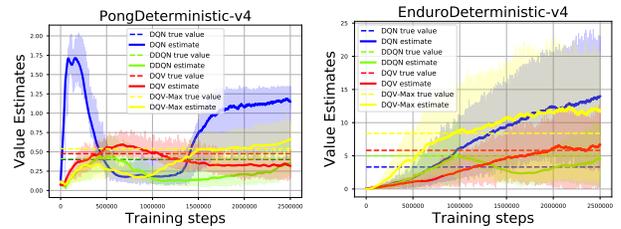


Fig. 2: The results showing that DQV and DQV-Max suffer less from the overestimation bias of the  $Q$  function. We can observe that at training time the  $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$  estimates (denoted by the full lines) do not diverge too much from the return that is obtained by an already trained agent (dashed line). Please note that on these two games DDQN is able to correct for the overestimation bias of the  $Q$  function and that DQV-Max seems to still suffer (even though less when compared to DQN) from this issue.

This behavior is known to be caused by the overestimation bias of the  $Q$  function which can be corrected by the DDQN algorithm. By analyzing the value estimates of DQV and

DQV-Max we can observe that both algorithms produce value estimates which are more similar to the ones computed by DDQN than to the ones given by DQN. This is especially the case for DQV, in fact its value estimates nicely correspond to the averaged discounted return baseline, both on the `Pong` environment and on the `Enduro` environment. The estimates coming from DQV-Max, however, seem to diverge more when compared to DQV and DDQN’s ones. This is clearer on the `Enduro` environment, where the algorithm does show some divergence. However, we can also observe that this divergence is less strong when compared to DQN’s one. The value estimates of the latter algorithm keep growing over time, while DQV-Max’s ones get bounded while training progresses. This results in smaller estimated  $Q$  values. We believe that there are mainly two reasons why our algorithms suffer less from the overestimation bias of the  $Q$  function. When it comes to DQV, we believe that this algorithm suffers less from this bias since it is an *on-policy* learning algorithm. Such algorithms are trained on exploration actions with lower  $Q$  values. Because of its *on-policy* learning scheme, DQV also does not present one element of the ‘*Deadly Triad*’, which might help reducing divergence. When it comes to DQV-Max, we believe that the reason why this algorithm does not diverge as much as DQN can be found in the way it approximates the  $Q$  function. One key component of the ‘*Deadly Triad*’, is that divergence occurs if the  $Q$  function is learned by regressing towards itself. As given by Eq. 16 we can see that this does not hold for DQV-Max, since the  $Q$  function bootstraps with respect to estimates that come from the  $V$  network. We believe that this specific learning dynamic, which also holds for the DQV algorithm, makes our algorithms less prone to estimate large  $Q$  values.

## VI. ADDITIONAL STUDIES

As introduced in Sec. IV DQV and DQV-Max use two separate neural networks for approximating the  $Q$  function and the  $V$  function. To verify whether two different architectures are needed for making both algorithms perform well, we have experimented with a series of variants of the DQV-Learning algorithm. The aim of these experiments is to investigate whether the performance of DQV gets harmed when reducing its capacity. The studied DQV’s extensions are the following:

- i *Hard-DQV*: a version of DQV which uses one single common neural network for approximating both the  $Q$  and the  $V$  functions. An additional output node, needed for estimating the value of a state, is added next to the output nodes which estimate the different  $Q$  values. The parameters of this algorithm are therefore ‘hardly-shared’ among the agent, and provide the benefit of halving the total amount of trainable parameters of DQV. The different outputs of the network get then alternatively optimized according to Eq. 12 and 13.
- ii *Dueling-DQV*: a slightly more complicated version of *Hard-DQV* which adds one specific hidden layer before the output nodes that estimate the  $Q$  and  $V$  functions. In this case, the outputs of the neural network which learn one of the two value functions, partly benefit from

some specific weights that are not shared within the neural network. This approach is similar to the one used by the ‘*Dueling-Architecture*’ [17], therefore the name *Dueling-DQV*. While it is well established that three convolutional layers are needed [6], [7] for learning the  $Q$  function, the same might not be true when it comes to learning the  $V$  function. We thus report experiments with three different versions of *Dueling-DQV*: *Dueling-1st*, *Dueling-2nd*, and *Dueling-3rd*. The difference between these methods is simply the location of the hidden layer which precedes the output that learns the  $V$  function. It can be positioned after the first convolutional layer, the second or the third one. Training this architecture is done as for *Hard-DQV*.

- iii *Tiny-DQV*: the neural architectures used by DQV and DQV-Max that approximate the  $V$  function and the  $Q$  function follow the one which was initially introduced by the DQN algorithm [6]. This corresponds to a three-hidden layer convolutional neural network which is followed by a fully connected layer of 512 hidden units. The first convolutional layer has 32 channels while the last two layers have 64 channels. In *Tiny-DQV* we reduce the number of trainable parameters of DQV by reducing the number of channels at each convolution operation. *Tiny-DQV* only uses 8 channels after the first convolutional layer and 16 at the second and third convolutional layers. Furthermore, the size of the final fully connected layer is reduced to only 128 hidden units. The choice of this architecture is motivated by the work presented in [11] which studies the role of the capacity of the DDQN algorithm. Unlike the *Hard-DQV* and *Dueling-DQV* extensions, the parameters of *Tiny-DQV* are not shared at all among the networks that are responsible for approximating the  $V$  function and the  $Q$  function.

The results obtained by these alternative versions of DQV are presented in Fig. 3 where we report the learning curves obtained by the tested algorithms on six different `Atari` games. Each DQV extension is directly compared to the original DQV algorithm. We can observe that all the extensions of DQV, which aim at reducing the number of trainable parameters of the algorithm, fail in performing as well as the original DQV algorithm. We can observe that *Hard-DQV* does not only yield significantly lower rewards (see the results obtained on `Boxing`) but also presents extremely unstable training (as highlighted by the results obtained on the `Pong` environment). Lower rewards and unstable training also characterize the *Tiny-DQV* algorithm (see results on `BankHeist` and `CrazyClimber`). Overall the most promising extensions of DQV are its *Dueling* counterparts, we have observed in particular that the best performing architecture over most of our experiments was the *Dueling-DQV-3rd* one. As can be seen by the results obtained on the `Pong` environment we can observe that *Dueling-DQV-3rd* has comparable performance to DQV, even though it converges slower. Unfortunately, *Dueling-DQV-3rd* still shows some limitations, in particular when tested on more complicated environments such as `Enduro`, we can

observe that it under-performs DQV with  $\approx 200$  points. It is also worth mentioning that the idea of approximating the  $V$  function before the  $Q$  function explored by *Dueling-DQV-1st* and *Dueling-DQV-2nd* yielded negative results.

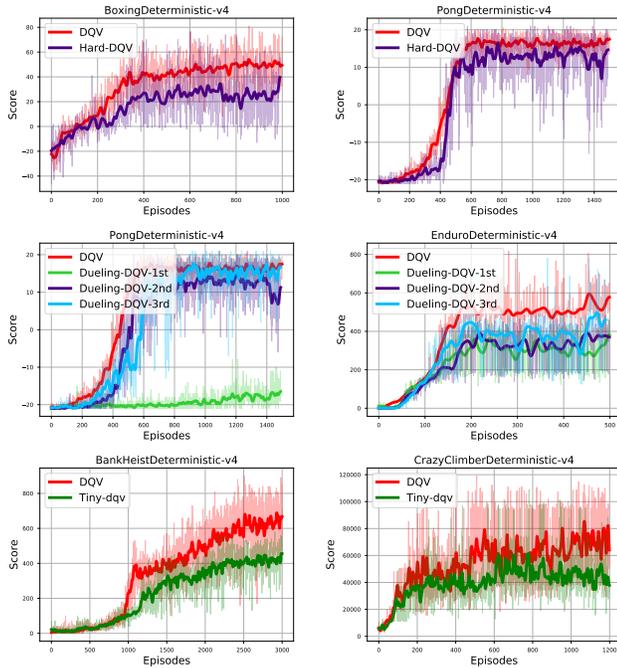


Fig. 3: Learning curves obtained by the alternative versions of the DQV algorithm which explore whether one key component of the DQV algorithm is that of using two separate neural networks with independent parameters. We can observe that this is indeed the case since all the tested extensions fail to match the performance of the original DQV algorithm. These results highlight that the best strategy for approximating two different value functions is with two separately parameterized neural networks.

## VII. DISCUSSION AND CONCLUSION

We have presented two novel model-free DRL algorithms which in addition to learning an approximation of the  $Q$  function also aim at learning an approximation of the  $V$  function. We have compared DQV and DQV-Max Learning to DRL algorithms which only learn an approximation of the  $Q$  function, and showed the benefits which come from jointly approximating two value functions over one. Our newly introduced algorithms learn significantly faster than DQN and DDQN and show that approximating both the  $V$  function and the  $Q$  function can yield significant benefits both in an *on-policy* learning setting as in an *off-policy* learning one. This specific training dynamic allows for a better learned  $Q$  function which makes DQV and DQV-Max less prone to estimate unrealistically large  $Q$  values. All these benefits come however at a price: to successfully learn two value functions, two separate neural networks with enough capacity are required. We identify several directions for further research that focus

on the following points: an integration of the algorithms of the DQV-family with all the extensions which have improved the DQN algorithm over the years; an integration of DQV and DQV-Max within an Actor-Critic framework which will allow us to tackle continuous-control problems, and lastly, a study of how the algorithms of the DQV-family will perform in a Batch-DRL setting.

## VIII. ACKNOWLEDGEMENTS

Matthia Sabatelli kindly acknowledges the financial support of BELSPO, Federal Public Planning Service Science Policy, Belgium, in the context of the BRAIN-be project.

## REFERENCES

- [1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [2] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, p. 436, 2015.
- [3] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
- [4] H. Van Hasselt, “Double Q-learning,” in *Advances in Neural Information Processing Systems*, 2010, pp. 2613–2621.
- [5] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, England, 1994, vol. 37.
- [6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.
- [7] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [8] D. Zhao, H. Wang, K. Shao, and Y. Zhu, “Deep reinforcement learning with experience replay based on SARSA,” in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016, pp. 1–6.
- [9] Y. Li, “Deep reinforcement learning: An overview,” *arXiv preprint arXiv:1701.07274*, 2017.
- [10] J. Achiam, E. Knight, and P. Abbeel, “Towards characterizing divergence in deep Q-learning,” *arXiv preprint arXiv:1903.08894*, 2019.
- [11] H. Van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, “Deep reinforcement learning and the deadly triad,” *arXiv preprint arXiv:1812.02648*, 2018.
- [12] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.
- [13] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.
- [14] M. A. Wiering, “QV (lambda)-learning: A new on-policy reinforcement learning algorithm,” in *Proceedings of the 7th European Workshop on Reinforcement Learning*, 2005, pp. 17–18.
- [15] M. A. Wiering and H. Van Hasselt, “The QV family compared to other reinforcement learning algorithms,” in *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL’09. IEEE Symposium on*. IEEE, 2009, pp. 101–108.
- [16] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin *et al.*, “Noisy networks for exploration,” *arXiv preprint arXiv:1706.10295*, 2017.
- [17] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. Freitas, “Dueling network architectures for deep reinforcement learning,” in *International Conference on Machine Learning*, 2016, pp. 1995–2003.
- [18] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [19] J. N. Tsitsiklis and B. Van Roy, “Analysis of temporal-difference learning with function approximation,” in *Advances in neural information processing systems*, 1997, pp. 1075–1081.

---

**Algorithm 1** DQV and DQV-Max Learning

---

**Require:** Experience Replay Queue  $D$  of maximum size  $N$

**Require:**  $Q$  network with parameters  $\theta$

▷ Network required by DQV

**Require:**  $V$  networks with parameters  $\Phi$  and  $\Phi^-$

▷ Networks required by DQV

**Require:**  $Q$  networks with parameters  $\theta$  and  $\theta^-$

▷ Networks required by DQV-Max

**Require:**  $V$  network with parameters  $\Phi$

▷ Network required by DQV-Max

**Require:** total\_a = 0

**Require:** total\_e = 0

**Require:** c = 10000

**Require:**  $\mathcal{N} = 50000$

1: **while** True **do**

2:   set  $s_t$  as the initial state

3:   **while**  $s_t$  is not terminal **do**

4:     select  $a_t \in \mathcal{A}$  for  $s_t$  with policy  $\pi$  (using the epsilon-greedy strategy)

5:     get  $r_t$  and  $s_{t+1}$

6:     store  $\langle s_t, a_t, r_t, s_{t+1} \rangle$  in  $D$

7:      $s_t := s_{t+1}$

8:     total\_e += 1

9:     **if** total\_e  $\geq \mathcal{N}$  **then**

10:       sample a minibatch  $B = \{\langle s_t^i, a_t^i, r_t^i, s_{t+1}^i \rangle | i = 1, \dots, 32\}$  of size 32 from  $D$

11:       **for** i = 1 to 32 **do**

12:          **if**  $s_{t+1}^i$  is terminal **then**

13:            $y_t^i := r_t^i$

▷ TD-Error for DQV

14:            $v_t^i := r_t^i$

▷ 1st TD-Error for DQV-Max

15:            $q_t^i := r_t^i$

▷ 2nd TD-Error for DQV-Max

16:          **else**

17:            $y_t^i := r_t^i + \gamma V(s_{t+1}^i, \Phi^-)$

▷ TD-Error for DQV

18:            $v_t^i := r_t^i + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}^i, a, \theta^-)$

▷ 1st TD-Error for DQV-Max

19:            $q_t^i := r_t^i + \gamma V(s_{t+1}^i, \Phi)$

▷ 2nd TD-Error for DQV-Max

20:          **end if**

21:       **end for**

22:        $\theta := \arg \min_{\theta} \sum_{i=1}^{32} (y_t^i - Q(s_t^i, a_t^i, \theta))^2$

▷ Train the  $Q$  network for DQV

23:        $\Phi := \arg \min_{\Phi} \sum_{i=1}^{32} (y_t^i - V(s_t^i, \Phi))^2$

▷ Train the  $V$  network for DQV

24:        $\theta := \arg \min_{\theta} \sum_{i=1}^{32} (q_t^i - Q(s_t^i, a_t^i, \theta))^2$

▷ Train the  $Q$  network for DQV-Max

25:        $\Phi := \arg \min_{\Phi} \sum_{i=1}^{32} (v_t^i - V(s_t^i, \Phi))^2$

▷ Train the  $V$  network for DQV-Max

26:       total\_a += 1

27:       **if** total\_a = c **then**

28:           $\Phi^- := \Phi$

▷ Update the target  $V$  network in DQV

29:           $\theta^- := \theta$

▷ Update the target  $Q$  network in DQV-Max

30:       total\_a := 0

31:       **end if**

32:     **end if**

33:   **end while**

34: **end while**

---