Approximating two value functions instead of one: towards characterizing a new family of Deep Reinforcement Learning algorithms

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

This paper makes one step forward towards characterizing a new family of *model-free* Deep Reinforcement Learning (DRL) algorithms. The aim of these algorithms is to jointly learn an approximation of the state-value function (V), alongside an approximation of the state-action value function (Q). Our analysis starts with a thorough study of the Deep Quality-Value Learning (DQV) algorithm, a DRL algorithm which has been shown to outperform popular techniques such as Deep-Q-Learning (DQN) and Double-Deep-Q-Learning (DDQN) [15]. Intending to investigate why DQV's learning dynamics allow this algorithm to perform so well, we formulate a set of research questions which help us characterize a new family of DRL algorithms. Among our results, we present some specific cases in which DQV's performance can get harmed and introduce a novel *off-policy* DRL algorithm, called DQV-Max, which can outperform DQV. We then study the behavior of the V and Q functions that are learned by DQV and DQV-Max and show that both algorithms might perform so well on several DRL test-beds because they are less prone to suffer from the overestimation bias of the Q function.

1   Introduction

Value-based Reinforcement Learning (RL) algorithms aim to 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 $Q(s, a)$ [18]. Both can be formally defined by considering the RL setting as a Markov Decision Process (MDP) where the main components are a finite set of states $\mathcal{S} = \{s^1, s^2, ..., s^n\}$, 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 $\Re(s_t, a_t, s_{t+1})$. 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 π its *value function V* is defined as:

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

which denotes the expected cumulative discounted reward that the agent will get when starting in state s and by following policy π 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 π as:

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

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 π^* that realizes the optimal expected return:

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

and the optimal Q value function:

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

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

$$V^*(s_t) = \max_a \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t)\left[\Re(s_t, a_t, s_{t+1}) + \gamma V^*(s_{t+1})\right] \tag{5}$$

for the state-value function, and by

$$Q^*(s_t, a_t) = \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t)\left[\Re(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})\right], \tag{6}$$

for the state-action value function. Both functions can either be learned via Monte Carlo methods or by Temporal-Difference (TD) learning [17], with the latter approach being so far the most popular choice among model-free RL algorithms [23, 14, 7, 13]. In Deep Reinforcement Learning (DRL) the aim is to approximate these value functions with e.g. deep convolutional neural networks [16, 10]. This has led to the development of a large set of DRL algorithms [8] among which we mention Deep-Q-Learning (DQN) [12] and Double Deep-Q-Learning (DDQN) [21]. Both algorithms have learning an approximation of the state-action-value function as their main goal. This approach however, has recently shown to be outperformed by the Deep Quality-Value-Learning (DQV) algorithm [15], a relatively novel algorithm which simultaneously approximates the state-value function alongside the state-action value function.

1.1 The Deep Quality-Value Learning Algorithm

DQV-Learning [15] is based on the tabular RL algorithm QV(λ) [24], and learns an approximation of the V function and the Q function with two distinct neural networks that are parametrized by Φ and θ respectively. Both neural networks learn via TD-learning and from the same target $(r_t + \gamma V(s_{t+1}; \Phi^-))$, which is computed by the state-value network Φ. The approximation of the Q function is achieved by minimizing the following loss:

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)}\left[\left(r_t + \gamma V(s_{t+1}; \Phi^-) - Q(s_t, a_t; \theta)\right)^2\right], \tag{7}$$

while the following loss is minimized for learning the V function:

$$L(\Phi) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)}\left[\left(r_t + \gamma V(s_{t+1}; \Phi^-) - V(s_t; \Phi)\right)^2\right], \tag{8}$$

where D is the Experience-Replay memory buffer [11], used for uniformly sampling batches of RL trajectories $\langle s_t, a_t, r_t, s_{t+1}\rangle$, and Φ^- is the target-network that is used for the construction of the TD errors. We refer the reader to the supplementary material for a more in-depth explanation of the algorithm which is presented in Algorithm 1.

In [15] it is shown that DQV is able to learn faster and better than DQN and DDQN on six different RL test-beds. It is however not clear why DQV is able to outperform such algorithms so significantly. In what follows we aim at gaining a more in-depth understanding of this algorithm, and build upon this knowledge to investigate the potential benefits that could come from approximating both the V function and the Q function instead of only the latter one.

2   Research Questions and Methods

We now present the different research questions that are tackled in this work. We propose some modifications to the original DQV-Learning algorithm, construct a novel DRL algorithm called Deep-Quality-Value-Max Learning (DQV-Max), and finally investigate the learning dynamics of DQV and DQV-Max by studying their performance under the lens of the *Deadly Triad* phenomenon [18, 20].

2.1   Question 1: does DQV perform better than DQN because it uses twice as many parameters?

Since the state-value function and the state-action value function are approximated with two separate neural networks, DQV uses twice as many parameters as DQN. It is therefore possible that DQV outperforms this algorithm because the capacity of the model is larger. Approximating two value functions instead of one comes at a price that needs to be paid in terms of memory requirements, a problem which we tackle by taking inspiration from the work proposed in [22] on the *Dueling-Architecture*. Aiming to reduce the amount of trainable parameters of DQV, we modify the original version of the algorithm by exploring two different approaches. The first one, simply adds one output node dedicated for learning the V function next to the nodes that estimate the different state-action values (see Fig. 1a). This significantly reduces the parameters of DQV but assumes that the features that are learned for approximating the V function correspond to the ones that are required for learning the Q function. This also assumes that the capacity of the neural network (which structure follows the one proposed in [12]) is large enough for approximating both value functions. Since this might not be the case, the second approach adds one specific hidden layer to the network which precedes the outputs of the different estimated value functions (see Fig. 1b). Since, as introduced in [24], the V function could be easier to learn than the Q function, we also experiment with the location of the hidden layer preceding the V output. We position it after each convolution layer, intending to explore whether the depth of the network influences the quality of the learned V function.
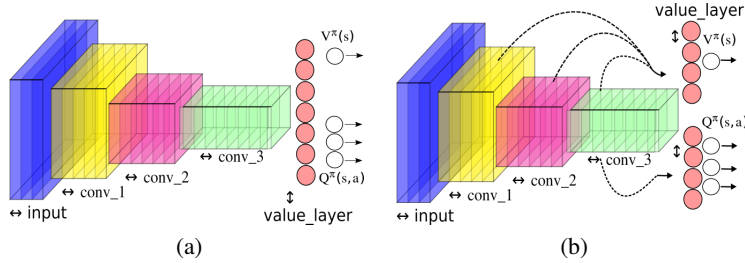

Figure 1: Two representations of convolutional neural networks which jointly approximate the V function and the Q function with the aim of reducing DQV's learning parameters. On the left, an architecture which simply adds one output node to the network next to the output nodes which estimate the Q function. On the right an architecture in which a specific hidden layer precedes the output that is necessary for computing each value function. When it comes to the V function we experiment with different locations of such hidden layer, which is positioned after each possible convolution block.

2.2   Question 2: is using two value function approximators also beneficial in an *off-policy* learning setting?

One important difference between DQV compared to DQN and DDQN is that the last two algorithms learn the Q function with an *off-policy* learning scheme, while DQV is an *on-policy* learning algorithm. In an *off-policy* learning setting, the TD errors are constructed with respect to values which are different from the agent's actual behavior. This has the benefit of allowing the agent to explore many different policies [20] because learning is not restricted by the policy that is being followed. If on the one hand, this can be extremely beneficial when it comes to value iteration algorithms [3, 23], it is also well-known that this particular learning setting yields DRL algorithms that can diverge [18, 20, 6, 1]. It is therefore not clear whether DQV strongly outperforms DQN and DDQN because

it is actually beneficial to approximate both value functions, or simply because being an *on-policy* learning algorithm, DQV is just less prone to diverge.

To answer this question we introduce a novel DRL algorithm called Deep-Quality-Value-Max Learning (DQV-Max). Similarly to DQV this algorithm is also based on a tabular RL-algorithm which was initially introduced in [25]. DQV-Max is constructed in a resembling way as DQV, even though its objectives change. The V function is learned with respect to the greedy strategy $\max_{a \in \mathcal{A}} Q$, therefore making DQV-Max an *off-policy* learning algorithm. DQV's loss that is used for learning the V function presented in Eq. 8 gets modified as follows:

$$L(\Phi) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)} \left[\left(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - V(s_t; \Phi) \right)^2 \right]. \quad (9)$$

The way the Q function is approximated is not modified, with the only difference being that in this case we decided to not use any target network, since one is already used for learning the V function:

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)} \left[\left(r_t + \gamma V(s_{t+1}; \Phi) - Q(s_t, a_t; \theta) \right)^2 \right]. \quad (10)$$

Similarly as done for DQV, we report an in-depth presentation of this algorithm in Algorithm 2, which can be found in the supplementary material of this work.

2.3 Question 3: DQV, DQV-Max, and the *Deadly Triad*, how are they related?

It is known that DRL algorithms are prone to damage the quality of their learned policy when a function approximator that regresses towards itself is used for learning a value function [4]. This phenomenon is formally known to be caused by the *Deadly Triad* [18]. DQV and DQV-Max are at least connected to two out of three components of the *Triad* and could, therefore, help to gain new insights in the study of DRL divergence. We briefly present how each algorithm is related to each element of the *Triad*, which is highlighted in bold, hereafter:

- Both algorithms make obvious use of deep neural networks which serve as **function approximators** for learning a value function.
- DQV and DQV-Max make use of **bootstrapping**: as shown in Eq.7 and Eq.8 DQV bootstraps towards the same target (which is given by the state-value network), whereas DQV-Max bootstraps towards two distinct targets (Eq. 9 and Eq. 10).
- Lastly only DQV-Max presents the final element of the *Triad*: differently from DQV this is the only algorithm which learns in an **off-policy** learning setting.

Based on this information we formulate two hypotheses: the first one is that being an *on-policy* learning algorithm, DQV should be less prone to suffer from divergence than more popular *off-policy* algorithms. The second hypothesis is that even though DQV-Max is an *off-policy* learning algorithm, there is one important difference within it: it learns an approximation of the Q function with respect to TD errors which are given by the V network, therefore not regressing the Q function towards itself anymore (as DQN and DDQN do). Because of this specific learning dynamic, we hypothesize that DQV-Max, could be less prone to diverge. However, if compared to DQV, DQV-Max should still diverge more since it presents all elements of the *Deadly Triad*, while DQV does not.

To quantitatively assess our hypotheses we will investigate whether DQV and DQV-Max suffer from what is known to be one of the main causes of divergence in DRL [20]: the overestimation bias of the Q function [7].

3 Results

We now answer the presented research questions by reporting the results that we have obtained on the well-known `Atari-Arcade-Learning` (ALE) benchmark [2]. We have used the games `Boxing`, `Pong` and `Enduro` since they are increasingly complex in terms of difficulty and can be mastered within a reasonable amount of training time. We refer the reader to the supplementary material for a thorough presentation of the experimental setup that has been used in this work.

3.1 HARD-DQV and Dueling Deep Quality-Value Learning (Question 1)

We start by reporting the results that have been obtained with the neural architecture reported in Fig. 1a. We refer to this algorithm as HARD-DQV since all the parameters of the agent are hardly-shared within the network [5]. We can observe from the blue learning curves presented in Fig. 2 that this approach drastically reduces the performance of DQV on all the tested environments. The results are far from the original DQV algorithm and suggest that a proper approximation of the V function and Q function can only be learned with a neural architecture which has a higher capacity and that reserves some specific parameters that are selective for learning one value function.
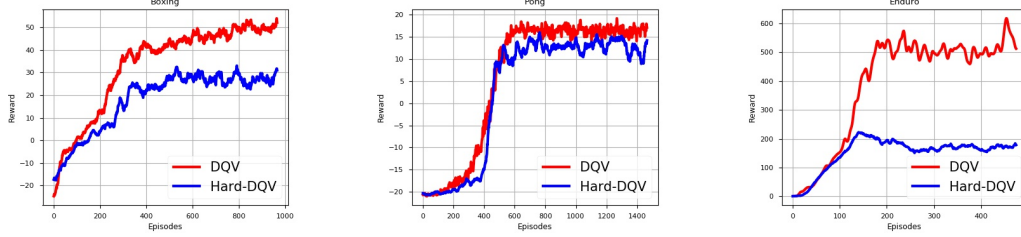

Figure 2: The results obtained with a neural architecture in which the output node dedicated to learning the V function, is placed next to the output nodes which estimate the Q function. It is clear from the results that this approach, albeit drastically reducing the memory requirements of DQV, also significantly harms its performance.

Better results have been obtained with a neural architecture which uses a specific hidden layer before the outputs of the V and Q functions (Fig. 1b). We refer to this algorithm as Dueling-DQV, where Dueling-DQV-1st corresponds to an architecture in which the V function is learned after the first convolution block, Dueling-DQV-2nd after the second block and Dueling-DQV-3rd after the third. As reported in Fig. 3 on the simple `Boxing` environment we can observe that no matter where the state-value layer is positioned, all versions of Dueling-DQV perform as well as the original DQV algorithm. On the more complicated `Pong` environment, however, the performance of Dueling-DQV starts to be strongly affected by the depth of the state-value layer. The only case in which Dueling-DQV performs as well as DQV is when the V function is learned after all three convolutional layers. More interestingly, on the most complicated `Enduro` environment, the performance of DQV cannot be matched by any of the Dueling architectures, in fact, the rewards obtained by Dueling-3rd and DQV differ with ≈ 200 points. These results seem to suggest that the parameters of DQV can be reduced, as long as some specific value-layers are added to the neural architecture. However, this approach presents limitations. As shown by the results obtained on the `Enduro` environment, the best version of DQV remains the one in which two distinct neural networks approximate the V and Q functions.
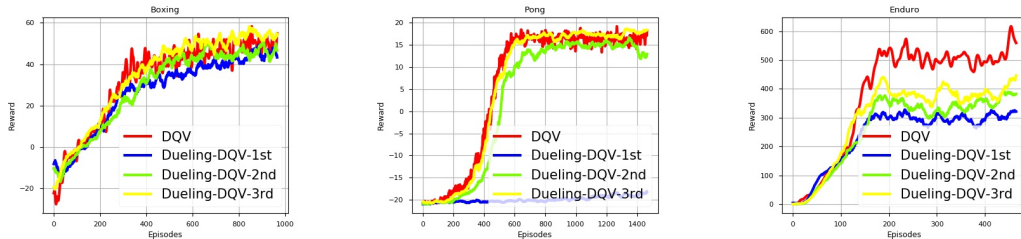

Figure 3: The results obtained by the different Dueling-DQV architectures. On the `Boxing` and `Pong` environments similar results as the ones obtained by DQV can be achieved. However on the `Enduro` environment none of the proposed architectures is able to perform as well as DQV. This suggests that the most beneficial strategy for approximating two value functions is with two distinct neural networks.

5

3.2 The Deep Quality-Value-Max Algorithm (Question 2)

By investigating the performance of the novel DQV-Max algorithm, promising results have been obtained. As we can see in Fig. 4, DQV-Max has a comparable, and sometimes even better performance than the DQV algorithm, therefore strongly outperforming DQN and DDQN. It learns as fast when it comes to the `Boxing` and `Pong` environments and achieves an even higher cumulative reward on the `Enduro` environment, making it the overall best performing algorithm. These results remark the benefits of jointly learning an approximation of the V function and the Q function, and show that this approach is just as beneficial when it comes to an *off-policy* learning setting than it is in an *on-policy* learning one. We can therefore answer affirmatively to the second research question analyzed in this work.
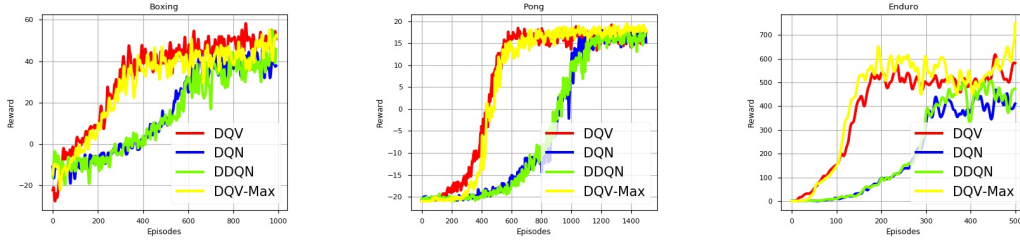

Figure 4: The results obtained by the DQV-Max algorithm on the `Boxing, Pong` and `Enduro` environments. DQV-Max is able to learn as fast and even better than DQV, suggesting that jointly approximating two value functions is also beneficial in an *off-policy* learning setting.

3.3 Reducing the overestimation bias of the Q function (Question 3)

To investigate whether DQV and DQV-Max suffer from the overestimation bias of the Q function we have performed the following experiment. As proposed in [21] we monitor the Q function of the algorithms at training time by computing the averaged $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ over a set (n) of full evaluation episodes as defined by $\frac{1}{n} \sum_{t=1}^{n} \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta)$. We then compare these estimates with the averaged discounted return of all visited states which is given by the same agent that has already concluded training. This provides a reliable baseline for measuring whether the estimated Q values diverge from the ones which should actually be predicted. We report these results in Fig. 5a and 5b, where for each plot the dashed line corresponds to the actual averaged discounted return of the visited states, while the full lines correspond to the value estimates that are computed by each algorithm.

Our results show that DQV and DQV-Max seem to suffer less from the overestimation bias of the Q function since both algorithms learn more stable and accurate value estimates. This allows us to answer both hypotheses introduced in Sec. 2.3 affirmatively: jointly approximating the V function and the Q function can prevent DRL from diverging, since this learning dynamic allows the algorithms to estimate more realistic and not increasingly large Q values. However, this becomes harder to achieve once the algorithm learns *off-policy*. By analyzing the plot presented in Fig. 5b, we can observe that the value estimates of DQV-Max are still higher from the ones which should be predicted by the end of training. However, differently, from the ones of DQN, they get bounded over time, therefore resulting in a less strong divergence. It is also worth noting how in our experiments the DDQN algorithm fully corrects the overestimation bias of DQN as expected.

One could argue that DQV and DQV-Max might not be overestimating the Q function because they are overestimating the V function instead. In order to verify this, we have investigated whether the estimates of the state-value network are higher than the ones coming from the state-action value network. Similarly to the previous experiment we use the models that are obtained at the end of training and randomly sample a set of states from the ALE which are then used in order to compute $V(s)$ and $\max Q(s, a)$. We then investigate whether $V(s)$ is higher than the respective $\max Q(s, a)$ estimate. As can be seen in Fig. 6 this is almost never the case, which empirically shows that both DQV and DQV-Max do not overestimate the V function instead of the Q function.
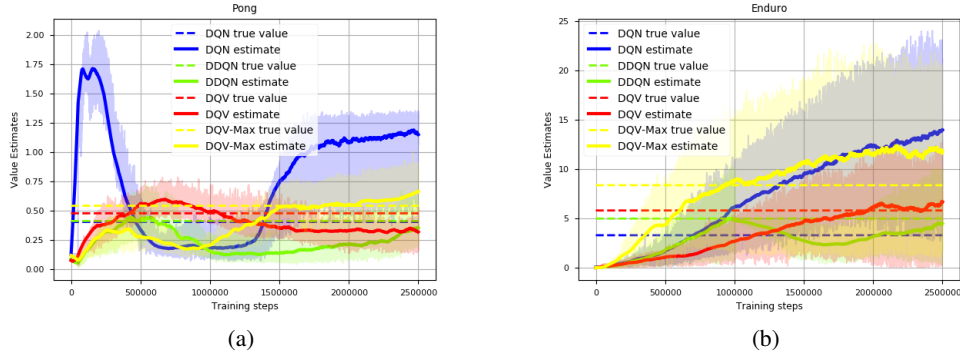
Figure 5: A representation of the value estimates that are obtained by each algorithm at training time (denoted by the full lines), with respect to the actual discounted return obtained by an already trained agent (dashed lines). We can observe that on `Pong` DQN's estimates rapidly grow before getting closer to the baseline discounted value, while DQV and DQV-Max do not show this behavior. Their value estimates are more similar to the ones of the DDQN algorithm. This is especially the case for DQV. On the `Enduro` environment, DQN's estimates do not stop increasing over time and keep moving away from the actual values of each state while DQV's value estimates remain bounded and do not diverge as much. DQV-Max diverges more than DQV but still not as much as DQN. DQN's behavior corresponds to the one observed in [21] and can be linked to the overestimation bias of the algorithm, (which is fully corrected by DDQN). This suggests that DQV and DQV-Max might perform so well on the ALE environment because they are less prone to overestimate the Q function. Furthermore, it is worth noting the different baseline values which denote the averaged discounted return estimates when it comes to the `Enduro` environment: the lines representing the true values of the final policy are very different among algorithms, indicating that DQV and DQV-Max do not only learn more accurate value estimates but also lead to better final policies.
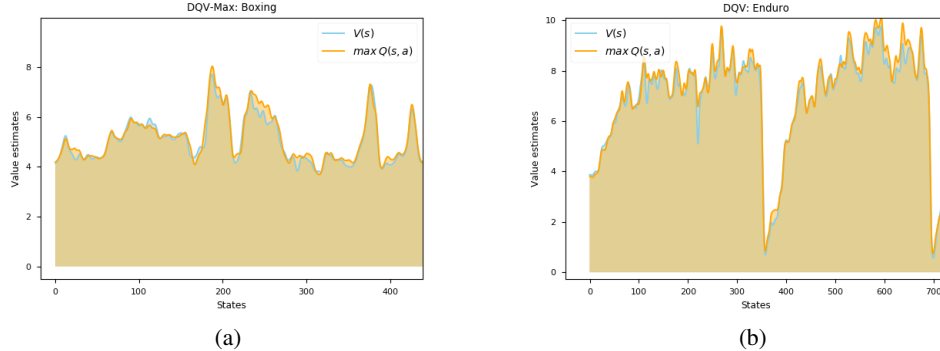

Figure 6: The results showing that the value estimates of the V network are never significantly higher than the ones coming from the Q network for a set of randomly sampled states. On the contrary, the $V(s)$ is mostly lower than the respective $\max Q(s, a)$ estimate. This suggests that DQV and DQV-Max do not overestimate the state-value function instead of the state-action value function.

4   Final Evaluation Performance

In order to better evaluate the performance of all analyzed algorithms we now present results which have been obtained on a larger set of `Atari` games. The main aim of this section is to investigate whether the DQV-Max algorithm can be as successful as DQV and if both algorithms are able to outperform DQN and DDQN. Since due to space limitations it is not possible to present the learning curves for each tested game, we refer the reader to the supplementary material.

Instead, we report the different learning speeds and scores of each algorithm with respect to its overall performance in Table 1. We divide the total amount of testing-episodes that are required for the algorithms to converge in three different testing-stages, equally separated, and denoted respectively as ①, ② and ③. For each stage, we report the score that the agent has obtained at that corresponding testing episode. A visualization of such stages can be seen in Fig. 7 where they are denoted by the black dashed lines. As a concrete example of what is reported in the table, we can consider the DQV-Max algorithm and its results obtained on the game `Enduro`. It is possible to see that the last reported score in the table corresponds to 748.5 which matches with the score that is represented by the

Figure 7: A visualization of the evaluation protocol used in this work.

yellow line in Fig. 7. For each game and testing-stage, we report the results of the best performing algorithm in a green cell, while the second-best result is shown in a yellow cell. For the case of the `River Raid` environment there is no green cell at the last testing stage since the results are not in favor of a single algorithm. Our results show that there is at least one algorithm between DQV and DQV-Max which outperforms DQN and DDQN. This empirically highlights the benefits of learning two value functions simultaneously on a larger set of DRL environments. It is worth noting that the novel DQV-Max algorithm introduced in this work is able to mostly perform as well as DQV and even outperform it in four different games.
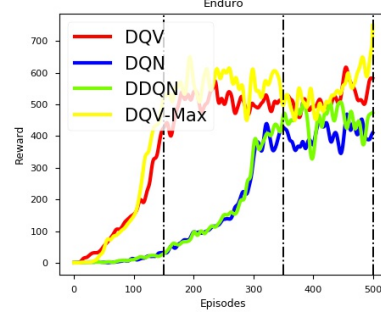
Table 1: The performance obtained by all analyzed algorithms on 7 `Atari` games. For each game, we report the score obtained by the agents which is measured at three different testing-stages as explained in Sec. 4. For each stage, the best performing algorithm is represented in a green cell and the second-best performing one in a yellow cell. We can see that for every game there is at least one algorithm between DQV and DQV-Max which learns either faster or better than DQN and DDQN. Specific attention should be given to the games `Pong`, `Enduro`, `Space-Invaders` and `River Raid`, where the fastest performing algorithm is DQV-Max.

Environment	DQN			DDQN			DQV			DQV-Max		
	①	②	③	①	②	③	①	②	③	①	②	③
Pong	−17.5	13.3	18.9	−19.5	10.5	19.2	10.2	14.5	19.4	2.5	17.5	19.9
Enduro	26	437.5	457	30.4	431.8	467.85	447.5	464.5	506.8	482.2	625.5	748.5
Boxing	−1	34.3	55.5	1.3	40.2	56.8	38.5	63.5	70.3	28.8	57.5	68.5
Sp.Invaders	347.5	387.5	585.3	160.5	261.4	598.8	215.6	480.8	528.4	385.4	650.5	612.9
IceHockey	−12	−7.5	−6.2	−13	−6.1	−3.5	−8.4	−5	−2.1	−9.5	−6.9	−3.1
Ms-Pacman	280	675	958	410.8	520.6	1015	685	915	1100	550	695	985
River Raid	1060	2150	2450	1220	2130	2150	1520	2075	2075	2070	2280	2450

5   Conclusion

In this work, we have made one step towards properly characterizing a new family of DRL algorithms which simultaneously learns an approximation of the V function and the Q function. We have started by thoroughly analyzing the DQV algorithm [15], and have shown in Sec. 3.1 that one key component of DQV is to use two independently parameterized neural networks for learning the V and Q functions. We have then borrowed some ideas from DQV to construct a novel DRL algorithm in Sec. 3.2 in order to show that approximating two value functions instead of one is just as beneficial in an *off-policy* learning setting as it is in an *on-policy* learning one. We have then studied how the DQV and DQV-Max algorithms are related to the DRL *Deadly Triad*, and hypothesized that the promising results obtained by both algorithms, could partially be achieved because both algorithms could suffer less from the overestimation bias of the Q function. From Sec. 3.3 we have concluded that this was indeed the case, even though DQV-Max seems to be more prone to suffer from this phenomenon. We have then ended the paper with an in-depth empirical analysis of the studied algorithms in Sec. 4, which generalizes all the results used in the previous sections to a larger set of DRL test-beds. To conclude, this paper sheds some light on the benefits that could come from approximating two value

functions instead of one, and properly characterizes a new family of DRL algorithms which follow such learning dynamics.

References

[1] Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards characterizing divergence in deep Q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[3] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[4] Justin A Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376, 1995.

[5] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[6] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018.

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

[8] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[9] 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. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

[11] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[12] 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.

[13] 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.

[14] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[15] Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco Wiering. Deep quality-value (dqv) learning. In *Advances in Neural Information Processing Systems, Deep Reinforcement Learning Workshop*. Montreal, 2018.

[16] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

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

[18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[19] 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.

[20] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

[21] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[22] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003, 2016.

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

[24] 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.

[25] 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.

6 Supplementary Material

6.1 Experimental Setup

We follow the experimental setup which is widely used in the literature [12, 21, 9]. A convolutional neural network with three convolutional layers is used [12], followed by a fully connected layer. The network receives the frames of the different games as input. When it comes to DQV and DQV-Max we use the same architecture for approximating the V function and the Q function with the only difference being the size of the output layer. For the V network this is simply 1, while for the Q network there are as many output nodes as possible actions. Both networks are trained with the RMSprop optimizer [19] initialized as in [12]. The version on the ALE environment corresponds to the `Deterministic-v4` one which uses '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 for resizing each frame of the environment to an 84×84 gray-scaled matrix and as exploration strategy we use for all algorithms the epsilon-greedy approach. Regarding the target networks which are used in all our experiments, we update their weights each time the agent has performed a total of 10,000 actions. Lastly, the discount factor γ is set to 0.99 and the size of the memory buffer is set to contain 400,000 transitions. The latter choice is motivated by a lack of sufficient computational resources that did not allow us to use an experience replay buffer able to collect up to 1,000,000 transitions as originally used in the DQN algorithm. For similar reasons, we have limited our analysis to a subset of games which show constant and successful learning within 3 days of training. This is different from e.g. the DDQN algorithm [21] which allocates one week of training per game. For most of the tested games, we noticed that the obtained results did not significantly change from the ones reported in the literature. However, when this was the case, this can be related to a partially lower amount of training time reserved to the algorithms, and the limited memory size of the experience replay buffer. Nevertheless, since all algorithms use the same experimental setting we believe that this should not change the significance of the results. In total, we have run experiments on 7 different games for which each algorithm is tested by running 5 different simulations with 5 different random seeds. The code for reproducing our experiments can be found at `https://github.com/paintception/Deep-Quality-Value-Family-/tree/master/src`.

6.2 Pseudocode of DQV and DQV-Max learning.

This section reports the pseudo-codes of Deep Quality-Value (DQV) learning (Algorithm 1) and its novel Deep Quality-Value-Max (DQV-Max) extension (Algorithm 2). Each algorithm requires the

initialization of two neural networks that are required for approximating the state-value function Φ and the state-action value function θ. DQV requires a target network for estimating the state-value (V) function which is initialized as Φ^-, whereas DQV-Max requires it for learning the state-action value (Q) function, therefore it is defined as θ^-. For our experiments the Experience Replay buffer D is set to contain at most 400,000 trajectories (N), from which we start sampling as soon as it contains 50000 ($\mathcal{N}$) trajectories ($\langle s_t, a_t, r_t, s_{t+1} \rangle$). D is handled as a queue: when its maximum size is reached, every new sample stored in the queue overwrites the oldest one. We also initialize the counter-variable, total_a which is required for keeping track of how many actions the agent has performed over time. Once it corresponds to the hyperparameter c we update the weights of the target network with the ones of the main network. Regarding the targets that are constructed in order to compute the Temporal-Difference errors we refer to them as y_t in the DQV algorithm, while as v_t and q_t when it comes to DQV-Max. This is done in order to highlight the fact that the latter algorithm requires the computation of two different targets to bootstrap from.

Algorithm 1 DQV Learning

Require: Experience Replay Queue D of maximum 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: **while** True **do**
  2:      set s_t as the initial state
  3:      **while** s_t is not over **do**
  4:          select $a_t \in \mathcal{A}$ for s_t with policy π (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 $B = \{\langle s_t^i, a_t^i, r_t^i, s_{t+1}^i \rangle | i = 1, \ldots, 32\}$ of size 32 from D
11:             **for** i = 1 to 32 **do**
12:                **if** s_{t+1}^i is over **then**
13:                    $y_t^i := r_t^i$
14:                **else**
15:                    $y_t^i := r_t^i + \gamma V(s_{t+1}^i, \Phi^-)$
16:                **end if**
17:             **end for**
18:             $\theta := \underset{\theta}{\arg\min} \sum_{i=1}^{32} (y_t^i - Q(s_t^i, a_t^i, \theta))^2$
19:             $\Phi := \underset{\Phi}{\arg\min} \sum_{i=1}^{32} (y_t^i - V(s_t^i, \Phi))^2$
20:             total_a += 1
21:             **if** total_a = c **then**
22:                $\Phi^- := \Phi$
23:                total_a := 0
24:             **end if**
25:          **end if**
26:      **end while**
27: **end while**

6.3 Additional Learning Curves

We report the additional learning curves obtained on the games `Space-Invaders`, `Ms-Pacman`, `Ice-Hockey` and `River Raid`.

Algorithm 2 DQV-Max Learning

Require: Experience Replay Queue D of maximum size N
Require: Q networks with parameters θ and θ^-
Require: V network with parameters Φ
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 over **do**
4:          select $a_t \in \mathcal{A}$ for s_t with policy π (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 $B = \{\langle s_t^i, a_t^i, r_t^i, s_{t+1}^i \rangle | i = 1, \ldots, 32\}$ of size 32 from D
11:              **for** i = 1 to 32 **do**
12:                  **if** s_{t+1}^i is over **then**
13:                      $v_t^i := r_t^i$
14:                      $q_t^i := r_t^i$
15:                  **else**
16:                      $v_t^i := r_t^i + \gamma \max_{a \in \mathcal{A}} (Q(s_{t+1}^i, a, \theta^-))$
17:                      $q_t^i := r_t^i + \gamma\, V(s_{t+1}^i, \Phi)$
18:                  **end if**
19:              **end for**
20:              $\theta := \arg\min_{\theta} \sum_{i=1}^{32} (v_t^i - Q(s_t^i, a_t^i, \theta))^2$
21:              $\Phi := \arg\min_{\Phi} \sum_{i=1}^{32} (q_t^i - V(s_t^i, \Phi))^2$
22:              total_a += 1
23:              **if** total_a = c **then**
24:                  $\theta^- := \theta$
25:                  total_a := 0
26:              **end if**
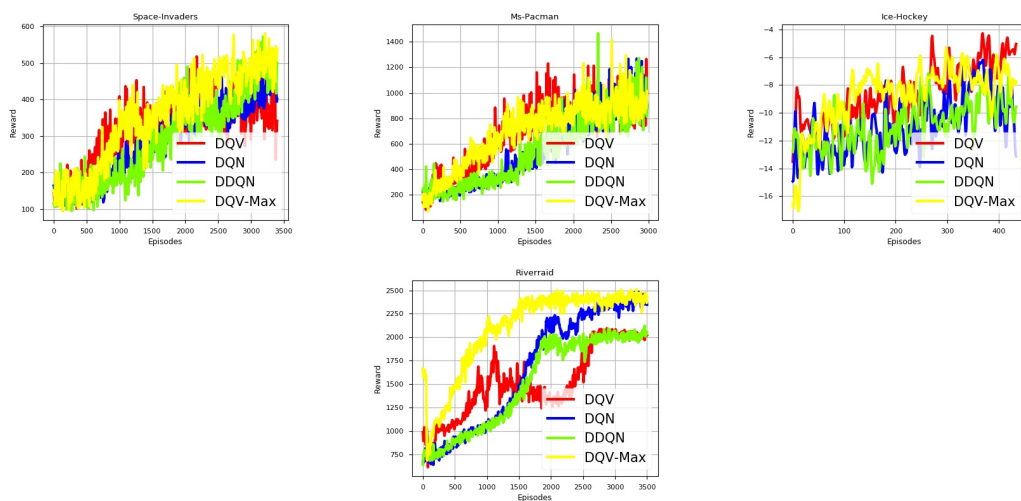27:   **end if**
28:      **end while**
29: **end while**

Figure 8: Additional learning curves obtained on more `Atari` games.