An Analysis of Action-Value Temporal-Difference Methods That Learn State Values

Anonymous authors
Paper under double-blind review

Keywords: TD learning, QV-learning, Dueling DQN, advantage estimation.

Summary

The hallmark feature of temporal-difference (TD) learning is bootstrapping: using value predictions to generate new value predictions. The vast majority of TD methods for control learn a policy by bootstrapping from a single action-value function (e.g., Q-learning and Sarsa). Significantly less attention has been given to methods that bootstrap from two asymmetric value functions: i.e., methods that learn state values as an intermediate step in learning action values. Existing algorithms in this vein can be categorized as either QV-learning or AV-learning. Though these algorithms have been investigated to some degree in prior work, it remains unclear if and when it is advantageous to learn two value functions instead of just one—and whether such approaches are theoretically sound in general. In this paper, we analyze these algorithmic families in terms of convergence and sample efficiency. We find that while both families are more efficient than Expected Sarsa in the prediction setting, only AV-learning methods offer any major benefit over Q-learning in the control setting. Finally, we introduce a new AV-learning algorithm called Regularized Dueling Q-learning (RDQ), which significantly outperforms Dueling Deep Q-Network in the MinAtar benchmark.

Contribution(s)

1. We prove that QV-learning converges for on-policy prediction under standard assumptions.
   Context: Wiering (2005) introduced the QV-learning algorithm, but omitted a convergence proof. To our knowledge, there is no published convergence proof of QV-learning to date.
2. We raise the issue that QVMAX, the main off-policy control variant of QV-learning, is biased. We empirically demonstrate that such bias can significantly impact performance. We then introduce a new, unbiased algorithm, BC-QVMAX, and empirically demonstrate that it converges to a similar solution to that of Q-learning in our considered settings.
   Context: Wiering & van Hasselt (2009) introduced QVMAX without theoretical justification, heuristically mirroring the Q-learning update. Its bias has not been identified until now. Our empirical results were obtained with parametric MDP experiments that we designed; they should be interpreted only as anecdotal evidence for the convergence of BC-QVMAX.
3. In the context of AV-learning algorithms, we formalize a tabular version of Dueling DQN, and we introduce a new algorithm, Regularized Dueling Q-learning (RDQ). RDQ addresses the identifiability issue of the naive dueling decomposition by using an l_2 penalty instead of subtracting the mean advantage. We empirically demonstrate that, given the same network architecture, RDQ significantly outperforms Dueling DQN in the MinAtar domain.
   Context: Wang et al. (2016) introduced Dueling DQN from the perspective of an improvement to the neural network architecture used by DQN. RDQ is the result of relaxing the semantics behind estimating $Q(s, a)$ through two value functions. Instead of requiring $Q(s, a)$ to be approximated by $V(s)$ and $A(s, a)$, RDQ searches for the closest point on the hyperplane defined by two arbitrary functions that sum to $Q(s, a)$. We used heavily tuned versions of DQN and Dueling DQN as baselines for MinAtar (Ceron & Castro, 2021).

An Analysis of Action-Value Temporal-Difference Methods That Learn State Values

Anonymous authors
Paper under double-blind review

Abstract

The hallmark feature of temporal-difference (TD) learning is bootstrapping: using value predictions to generate new value predictions. The vast majority of TD methods for control learn a policy by bootstrapping from a single action-value function (e.g., Q-learning and Sarsa). Significantly less attention has been given to methods that bootstrap from two asymmetric value functions: i.e., methods that learn state values as an intermediate step in learning action values. Existing algorithms in this vein can be categorized as either QV-learning or AV-learning. Though these algorithms have been investigated to some degree in prior work, it remains unclear if and when it is advantageous to learn two value functions instead of just one—and whether such approaches are theoretically sound in general. In this paper, we analyze these algorithmic families in terms of convergence and sample efficiency. We find that while both families are more efficient than Expected Sarsa in the prediction setting, only AV-learning methods offer any major benefit over Q-learning in the control setting. Finally, we introduce a new AV-learning algorithm called Regularized Dueling Q-learning (RDQ), which significantly outperforms Dueling Deep Q-Network in the MinAtar benchmark.

1 Introduction

Reinforcement learning (RL) is the study of decision-making agents that interact with their environment to maximize a notion of cumulative reward. Temporal-difference (TD) learning is among the most widely used classes of RL algorithms. Like dynamic programming, TD methods learn one or more value functions to guide policy improvement. However, unlike dynamic programming, TD approaches do not assume access or knowledge of the environment's dynamics, and as such value-function estimates must be generated iteratively through direct interaction with the MDP.

The key feature of TD is bootstrapping: constructing new value predictions from other value predictions. How such predictions should be integrated for the purposes of bootstrapping to learn as efficiently as possible has been the subject of much RL research. For example, classic methods like Q-learning (Watkins, 1989), Sarsa (Rummery & Niranjan, 1994), and Expected Sarsa (John, 1994) all learn action-value functions, and differ only by their bootstrapping terms, but exhibit dramatically different learning properties in terms of risk aversion, bias-variance trade-off, and convergence.

The vast majority of TD algorithms that attempt to learn an optimal control policy, including the three algorithms mentioned above, rely on just a single action-value function, denoted by $Q(s, a)$ where (s, a) is a state-action pair. A far less common paradigm is to jointly learn a secondary state-value function, $V(s)$, in the process of estimating $Q(s, a)$. That is to say, such TD methods learn state values as an intermediate step in learning action values.[1]

We broadly categorize such methods as either *QV-learning* or *AV-learning*. In QV-learning, the agent directly estimates $Q(s, a)$ and $V(s)$, with mutual bootstrapping between the two value functions. For

[1] Note that this precise definition of learning state and action values excludes double action-value methods such as Double Q-learning (van Hasselt, 2010), which are beyond the scope of this paper.

instance, the classic QV-learning method (Wiering, 2005) essentially replaces the bootstrap term of Expected Sarsa (John, 1994) with a learned approximation generated by TD(0) (Sutton, 1988). Conversely, AV-learning makes use of the advantage decomposition (Baird, 1993), which breaks down the action value as $Q(s,a) = V(s) + \mathrm{Adv}(s,a)$. In contrast to QV-learning, $\mathrm{Adv}(s,a)$ and $V(s)$ do not explicitly bootstrap from one another, but do bootstrap from the composite action-value function, $Q(s,a)$. This technique was popularized in deep RL by the dueling network architecture (Wang et al., 2016). Analyzing the behavior of the underlying Dueling Q-learning algorithm is important to understand how these networks learn.

In spite of past empirical evidence supporting QV-learning (e.g., Sabatelli et al., 2020; Modayil & Abbas, 2023) and AV-learning (Baird, 1993; Wang et al., 2016; Tang et al., 2023), the exact circumstances and mechanisms behind these improvements remain unclear. After all, estimating two value functions instead of one inherently requires more parameters to be learned. On the other hand, it seems that there are opportunities for synergistic information sharing between the value functions which could potentially explain the observed sample-efficiency gains. Still, it is unknown when or how reliably such gains can be obtained. Furthermore, several of these methods—particularly in the QV-learning family—are missing formal convergence guarantees.

Towards an understanding of when and how state-value estimation can be leveraged to learn action values more quickly (and soundly), we investigate the theoretical underpinnings of both QV-learning and AV-learning algorithms and conduct experiments to isolate some of their unique properties. We show that QV-learning is often more efficient than Expected Sarsa for on-policy prediction and that the performance gap increases with more available actions. However, QV-learning's main extension for off-policy control, QVMAX (Wiering & van Hasselt, 2009), suffers from bias; even when we correct this bias, we show empirically that its sample efficiency is surpassed by that of Q-learning. In contrast, we find that AV-learning methods work much more consistently for control problems overall and easily outperform Q-learning. We discuss the implicit assumptions encoded by Dueling Q-learning and introduce an algorithm called Regularized Dueling Q-learning (RDQ) that converges to a different AV-decomposition. An advantage of RDQ is that it easily extends to function approximation, and we show that it is significantly more sample efficient than Dueling DQN (Wang et al., 2016) in the five MinAtar games (Young & Tian, 2019) when using the same network architecture and hyperparameters. Our results help to characterize the nuanced distinction between these two main algorithm families, and ultimately motivate state-value estimation as a sound and effective way to accelerate action-value estimation.

2 Background

We formalize the RL problem as a standard Markov decision process (MDP) described by $(\mathcal{S}, \mathcal{A}, p, \mathcal{R}, \gamma)$. At each time step $t \geq 0$, the agent makes an observation of the environment state, $S_t \in \mathcal{S}$, and executes an action, $A_t \in \mathcal{A}$. The environment consequently transitions to a new state, $S_{t+1} \in \mathcal{S}$, and returns a reward, $R_{t+1} \in \mathcal{R}$, with probability $p(S_{t+1}, R_{t+1} \mid S_t, A_t)$.

In the policy evaluation setting, the agent's goal is to predict the action-value function: the expected discounted returns achieved when behaving according to some policy π, a mapping from states to distributions over actions. Letting $G_t = \sum_{i=0}^{\infty} \gamma^i R_{t+1+i}$ be the discounted return at time t, the action-value function is defined as

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid (S_t, A_t) = (s,a)]\,, \tag{1}$$

where the expectation $\mathbb{E}_\pi$ indicates that actions are sampled according to π. For any policy π, the corresponding action-value function q_π uniquely solves the action-value Bellman equation (Bellman, 1966):

$$q_\pi(s,a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) \left(r + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s')\, q_\pi(s', a') \right). \tag{2}$$

In the control setting, the agent's goal is to find an *optimal* policy: one that simultaneously maximizes $q_\pi(s,a)$ for every state-action pair (s,a). Every optimal policy has the same action-value function, denoted by q_*, which uniquely solves the action-value Bellman optimality equation:

$$q_*(s,a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma \max_{a' \in \mathcal{A}} q_*(s', a') \right) . \tag{3}$$

The policy evaluation and control settings are closely related, as accurate estimates of the expected returns under a particular policy can be leveraged to generate a better policy.

Temporal-difference (TD) methods for control estimate action-value functions, either q_π or q_*, from sample-based interaction with the environment. Given a transition $(S_t, A_t, S_{t+1}, R_{t+1})$, the agent then conducts an incremental update to its estimate of the action-value function, $Q(S_t, A_t)$. In the *off-policy* setting, the agent is assumed to select actions with probability $b(A_t|S_t)$, where b is a behavior policy which differs from the target policy, π.

Most of the algorithms considered in this paper are off-policy methods. They learn an action-value function $Q(s,a)$ that estimates either $q_\pi(s,a)$ or $q_*(s,a)$ using samples obtained from executing policy b in the environment. For example, Expected Sarsa (John, 1994) is an off-policy TD method that uses explicit knowledge of the target policy π to approximate the solution to Eq. (2) from sample-based approximation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t) \right) , \tag{4}$$

where $\alpha \in (0, 1]$ is the step size of the update. We revisit Expected Sarsa as an important baseline several times in our work because it generalizes a number of fundamental bootstrapping algorithms (van Hasselt, 2011). For instance, when the target policy is greedy with respect to Q, the expectation becomes equivalent to $\max_{a'} Q(s', a')$, yielding the Q-learning algorithm:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - Q(S_t, A_t) \right) , \tag{5}$$

which will be useful for the control case in Section 3.2.

2.1 QV-learning

We now begin to discuss methods that learn state values as a means to estimate action values. The state-value function v_π is defined analogously to as

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \tag{6}$$

and uniquely solves the Bellman equation

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma v(s') \right) . \tag{7}$$

The state-value function is related to the action-value function by $v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$. This implies that $v_\pi(s')$ can be substituted as an intermediate quantity in Eq. (2) to avoid the summation over actions. QV-learning (Wiering, 2005) is an on-policy TD algorithm based on this concept. The idea is to use TD(0) (Sutton, 1988) to independently learn $V(s) \approx v_b(s)$, while concurrently bootstrapping from these estimated state values to learn $Q(s,a) \approx q_b(s,a)$. The specific value-function updates become

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - Q(S_t, A_t) \right) , \tag{8}$$

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right) . \tag{9}$$

Wiering (2005, Sec. 2) is explicit that $Q(S_t, A_t)$ is always updated before $V(S_t)$, though the order is irrelevant in the tabular case. Additionally, it is commonly assumed that both updates share the same step size, α, as is shown here. We analyze this algorithm in Section 3.1.

The main variant of QV-learning for off-policy control is QVMAX (Wiering & van Hasselt, 2009), which essentially substitutes the max operator from Q-learning into Eq. (9) in an attempt to induce convergence to q_* instead of q_b:

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - V(S_t) \right) .$$

The update to $Q(S_t, A_t)$ remains the same as Eq. (8) and is still conducted prior to Eq. (12) on each time step. We analyze this algorithm in Section 3.2.

2.2 Dueling Q-learning

The basic idea behind Dueling Q-learning (Wang et al., 2016) is to decompose the action-value function, $Q(s, a)$, into a state-value function, $V(s)$, and an advantage function, $A(s, a)$. The particular decomposition used by Wang et al. (2016) is

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s, a') , \tag{10}$$

where the last term is subtracted to make the solution unique (we further discuss this point in Section 4.2). Then, the squared-error Q-learning loss is minimized by stochastic gradient descent (SGD) with respect to the action-value function's subcomponents:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - Q(S_t, A_t) \right) \nabla Q(S_t, A_t) . \tag{11}$$

Substituting Eq. (10) into $\nabla Q(S_t, A_t)$ and applying the chain rule would yield the complete update. In practice, the decomposition in Eq. (10) is represented by a branching and merging neural network, and the chain rule is automatically handled by backpropagation (Rumelhart et al., 1986) which computes the partial derivatives with respect to the neural network's many parameters. In Section 4.1, we instead apply the chain rule to Eq. (10) assuming tabular value functions to derive a Dueling Q-learning update without function approximation, allowing us to gain further insight into this fundamental algorithm beyond the deep RL setting.

3 QV-learning Algorithms

In this section, we analyze QV-learning algorithms, which jointly learn an action-value function $Q(s, a)$ and a state-value function $V(s)$. Although the fundamental QV-learning algorithm has existed for about 20 years and several studies, theoretical analysis has been limited.

3.1 On-Policy Prediction

In this subsection, we conduct a focused experiment to determine when QV-learning these questions. We then conclude with a formal convergence analysis of QV-learning.

During our experiments, we revisit Expected Sarsa several times as a prototypical baseline for TD learning with action values. Recall from Section 2 that Expected Sarsa's update rule is

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big(R_{t+1} + \gamma \underbrace{\sum_{a' \in \mathcal{A}} \pi(a'|S_{t+1}) Q(S_{t+1}, a')}_{V(S_{t+1})} - Q(S_t, A_t) \Big) ,$$

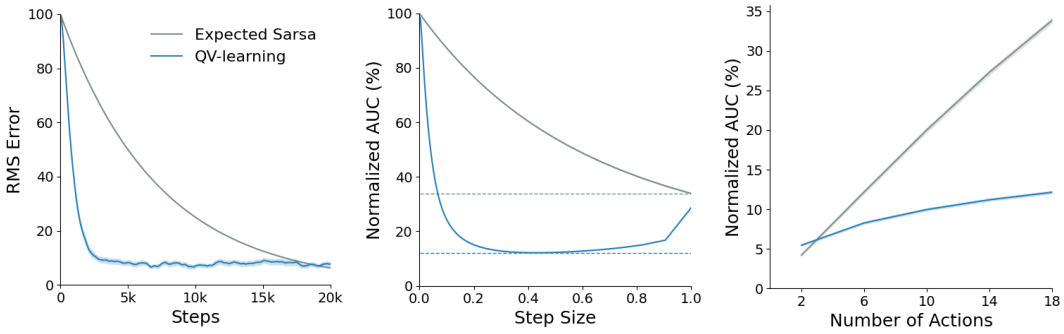

Figure 1: On-policy prediction performance with increasing number of actions.

The comparison to Expected Sarsa is natural because QV-learning can be seen as learning approximation $V(S_{t+1})$ to the expected next action value, which we have underscored above. This provides a clue as to when QV-learning may have an advantage over Expected Sarsa; as the action-space cardinality $|\mathcal{A}|$ of the MDP grows, Expected Sarsa requires many more sample interactions to estimate all of the action values in the expectation, whereas QV-learning can simply learn $V(S_{t+1})$ with a small number of samples that is roughly independent of $|\mathcal{A}|$.

To test this, we design a parametric MDP experiment where the environment has 4 states and $|\mathcal{A}|$ actions. Taking any action in any state triggers a transition to a random state (possibly the same one) with equal probability. The agents' behavior policy is uniform random in every state. Agents receive a reward of $+1$ whenever the first action, a_0, is taken, and a reward of 0 otherwise.

Because the MDP dynamics are state invariant, $v_\pi(s)$ is a constant for all $s \in \mathcal{S}$. Solving the state-value Bellman equation, Eq. (7), yields $v_\pi(s) = 1 \,/\, \big((1-\gamma)\,|\mathcal{A}|\big)$. It follows that $q_\pi(s, a_i) = \mathbf{1}_{i=0} + \gamma \,/\, \big((1-\gamma)\,|\mathcal{A}|\big)$ for all $a_i \in \mathcal{A}$, where $\mathbf{1}$ is the indicator function.

We evaluate the agents' performance by measuring the root mean square (RMS) prediction error, $\|q_\pi - Q\|_2$, as a function of the number of environment interactions. We normalize the errors by expressing them as a percentage of the initial RMS error, $\|q_\pi - Q_0\|_2$, where Q_0 is initialized with zeros. We compare Expected Sarsa and QV-learning, training both agents for 20,000 time steps. In Figure 1 (left), we plot the learning curves for the case where $|\mathcal{A}| = 18$, the largest instantiation of our MDP that we consider here. We then average the results over 100 independent trials, with shading to indicate 95% confidence intervals. The agents' step sizes are chosen from a natural-logarithmic grid search of 61 values over the interval $(0, 1]$, similarly to the random walk experiment of Sutton & Barto (2018, Sec. 12.1). The selection criterion is to minimize the area under the curve (AUC), which we normalized as a percentage. Specifically, an AUC of 100% indicates that the initial error did not change at all, whereas a lower percentage indicates faster average progress towards the fixed point. In Figure 1 (center), we plot the corresponding AUCs for each step size tested by the sweep. The dashed horizontal line corresponds to the smallest AUC achieved by each agent.

To investigate the scalability of these methods, we repeat this experiment setup for $|\mathcal{A}| \in \{2, 6, 10, 14, 18\}$. We then plot the best AUC achieved in each problem instance in Figure 1 (right). The slope of the resulting lines roughly correspond to the scalability of the algorithms. We note that QV-learning's line is much closer to the horizontal, indicating better robustness to large action-space cardinalities, as we hypothesized earlier. However, the trade-off for this improved sample efficiency is a much noisier update, which can be seen in Figure 1 (left); if we were to train for longer, Expected Sarsa would eventually obtain a more accurate solution. This indicates that QV-learning is perhaps best suited for cases where learning a decent—but possibly imperfect—value function is required in a short amount of time (e.g., policy learning).

These results do not, however, suggest that QV-learning fails to converge to the correct pair of on-policy value functions, q_b and v_b. The noisy behavior observed in Figure 1 (left) is primarily due to

the fact that the step size is not being annealed. The following theorem shows that, in expectation, the joint update of QV-learning is a contraction mapping when we represent the two value functions as a concatenated vector of size $|\mathcal{S}| + |\mathcal{S} \times \mathcal{A}|$.

Theorem 3.1 (QV-learning contraction)**.** *The expected QV-learning update corresponds to a linear joint operator* $H : \left[\begin{smallmatrix} \boldsymbol{v} \\ \boldsymbol{q} \end{smallmatrix}\right] \mapsto \boldsymbol{b} + \boldsymbol{A}\left[\begin{smallmatrix} \boldsymbol{v} \\ \boldsymbol{q} \end{smallmatrix}\right]$*, where* $\boldsymbol{b} = \left[\begin{smallmatrix} \boldsymbol{P}_\pi \boldsymbol{r} \\ \boldsymbol{r} \end{smallmatrix}\right]$ *and* $A = \begin{bmatrix} \gamma \boldsymbol{P}_\pi \boldsymbol{P}_p & \boldsymbol{0} \\ \gamma \boldsymbol{P}_p & \boldsymbol{0} \end{bmatrix}$*. The operator* H *is a contraction mapping with its unique fixed point equal to* $\left[\begin{smallmatrix} \boldsymbol{v}_\pi \\ \boldsymbol{q}_\pi \end{smallmatrix}\right]$.

Proof. See Appendix A. □

The significance of this result is that it shows that both value functions make progress (on average) towards their respective fixed points, without having to wait for V to converge first. This is in contrast to less formal convergence arguments for QV-learning, which may rely on the assumption that TD(0) would first converge to v_π based on well-established convergence results, and then Eq. (8) would be able to bootstrap from it to extract q_π. The issue with this two-timescale approach is that V may never exactly equal v_π after any finite amount of time, meaning that Q would still incur some bootstrapping bias—a caveat that is directly addressed by considering the joint operator space, as we have done here.

The fact that H is a contraction mapping implies that both Q and V converge to their respective fixed points even when updates are conducted asynchronously, under the additional (but standard) technical assumptions that the step size α is appropriately annealed and the conditional variances of the updates are bounded (see Bertsekas & Tsitsiklis, 1996, Prop. 4.4). We note that the latter assumption is automatically satisfied by our MDP definition, which has finite sets of states, actions, and rewards. Finally, we remark that our analysis considers only the prediction setting in which the target policy π is fixed; for the control case, we would likely need to invoke a greedy in the limit with infinite exploration (GLIE) assumption (Singh et al., 2000) to prove eventual convergence to an optimal policy, but we leave this as an open problem.

3.2 Off-Policy Control

The previous experiment shows that bootstrapping from state values when learning action values can be much more efficient than directly learning the action values. Unfortunately, we cannot leverage this same technique for off-policy control, as this would require a model-free method for estimating v_* directly from environment samples.

To get around this, Wiering & van Hasselt (2009) introduced an off-policy variant of QV-learning called QVMAX which borrows the max operator from Q-learning when updating $V(S_t)$:

$$V(S_t) \leftarrow V(S_t) + \alpha\big(R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - V(S_t)\big)\,. \tag{12}$$

One consequence of this change is that now there is a two-way information flow between Q and V. Previously, with QV-learning, we had only a one-way information flow: V bootstrapped from itself, and Q bootstrapped from V; hence, Q could not corrupt the state values in any way. We illustrate this point in Figure 2. Although this is not necessarily a bad property, it is worth noting that the fundamental characteristic of the algorithm has changed.

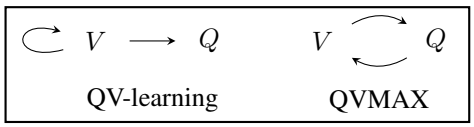

Figure 2: Depiction of bootstrapping in QV-learning variants. The arrows point from the bootstrapped value function to the value function being updated.

Another, more serious consequence of this change is that the update now suffers from off-policy bias. Wiering & van Hasselt (2009) hypothesized that the use of the max operator in Eq. (12) would induce convergence to q_*, just as it does in Q-learning. Unfortunately, this is not true; the update fails to correct the distribution of the observed reward, R_{t+1}, which is collected by the behavior policy. This is because Eq. (12) is *not* conditioned on the

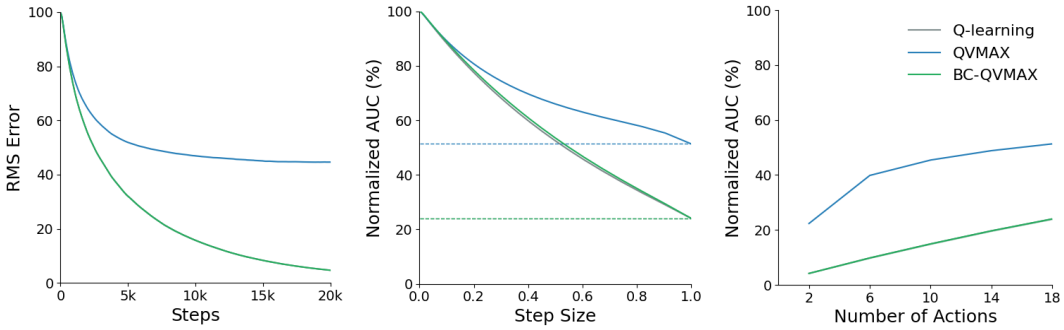

Figure 3: Off-policy control performance with a variable number of actions. Note that Q-learning is sometimes eclipsed by BC-QVMAX.

state-action pair, (S_t, A_t), as is normally the case for Q-learning, but is instead conditioned only on the state, S_t. The additional freedom in the selection of A_t means that the expected reward observed in state S_t now depends on behavior policy b.

To eliminate the bias, the update cannot depend explicitly on the sampled reward, R_{t+1}. We therefore propose to modify Eq. (12) to

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\max_{a \in \mathcal{A}} Q(S_t, a) - V(S_t) \right) . \tag{13}$$

We call this new variant *Bias-Corrected* QVMAX, or BC-QVMAX for short. This is the proper algorithm for learning q_*, as the update now approximates the Bellman optimality relationship $v_*(s) = \max_{a \in \mathcal{A}} q_*(s, a)$. However, a peculiarity of this update is that it closely resembles that of Q-learning in Eq. (5). In fact, if we set $\alpha = 1$, then Eq. (13) simply becomes a table overwrite: $V(S_t) \leftarrow \max_{a \in \mathcal{A}} Q(s, a)$. This would make the algorithm almost equivalent to Q-learning, but with an artificial delay introduced between bootstrapping. When $\alpha < 1$, then Eq. (13) has an extra smoothing effect, mixing together stale estimates of the maximal action value in each state and exacerbating this delay.

We slightly modify the prediction experiment from Section 3.1 to demonstrate the bias of QVMAX. This time, we target a greedy policy with respect to the current action-value function, making Expected Sarsa equivalent to Q-learning. We additionally compare QVMAX and BC-QVMAX. The optimal policy is to select a_0 unconditionally, making $q_*(s, a_0) = 1 \,/\, (1 - \gamma)$. It follows that $q_*(s, a_i) = \gamma \,/\, (1 - \gamma)$ for $i \neq 0$.

All experiment and plotting procedures remain the same as before, except that now the RMS error is defined in terms of q_*: i.e., $\|q_* - Q\|_2$. We plot the results in Figure 3. As can be seen in Figure 3 (left), and as our theory predicted, QVMAX asymptotes substantially above the zero-error mark due to its biased update. In contrast, BC-QVMAX achieves a much lower error. However, as we also discussed above, the smoothing effect of BC-QVMAX makes it similar to, but rather slower than, Q-learning. The results in Figure 3 (left) are identical only because $\alpha = 1$ happens to be the best step size for both methods, which makes them nearly equivalent (up to the 1-step delay mentioned previously). However, for all $\alpha < 1$, we can see from Figure 3 (center) that BC-QVMAX slightly lags behind Q-learning.

These results unfortunately indicate that off-policy control with QV-learning algorithms is much harder than on-policy prediction, since model-free estimation of v_* is not straightforward. While we note that this one experiment is not enough to rule out the viability of QVMAX approaches in this setting, it does provide convincing evidence against it. In particular, the theoretically correct version of the algorithm, BC-QVMAX, is similar to a learned approximation of Q-learning which introduces delay into the bootstrapping. This does not serve an immediately obvious benefit and

appears to negatively impact sample efficiency, though there is the possibility that this smoothing effect could have utility elsewhere (e.g., to enhance stability in tracking problems).

4 AV-learning Algorithms

Rather than learning explicit state- and action-value functions that bootstrap from each other like QV-learning methods do, an alternative approach is to decompose the action-value function into constituent state-value and advantage functions that can be implicitly updated using gradient descent. This *dueling* strategy was introduced by Wang et al. (2016), but we generalize it significantly in this section.

4.1 Dueling Q-learning

To obtain a tabular Dueling Q-learning update, we evaluate the chain rule in Eq. (11), as we discussed earlier in Section 2.2. Let $\delta_t = R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - Q(S_t, A_t)$ be the Q-learning error. Because $Q(s,a)$ is defined according to Eq. (10), the chain rule preserves the error δ_t inside the quadratic term and then scales it in the following manner:

$$\mathrm{Adv}(S_t, a) \leftarrow \mathrm{Adv}(S_t, a) + \alpha \left(\mathbf{1}_{a=A_t} - \frac{1}{|\mathcal{A}|} \right) \delta_t \,, \quad \forall a \in \mathcal{A} \,, \tag{14}$$

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t \,. \tag{15}$$

It may seem unusual to write out these updates in this manner; Dueling Q-learning was originally proposed for deep RL, and backpropagation would just handle the partial derivatives automatically. Nevertheless, these updates are well defined for tabular value functions. Furthermore, they reveal an property behind this particular style of dueling. On each time step, a single advantage value is incremented in proportion to δ_t, and then the $|\mathcal{A}|$ advantage values are decremented in the same proportion to δ_t / $|\mathcal{A}|$. This implies that the arithmetic mean of the advantages is an *invariant* quantity; if we initialize the advantage values to zero, then the mean will remain zero throughout training.

4.2 Regularized Dueling Q-learning

We propose a new dueling algorithm which does not rely on subtracting the identifiability term in Eq. (10). We start with the most general decomposition of the action-value function,

$$Q(s,a) = V(s) + \mathrm{Adv}(s,a) \,. \tag{16}$$

As noted by Wang et al. (2016, Sec. 3), this decomposition is unidentifiable; we can add a constant to $V(s)$ and subtract the same constant $A(s,a)$ without changing $Q(s,a)$. Consequently, there are infinitely many valid decompositions that satisfy Eq. (16) and no way for an algorithm to choose between them.

Dueling DQN seeks to approximate both $v_\pi(s)$ and $\mathrm{Adv}_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$ accurately in order to estimate $q_\pi(s,a)$, which motivates the subtraction of the mean advantage (which is meant to coarsely approximate the expected advantage under the target policy). We propose an alternative approach in which we search for *any* two functions $V(s)$ and $\mathrm{Adv}(s,a)$ that accurately reconstruct $q_\pi(s,a)$, even if that means sacrificing the semantics of v_π and Adv_π.

Let us consider the underdetermined system of equations in Eq. (16) to motivate our algorithm. For a particular state-action pair (s,a), we can represent the specific solution found by Dueling Q-learning as an ordered pair, $(\mathrm{Adv}(s,a), V(s))$. This particular solution is just one of infinitely many ordered pairs that satisfy Eq. (16), and form a negatively sloped line in the Adv-V plane: $V(s) = q_\pi(s,a) - \mathrm{Adv}(s,a)$. When $Q(s,a)$ is initialized with zeros, the locus of possible initializations of $V(s)$ and $A(s,a)$ is also a negatively sloped line that passes through the origin: $V(s) = -A(s,a)$.

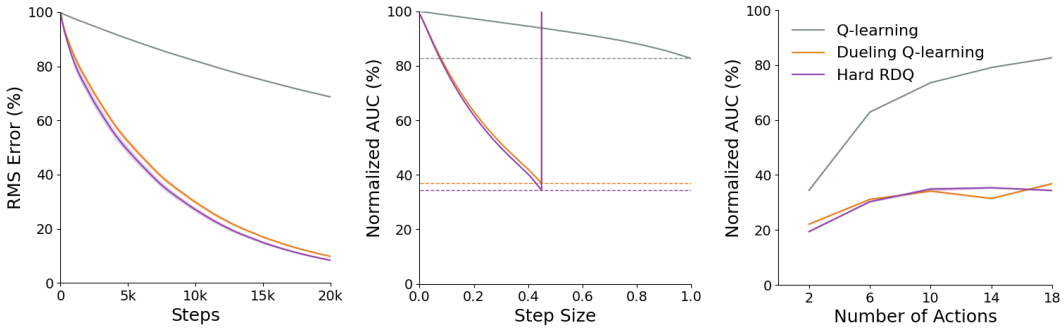

Figure 4: Off-policy control performance with a variable number of actions.

These two lines are parallel, and therefore the shortest directional path between them is the vector orthogonal to both of them—the minimum-norm projection. This fact remains true even if we apply a different initialization to $Q(s,a)$, since the slope of both lines is always -1 with respect to the advantage.

For this reason, we hypothesize that a good solution to Eq. (16) is the point on the solution line closest to the initialization of $V(s)$ and $\mathrm{Adv}(s,a)$, since it may require fewer updates to reach. In practice, because the state-value and advantage functions are typically initialized near zero for both tabular and deep RL, we can approximate this solution more simply as the closest point to the origin.

We therefore propose to apply an l^2 regularization to Eq. (11) to restore identifiability to Eq. (16) and encourage the algorithm to find this minimum-norm approximation. Specifically, we penalize the squared-error loss from Eq. (11) with a term of

$$\frac{1}{2}V(s)^2 + \sum_{a\in\mathcal{A}} \mathrm{Adv}(s,a)^2 \,. \tag{17}$$

Since we now have $Q(s,a) = V(s) + \mathrm{Adv}(a)$ (no identifiability term), evaluating the chain rule gives us the following updates:

$$\mathrm{Adv}(S_t,a) \leftarrow (1-\beta)\,\mathrm{Adv}(S_t,a) + \alpha \mathbf{1}_{a=A_t}\delta_t \,, \quad \forall a \in \mathcal{A} \,, \tag{18}$$

$$V(S_t) \leftarrow (1-\beta)V(S_t) + \alpha\delta_t \,, \tag{19}$$

where $\beta \in [0,1)$ is the regularization strength. We call this new algorithm Regularized Dueling Q-learning (RDQ). Specifically, we refer to this variant with the l^2 penalty as *Soft* RDQ. In practice, we found Soft RDQ to be rather noisy in the tabular setting, since the state and advantage values become leaky—they get pushed towards zero by the $1-\beta$ factor, and then have to compensate using the stochastic TD error, δ_t. However, it is very useful in the case of function approximation, since the penalty in Eq. (17) is architecturally agnostic and can be combined with complex neural architectures. We experiment with this in Section 4.3.

In the tabular setting, we do not need to utilize an l_2 penalty to obtain the desired minimum-norm solution. This is because the penalty in Eq. (17) is equal to $V(s)^2 + \sum_{a\in\mathcal{A}} \big(Q(s,a) - V(s)\big)^2$. Taking the derivative with respect to $V(s)$ and then solving it for zero gives us

$$V(s) = \sum_{a'\in\mathcal{A}} \mathrm{Adv}(s,a') \,. \tag{20}$$

Furthermore, when we remove the l^2 penalty from Eqs. (18) and (19) by setting $\beta = 0$, we see that both $V(s)$ and exactly one advantage value, $\mathrm{Adv}(s,a)$, are incremented by exactly the same amount one each time step—indicating that Eq. (20) is another invariant quantity. This implies that, regardless of how V and Adv are initialized, Eq. (20) always remains true (at least, up to a translation determined by the difference at initialization: $V_0(s) - \sum_{a\in\mathcal{A}} \mathrm{Adv}_0(s,a)$). Therefore,

rather than using an explicit L_2 penalty, we can directly apply the RDQ update without a penalty to achieve the minimum-norm solution in the tabular setting:

$$\mathrm{Adv}(S_t, a) \leftarrow \mathrm{Adv}(S_t, a) + \alpha \mathbf{1}_{a=A_t} \delta_t \,, \quad \forall a \in \mathcal{A} \,, \tag{21}$$

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t \,. \tag{22}$$

We call this variant *Hard* RDQ, since it explicitly follows the minimum-norm path (on average) to the solution without the use of a soft penalty. Unfortunately, there does not appear to be an easy extension of this idea to the function approximation case. If we were to apply these update rules under function approximation, the different gradients calculated for $V(s)$ and $\mathrm{Adv}(s, a)$ would violate the invariant and ruin the minimum-norm convergence property.

To test this new dueling algorithm, we repeat the off-policy control experiment from Section 3.2. We once again use Q-learning as a baseline, and then additionally compare Dueling Q-learning with Hard RDQ. The only minor modifications we make is that we now change the suboptimal reward from 0 to -1 (which does not change the optimal policy), set $\gamma = 0.999$, initialize V and Adv using zero-centered Gaussian noise with a standard deviation of 2.

We plot the results in Figure 4. All experiment and plotting procedures remain the same as before. In contrast to the QVMAX algorithm tested earlier, both dueling methods dramatically outperform Q-learning, showing that the advantage decomposition is a highly effective strategy in general. In the largest instantiation of our MDP, where $|A| = 18$, Hard RDQ slightly outperforms Dueling Q-learning across the range of step sizes (see Figure 4; left, center). Across the various MDP instances (see Figure 4; right), Hard RDQ performs slightly better than Dueling Q-learning in a small majority of cases, but both algorithms perform very well overall.

4.3 Deep RL Experiments

Because Dueling Q-learning was originally proposed as a network architecture for Deep Q-Network (DQN; Mnih et al., 2015), we test the performance of RDQ in a deep RL setting, where a neural network is used to approximate the action-value function. Our benchmark is the MinAtar domain (Young & Tian, 2019), which includes five Atari-like games: Asterix, Breakout, Freeway, Seaquest, Space Invaders. The state representation for MinAtar is 10×10 multi-channel binary images revealing various objects and their velocities. The reward function is the incremental game score.

We compare the soft variant of RDQ against DQN and Dueling DQN, using the hyperparameters from Ceron & Castro (2021). The action-value function $Q(s, a; \boldsymbol{\theta}_t)$ is approximated by a neural network, where $\boldsymbol{\theta}_t$ is the network parameters at time t. The network architecture we use for DQN is the same as that from Young & Tian (2019): a 16-filter, 3×3 convolutional layer, a 128-unit dense layer, and a final linear layer which maps to the $|\mathcal{A}|$ action values. All layers except the last apply a rectified linear unit (ReLU) activation function. We initialize the parameters using the LeCun normal scheme (LeCun et al., 2002).

For RDQ and Dueling DQN, we adapt this to a dueling architecture following Wang et al.'s 2016 procedure. We duplicate the 128-unit hidden layer to branch the network in parallel streams, and then separately map these into linear outputs of size $|\mathcal{A}|$ and 1 for estimating $\mathrm{Adv}(s, a; \boldsymbol{\theta}_t)$ and $V(s, a; \boldsymbol{\theta}_t)$, respectively. These are added together (with broadcasting) to compute $Q(s, a; \boldsymbol{\theta}_t)$. Dueling DQN additionally subtracts the identifiability term $\frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} \mathrm{Adv}(s, a'; \boldsymbol{\theta}_t)$ per Eq. (10).

The agents execute an ϵ-greedy policy, where ϵ is fixed to 1 for the first 1,000 time steps of training and then linearly annealed to 0.01 over the next 250,000 time steps. Each experience transition $(S_t, A_t, S_{t+1}, R_{t+1})$ is stored in a replay buffer with a capacity of 100,000 transitions. We let D_t denote the contents of the replay memory at time t. Every 4 time steps, the agents take a stochastic descent update on the parameters using the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 2.5×10^{-4} and a denominator constant of $\epsilon = 3.125 \times 10^{-4}$. Both DQN and Dueling DQN

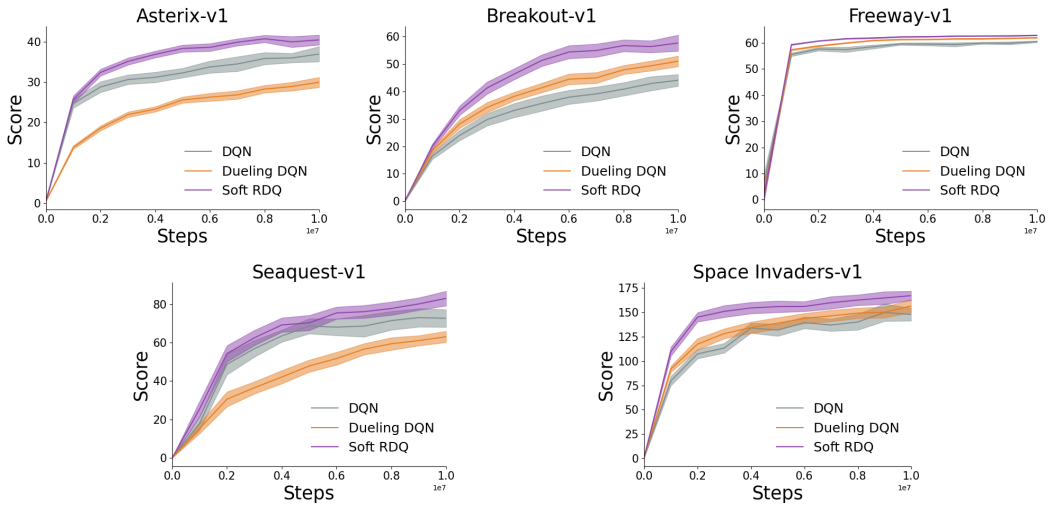

Figure 5: Deep RL results across five MinAtar games. Each algorithm is run for 30 seeds in each environment. The dark lines represent the mean across these 30 seeds, and the shaded region corresponds to a 95% confidence interval.

are trained to minimize the loss

$$L_t^{\text{DQN}} \stackrel{\text{def}}{=} \frac{1}{2}\,\mathbb{E}\left[\left(R + \gamma \max Q(S, A; \boldsymbol{\theta}^-) - Q(S, A; \boldsymbol{\theta})\right)^2\right], \tag{23}$$

where $\boldsymbol{\theta}^-$ is the target network parameters copied from $\boldsymbol{\theta}$ every 1,000 time steps. The expectation in Eq. (23) is taken over the uniform distribution of samples (S, A, R, S') in D_t; in practice, we approximate this with minibatches of size 32. RDQ inherits the same loss, but adds the regularization term:

$$L_t^{\text{RDQ}} \stackrel{\text{def}}{=} L_t^{\text{DQN}} + \frac{\beta}{2}\,\mathbb{E}\left[V(S;\theta)^2 + \sum_{a \in \mathcal{A}} \text{Adv}(S, a)^2\right]. \tag{24}$$

We choose $\beta = 10^{-3}$ for the regularization strength; we did not tune this value.

Each agent was trained for a total of 10 million time steps. Every 1 million time steps, we evaluated the current agent with an ϵ-greedy policy for 1,000 episodes with $\epsilon = 0.01$. In Figure 5, we plot the mean undiscounted return for the evaluation episodes as a function of training time; these results are averaged over 30 independent trials and the shading indicates 95% confidence intervals.

Figure 5 depicts our results. We see that between DQN and Dueling DQN, there does not appear to be a clearly superior algorithm, and one algorithm may outperform the other depending on the environment. However, we also see that RDQ significantly outperforms DQN and Dueling DQN in all five environments. Note that RDQ shares an identical architecture with Dueling DQN. Although $\beta = 10^{-3}$ works well in this setting, more experiments would be needed to determine its utility for other domains. Given that we did not tune β, it is also possible that performance could improve if tuned.

5 Conclusion

In this paper, we made several advances in the understanding of TD-learning algorithms in which state-value functions are learned in tandem with action-value functions. We showed that on-policy QV-learning algorithms have benefits for prediction, as in the original setting for which they were proposed, but tend to be much slower when the target policy is greedy. We also demonstrated that the prevailing off-policy control variant of QV-learning, QVMAX, is biased, and we introduced a new

variant called BC-QVMAX which empirically restores convergence. Lastly, we introduced a novel Dueling Q-learning method called RDQ, which does not require the subtraction of an identifiability term, and has desirable properties in terms of the geometry of the learned value functions. In a deep RL setting based on the MinAtar games, we showed that RDQ greatly outperforms Dueling DQN, despite having identical network architectures and the same, well-tuned hyperparameters. Although there is still much to learn about these algorithms, our analysis helps to clarify their efficacy and distinguishing properties, and has already demonstrated potential for the development of new and effective RL algorithms.

References

Leemon C. Baird. Advantage updating. Technical report, Wright-Patterson Air Force Base, 1993.

Richard Bellman. Dynamic programming. *science*, 153(3731):34–37, 1966.

Dimitri Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

Johan Samir Obando Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *International Conference on Machine Learning*, pp. 1373–1383. PMLR, 2021.

George H. John. When the best move isn't optimal: Q-learning with exploration. In *AAAI*, volume 1464, 1994.

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

Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–50. Springer, 2002.

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):529–533, 2015.

Joseph Modayil and Zaheer Abbas. Towards model-free rl algorithms that scale well with unstructured data. *arXiv preprint arXiv:2311.02215*, 2023.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Gavin A. Rummery and Mahesan Niranjan. On-line Q-Learning using connectionist systems. Technical report, University of Cambridge, 1994.

Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco A Wiering. The Deep Quality-Value Family of Deep Reinforcement Learning Algorithms. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2020.

Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38:287–308, 2000.

Richard S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3:9–44, 1988.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2018.

Yunhao Tang, Rémi Munos, Mark Rowland, and Michal Valko. Va-learning as a more efficient alternative to q-learning. In *International Conference on Machine Learning*, pp. 33739–33757. PMLR, 2023.

Hado van Hasselt. Double Q-learning. *Advances in neural information processing systems*, 23, 2010.

Hado van Hasselt. *Insights in reinforcement learning*. PhD thesis, 2011.

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

Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

Marco A Wiering. QV(λ)-learning: A New On-policy Reinforcement Learning Algorithm. In *Proceedings of the 7th European Workshop on Reinforcement Learning*, volume 7. Univ. Naples Dep. Math. Stat. Naples, Italy, 2005.

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

Kenny Young and Tian Tian. Minatar: An Atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

Supplementary Materials

The following content was not necessarily subject to peer review.

A Proofs

Theorem 3.1 (QV-learning contraction)**.** *The expected QV-learning update corresponds to a linear joint operator* $H\colon [\begin{smallmatrix}\boldsymbol{v}\\ \boldsymbol{q}\end{smallmatrix}] \mapsto \boldsymbol{b} + \boldsymbol{A}[\begin{smallmatrix}\boldsymbol{v}\\ \boldsymbol{q}\end{smallmatrix}]$*, where* $\boldsymbol{b} = [\begin{smallmatrix}\boldsymbol{P}_\pi \boldsymbol{r}\\ \boldsymbol{r}\end{smallmatrix}]$ *and* $A = \left[\begin{smallmatrix}\gamma P_\pi P_p & \mathbf{0}\\ \gamma P_p & \mathbf{0}\end{smallmatrix}\right]$*. The operator* H *is a contraction mapping with its unique fixed point equal to* $[\begin{smallmatrix}\boldsymbol{v}_\pi\\ \boldsymbol{q}_\pi\end{smallmatrix}]$*.*

Define the following partial transition operators:

$$(\boldsymbol{P}_p \boldsymbol{v})(s,a) = \sum_{s' \in \mathcal{S}} p(s'|s,a)\boldsymbol{v}(s) \tag{25}$$

$$(\boldsymbol{P}_\pi \boldsymbol{q})(s) = \sum_{a \in \mathcal{A}} \pi(a|s)\boldsymbol{q}(s,a) \tag{26}$$

Bellman equation for $\boldsymbol{v}_\pi$:

$$\boldsymbol{v}_\pi = P_\pi(\boldsymbol{r} + \gamma P_p \boldsymbol{v}_\pi) \tag{27}$$

Bellman equation for $\boldsymbol{q}_\pi$:

$$\boldsymbol{q}_\pi = \boldsymbol{r} + \gamma P_p P_\pi \boldsymbol{q}_\pi \tag{28}$$

Proof. We need to jointly solve these equations:

$$\boldsymbol{v} = \boldsymbol{P}_\pi(\boldsymbol{r} + \gamma \boldsymbol{P}_p \boldsymbol{v}) \tag{29}$$

$$\boldsymbol{q} = \boldsymbol{r} + \gamma \boldsymbol{P}_p \boldsymbol{v} \tag{30}$$

The fixed points of these equations are individually $\boldsymbol{v}_\pi$ and $\boldsymbol{q}_\pi$. For a block matrix $\boldsymbol{M} = [\begin{smallmatrix}\boldsymbol{A} & \boldsymbol{B}\\ \boldsymbol{C} & \boldsymbol{D}\end{smallmatrix}]$ with $\boldsymbol{B} = \mathbf{0}$, the matrix inversion is

$$\boldsymbol{M}^{-1} = \begin{bmatrix} A^{-1} & \mathbf{0} \\ -\boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{A}^{-1} & \boldsymbol{D}^{-1} \end{bmatrix} \tag{31}$$

which gives us

$$(I - A)^{-1} = \begin{bmatrix} (I - \gamma P_\pi P_p)^{-1} & \mathbf{0} \\ \gamma \boldsymbol{P}_p (I - \gamma P_\pi P_p)^{-1} & \boldsymbol{I} \end{bmatrix} \tag{32}$$

and thus the fixed point is

$$\begin{aligned}(\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{b} &= \begin{bmatrix} (I - \gamma P_\pi P_p)^{-1} P_\pi \boldsymbol{r} \\ \boldsymbol{r} + \gamma \boldsymbol{P}_p (I - \gamma P_\pi P_p)^{-1} P_\pi \boldsymbol{r} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{v}_\pi \\ \boldsymbol{r} + \gamma \boldsymbol{P}_p \boldsymbol{v}_\pi \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_\pi \\ \boldsymbol{q}_\pi \end{bmatrix}.\end{aligned}$$

The Bellman equations are simultaneously satisfied when

$$\begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{q} \end{bmatrix} = \begin{bmatrix} \boldsymbol{P}_\pi \boldsymbol{r} \\ \boldsymbol{r} \end{bmatrix} + \begin{bmatrix} \gamma \boldsymbol{P}_\pi \boldsymbol{P}_p & \mathbf{0} \\ \gamma \boldsymbol{P}_p & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{q} \end{bmatrix}. \tag{33}$$

Letting $Y = [\begin{smallmatrix}\boldsymbol{v}\\ \boldsymbol{q}\end{smallmatrix}]$ be a joint value function, i.e., the concatenation of state- and action-value functions, then Eq. (33) has the simple linear form of $\boldsymbol{y} = \boldsymbol{b} + \boldsymbol{A}\boldsymbol{y}$, where $\boldsymbol{b} = 0$ and $\boldsymbol{A} = 0$.

Finally, we can see that $\|A\|_\infty = \gamma$ because each row is a probability distribution scaled by γ. For any two joint value functions y, y', we therefore have

$$\|Hy - Hy'\|_\infty = \|Ay - Ay'\| \leq \gamma \|y - y'\|,$$

and the operator H is a contraction mapping. By the Banach fixed-point theorem, we have $\lim_{i\to\infty} H^i y = [\begin{smallmatrix}\boldsymbol{v}_\pi\\ \boldsymbol{q}_\pi\end{smallmatrix}]$ for any initial vector y, which completes the proof. □