university of groningen / faculty of science and engineering

Addressing Bootstrapping Errors in Offline Reinforcement Learning with Ensembles

Bachelor's Project Thesis

Marco Andrea Gallo, s3680622, m.a.gallo@student.rug.nl
Supervisor: Dr. Matthia Sabatelli

Abstract: Interest in Reinforcement Learning has surged in recent years on pair with its success stories. Nonetheless, deployment of Reinforcement Learning systems to real-world applications is still not on the scale of standard supervised learning models, which are able to exploit vast offline datasets. Through algorithms that can learn from data collected by other policies, off-policy Reinforcement Learning aims to improve the low sample efficiency of standard online algorithms and to better exploit existing offline datasets. One key challenge for off-policy value-based algorithms is the bootstrapping error (Kumar et al., 2019), where actions outside of the training data distribution incorrectly influence policy optimization. This error is exacerbated in the offline setting, and common solutions pertaining to uncertainty-based methods focus on bootstrap ensembles. This research seeks to assess whether the DQV algorithmic family (Sabatelli et al., 2020) benefits from the simple ensemble technique of Ensemble-DQN (Agarwal et al., 2020) for bootstrapping error control. Empirical studies are performed on two classic control OpenAI Gym environments, tracking the algorithms' accumulated reward and value estimates evolution during evaluation. Preliminary results found offline DQV and DQV-Max robust to bootstrapping errors due to their particular temporal difference updates. The proposed ensemble technique confirmed moderate bootstrapping error correction for offline DQN on one environment, yet no significant advantage was found for the DQV family, suggesting that these deep off-policy algorithms are already strong in the offline setting.

1 Introduction

In the past decade, machine learning methods have encountered major success over a wide number of real-world applications, ranging from Computer Vision to Natural Language Processing tasks. Much of the progress in these areas can be attributed to the development of *data-driven*, scalable learning methods. In fact, although advancements in models and architectures are an integral part of this success story, the learning techniques these models employ are well-founded and understood, and major improvements in model performance stem from the availability of large and diverse training datasets. Such data-driven methods do not map to the Reinforcement Learning (RL) framework equally well. RL involves sequential decision making problems where the best behavior strategy is learned through active interaction with the environment. This naturally *online* learning paradigm prevents effective exploitation of the rich *offline* datasets that make supervised learning methods so powerful. Moreover, data collection for complex real-world RL applications such as autonomous driving or healthcare support systems can often be expensive or hazardous. Therefore, developing safe and capable offline RL agents has a great appeal: by efficiently learning from large amounts of data, we could create "generalizable and powerful *decision making engines*" (Levine et al., 2020) to aid solving many real-world open problems.

In offline RL, an agent learns a policy from a static dataset of logged experiences produced by a *behavior policy*. This differs from classical online RL, where an agent can actively collect new experience by interacting with its environment (Sutton & Barto, 2018). For an offline agent to improve, it is therefore crucial that it is able to partake

in counterfactual reasoning to accurately estimate the outcomes of a decision different from the corresponding one in the training dataset (Levine et al., 2020). By contrast, an online RL agent could explore and learn on its own the effects of a decision different than the one previously chosen in the same situation. This additional constraint of offline RL is problematic, and it exceeds the capabilities of current machine learning methods that use expressive function approximators (neural networks) to generalize across examples. For one, it violates the assumption of independent and identically distributed data (i.i.d.) that standard supervised learning algorithms rely on: an offline RL agent may be trained under one distribution but tested under a different one. In addition, an offline agent must be able to reason differently from the data-generating policy in order to produce novel – and possibly favorable – courses of actions; this requirement breaks the i.i.d. assumption too. The mismatch between the behavior policy-induced distribution and the one learned during training is called *distributional shift*, and it presents a fundamental challenge for the efficacy of offline RL (Levine et al., 2020).

Distributional shift generally affects off-policy RL algorithms. These are methods that learn about a *target policy* using a different *behavior policy* (Sutton & Barto, 2018), such as the popular Q-learning agent (Watkins & Dayan, 1992). Due to their ability to learn from data generated by another policy, off-policy algorithms naturally lend themselves to offline RL. It is well known that off-policy methods exhibit high estimates variance (Sutton & Barto, 2018); additionally, off-policy algorithms which employ a maximization operation in the bootstrapping step, such as Q-learning, are prone to overoptimistic value estimates (Thrun & Schwartz, 1993). In the offline setting, this form of *bootstrapping error* results in the selection of actions that lie outside of the training data distribution (Kumar et al., 2019) which disrupt the training process and drive it towards regions of uncertainty.

To address distributional shift and bootstrapping errors in off-policy learning, many techniques found in the literature employ ensemble-based methods (Osband et al., 2016; Anschel et al., 2017; Fujimoto et al., 2019, appendix D.2). Most relevant for this research, ensembling methods have demonstrated a successful approach in offline RL, as seen in the REM agent by Agarwal et al. (2020). The present paper investigates to which extent results concerning bootstrapping error prevention based on ensemble variants of Q-learning transfer to other model-free, value-based RL algorithms in the context of offline RL. In particular, we will inquire whether benefits concerning bootstrapping error reduction stemming from simple ensemble methods apply to the deep RL agents of the DQV algorithmic family (Sabatelli et al., 2020).

1.1 Background

The following subsections will introduce definitions and prior knowledge needed to understand this paper.

1.1.1 Reinforcement Learning

Reinforcement Learning seeks to solve a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, p, R)$. In RL, an agent interacts with an environment at discrete time steps $t = 0, 1, 2, 3, \ldots$. At each time step t, the agent receives a representation of the environment $s_t \in \mathcal{S}$, on which it can perform some action $a_t \in \mathcal{A}(s)$ that makes it transition to a new state s_{t+1} according to a dynamics model $p\colon \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1], p(s' \mid s, a) \doteq \Pr\{s' = s_{t+1} \mid s = s_t, a = a_t\}$. At s_{t+1}, the agent receives reward r_t from a reward function $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}, r_t = R(s_t, a_t)$. The goal of a RL agent is then to find a mapping from states to action probabilities, called a *policy* $\pi\colon \mathcal{S} \times \mathcal{A} \to [0, 1], \pi(a \mid s) = \Pr\{a_t = a, s_t = s\}$, which maximizes the *expected return* $G_t \doteq \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)$ where $\gamma \in [0, 1]$ is a discount factor that scales the importance of future rewards. Each policy π has a matching *state value function* $V^\pi(s) = \mathbb{E}[G_t \mid s = s_t]$, which indicates the expected return obtained starting from state s therefore following π. This function can also be expressed in terms of state-action pairs as a *state-action value function* $Q^\pi(s, a) = \mathbb{E}[G_t | s = s_t, a = a_t]$, indicating the expected return taking action a in state s and consequently following π. Altogether, the RL optimization problem aims to achieve a policy π^* characterized by the *optimal Q value function* $Q^*(s, a) \doteq \max_\pi Q^\pi(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, whose solution is provided by the Bell-

man optimality equation

$$Q^*(s_t, a_t) = \mathbb{E}\Big[R(s_t, a_t) + \gamma \max_{a \in \mathcal{A}} Q^*(s_{t+1}, a) \,\Big|\, s_t = s, a_t = a\Big] \tag{1.1}$$

(Bellman, 1957). Note that the latter can also be expressed as the *optimal state value function* $V^*(s)$ by replacing the optimal Q value estimate at the next state $\max_{a \in \mathcal{A}} Q^*(s_{t+1}, a)$ with $V^*(s_{t+1})$.

$Q^*(s, a)$ and $V^*(s)$ can both be learned by Temporal Difference (TD) learning (Sutton, 1988), and Q-learning is the most popular TD method; it learns the state-action value function using the update rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[y_{\text{QL}}^{\text{TD}} - Q(s_t, a_t)\right], \tag{1.2}$$

where

$$y_{\text{QL}}^{\text{TD}} = R(s_t, a_t) + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) \tag{1.3}$$

(Watkins & Dayan, 1992). When the state space $\mathcal{S}$ is large and high-dimensional the Q function is approximated with deep neural networks, hence the name of Deep Reinforcement Learning (DRL). DRL agents such as Deep Q-Learning (DQN) (Mnih et al., 2013) have attained super-human performance on a range of complex tasks such as the ALE benchmark suite (Bellemare et al., 2013). DRL algorithms generally adapt the Q function to include a neural network parameterized by θ, and reformulate the standard Q-learning update rule to a differentiable loss function

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim \mathcal{D}} \left[\left(y_{\text{DQN}}^{\text{TD}} - Q(s_t, a_t; \theta)\right)^2\right] \tag{1.4}$$

where

$$y_{\text{DQN}}^{\text{TD}} = r_t + \gamma \max_{a \in \mathcal{A}} Q\left(s_{t+1}, a; \theta^-\right), \tag{1.5}$$

$\mathcal{D}$ is the Experience Replay buffer (Lin, 1992) used to store and sample trajectories during training, and θ^- are the parameters of a frozen target network commonly used to stabilize value estimates. The use of this different set of parameters is conceptually related to the bootstrapping error, as further explained in Section 1.1.2.

Finally, in *offline* RL – also known as *batch* RL – the standard MDP formulation remains valid, but the agent loses the ability to transition from state s_t to state s_{t+1} by *actively* choosing and performing action a_t. Instead, an offline RL agent is given a logged dataset $\mathcal{B}$ of experience tuples $\langle s_t, a_t, s_{t+1}, r_t \rangle$ generated by a *behavior policy* π_β, and its task is to learn a (possibly better) policy than π_β from these trajectories. Since learning occurs under a state-action distribution induced by a policy different from the current one, offline RL is also known as *fully off-policy* RL, and an offline agent needs to maximize data exploitation because it lacks the possibility to explore.

1.1.2 The Off-Policy Bootstrapping Error

Off-policy RL is systematically afflicted by a source of error denoted as *extrapolation error* (Fujimoto et al., 2019). Due to a mismatch between the state-action distribution induced by the current policy and the one contained in the experience replay batch, the Q function is unable to correctly estimate the value of unseen state-action pairs. As a result, such inputs receive inflated estimates which skew the Q function, and possibly cause it to diverge. Fujimoto et al. (2019) remark that, when combined with RL algorithms which employ a maximization operator to compute $Q^*(s, a)$ like Q-learning, the extrapolation error induces a consistent positive *overestimation bias* (Thrun & Schwartz, 1993) in the Q function.

The *bootstrapping error* (Kumar et al., 2019) is a form of extrapolation error which appears in algorithms that bootstrap to compute their targets. These algorithms create the true target of a regression problem using their own current estimate of such target, which is a biased estimator. Referring to update rules 1.3 and 1.5, it is clear that Q-learning-based algorithms are prone to the bootstrapping error since their respective TD-targets both come from a present estimate of Q. To stabilize the Q function's recursive regression, DRL algorithms form their target estimates from another static estimator, simulated by keeping a copy of parameters θ^- frozen and only updating them at intervals. Moreover, since the TD targets are arbitrarily wrong during training, maximizing the Q-values with respect to actions at the next state as in Equation 1.5 might evaluate the Q function on actions that do not correspond to the training data distribution. Such out-of-distribution

(OOD) actions (Kumar et al., 2019) are not contained in the training batch, and their true value is unknown; a naive maximization will then pick the overoptimistic Q-values, therefore compounding and propagating the bootstrapping error during training through Bellman backups (Equation 1.2). In the most extreme case where the Q function is initialized with high positive values only at OOD actions, a Q-learning based agent will thus learn to perform these very actions and disregard information gathered from the behavior policy π_β.

The bootstrapping error is especially detrimental in offline RL, where no additional data collection is possible. In the online case, the wrong estimation of $Q(s,a)$ for some (s,a) pair can be adjusted by actually performing a in s and assessing its result. However, the dataset $\mathcal{B}$ used by an offline RL agent is fixed and it does not allow for further exploration. Dealing with bootstrapping errors is therefore crucial for the success of offline RL algorithms.

1.1.3 Bootstrapping Error Correction

In offline reinforcement learning, techniques to correct bootstrapping errors explicitly, or implicitly by minimizing distributional shift, involve either *policy constraint* or *uncertainty-based* methods (Levine et al., 2020).

The Q function is evaluated on the same states that it is trained on. Therefore, only the action inputs across states can be out of distribution in the training process. *Policy constraint* methods address this issue by bounding the distribution over actions used for the computation of the TD-targets, $\pi(a'|s')$, to stay in the proximity of the one induced by the behavior policy, $\pi_\beta(a'|s')$. In this way, the Q function regression is driven by target values for which enough reliable information is found in π_β. The difference between these techniques resides in the metrics they employ to define distributional proximity. For example, Batch-Constrained deep Q-Learning (BCQ) (Fujimoto et al., 2019) trains a generative model – a variational auto-encoder (VAE) (Kingma & Welling, 2013) – to produce actions which are likely given the data in $\mathcal{B}$ and are then used to compute the TD-targets. By substituting the standard TD-target maximum over all possible actions at the next state with the maximum over actions likely under π_β, BCQ ensures that the learned policy π is centered around π_β and that the Q function is not queried on OOD actions which cause bootstrapping errors. Bootstrapping Error Accumulation Reduction Q-Learning (BEAR-QL) (Kumar et al., 2019) also follows the intuition of placing constraints on the learned action distribution, but it achieves so with fewer restrictions than BCQ. This makes it more viable for π to improve on π_β, a capability hindered by the tight constraint of BCQ. Instead of requiring the learned policy to be close in distribution to π_β, BEAR demands a *support constraint* (Kumar, Aviral, 2019). This loose condition means that the learned policy must place non-zero probability on all those actions that have non-negligible probability according to the behavior policy. With this precaution BEAR is able to improve over suboptimal, even random off-policy trajectories, where BCQ would instead learn a policy close to uniform (Kumar et al., 2019).

By contrast, *uncertainty-based* methods do not aim to restrict the learned policy to a safe region; rather, they rely on estimating epistemic uncertainty in the Q function and integrating this information in the computation of target values. This means learning an uncertainty distribution over Q functions as induced by the offline dataset $\mathcal{B}$, denoted $\mathcal{P}_\mathcal{B}(Q^\pi)$. When this is known, a penalty term of the form $-\alpha\mathrm{Unc}(\mathcal{P}_\mathcal{B}(Q^\pi))$ can be added to the TD-targets in order to produce a *conservative* estimate of the actual Q function that, ideally, is proportional to the model's confidence in the data. Since OOD actions are outside of the training data distribution, they should naturally have large uncertainty estimates and result in desirably conservative Q-values (Levine et al., 2020).

One common way of learning $\mathcal{P}_\mathcal{B}(Q^\pi)$ is to use bootstrap ensembles (Osband et al., 2016; Kumar et al., 2019; Agarwal et al., 2020). A Q function ensemble trains multiple Q_i functions on samples from $\mathcal{B}$ drawn with replacement, then it compounds the different $Q_i(s,a)$ predictions – typically by averaging in a regression problem – to obtain the final population prediction $Q(s,a)$. It is known that ensemble methods help stabilizing highly unstable prediction procedures (Breiman, 1996); in fact, simply using K approximators to estimate Q in DQN yields a K-fold variance reduction, with improved accuracy in TD-targets estimation and diminished overestimation bias (Anschel et al., 2017). When estimating uncertainty using ensembles, one common choice of 'Unc' is the variance across the ensemble

Q-value predictions (Kumar et al., 2019). Random Ensemble Mixture (REM) (Agarwal et al., 2020), a strong off-policy algorithm based on DQN, obtained state-of-the-art results in discrete and continuous domain offline RL using Q function ensembles. As a measure of uncertainty, REM employs a convex weighted sum of each Q function's estimate which is minimized globally by the ensemble.

2 Methods

The preliminary experiments in this research aimed at determining the presence of offline bootstrapping errors in DQV and DQV-Max, the two algorithms in the DQV algorithmic family (Sabatelli et al., 2020). Subsequently, in line with the research question, a second experiment sought to assess the impact of value function ensembles on the susceptibility of these algorithms to the offline bootstrapping error.

2.1 DQV and DQV-Max

DQV and DQV-Max were chosen because they are model-free, value-based deep RL algorithms like DQN, suitable for learning on discrete domains. Moreover, differently from DQN, both algorithms add a state-value function V to the optimization problem to obtain more robust value estimates and be less prone to the overestimation bias (Sabatelli et al., 2020). Given two neural networks $Q(s, a; \theta)$ and $V(s; \phi)$ with corresponding target parameters θ^- and ϕ^-, and trajectory batches sampled from the Experience Replay Buffer $\mathcal{D}$, the objective functions used by DQV become

$$\mathcal{L}(\phi) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim \mathcal{D}} \left[\left(y^{\text{TD}}_{\text{DQV}} - V(s_t; \phi) \right)^2 \right] \tag{2.1}$$

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim \mathcal{D}} \left[\left(y^{\text{TD}}_{\text{DQV}} - Q(s_t, a_t; \theta) \right)^2 \right], \tag{2.2}$$

where

$$y^{\text{TD}}_{\text{DQV}} = r_t + \gamma V\left(s_{t+1}; \phi^-\right). \tag{2.3}$$

Due to the lack of a maximization operation in the TD-target $y^{\text{TD}}_{\text{DQV}}$, DQV is an *on-policy* algorithm; moreover, it should be noted that both the Q and V function of DQV learn from the same TD-target computed by V. Conversely, the objective functions for DQV-Max are as follows:

$$\mathcal{L}(\phi) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim \mathcal{D}} \left[\left(v^{\text{TD}}_{\text{DQV-Max}} - V(s_t; \phi) \right)^2 \right] \tag{2.4}$$

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim \mathcal{D}} \left[\left(q^{\text{TD}}_{\text{DQV-Max}} - Q(s_t, a_t; \theta) \right)^2 \right] \tag{2.5}$$

where

$$v^{\text{TD}}_{\text{DQV-Max}} = r_t + \gamma \max_{a \in \mathcal{A}} Q\left(s_{t+1}, a; \theta^-\right) \tag{2.6}$$

$$q^{\text{TD}}_{\text{DQV-Max}} = r_t + \gamma V\left(s_{t+1}; \phi\right). \tag{2.7}$$

As seen in TD-target Equation 2.6 – equal to the DQN TD-target Equation 1.5 – DQV-Max is an off-policy algorithm, so of greater interest for offline RL. In addition, DQV-Max uses two different temporal difference targets, where the one used to learn the state-action value function resembles the TD-target for DQV but without employing a target network.

2.2 Common experimental details

All experiments are conducted on the `CartPole-v1` and `Acrobot-v1` OpenAI Gym environments (Brockman et al., 2016), two pole-balancing classic control problem well studied in the RL literature. Both environments provide continuous state representations $s \in \mathbb{R}^n$ and discrete action spaces, suitable for approximate dynamic programming (i.e. value-based) methods such as Q-learning. The algorithms involved in the experiments are implemented using Google's JAX machine learning library (Bradbury et al., 2018), then trained and evaluated under standard DRL neural networks architectures, hyper-parameters and preprocessing settings following the Dopamine reinforcement learning framework (Castro et al., 2018); see Table A.1 for the full hyper-parameters table.

Due to the offline nature of the proposed experiments, the first common step was online data collection. We replicated the data collection process employed by Agarwal et al. (2020) for the DQN Replay Dataset on the ALE environments, adapting it to the two proposed problems. A behavioral DQN agent was trained online on each environment for a total of approximately 500 000 steps (500 iterations of at least 1000 steps), starting to fit the Q function

after experiencing 500 trajectories, then performing a gradient update every 4 steps. Every trajectory observed by the behavioral agent during training was logged in order to gather a dataset of significant size and diverse policy composition, vital for the success of offline RL (Agarwal et al., 2020). This process was repeated across 3 random seed initializations to control for volatility due to stochasticity. In the offline experiments, each run was paired with one of these logged datasets, such that the reported response metrics are averages over 3 redundancies.

2.3 Bootstrapping error in offline DQV and DQV-Max

In order to detect the bootstrapping error, we need to track the evolution of the Q estimates as a function of training steps – a proxy for the number of Bellman backups. If the bootstrapping error occurs, it will cause an overestimation bias in the Q function, and the agents will inflate the expected discounted return G_t it believes it will gain starting from state s_t. Since the chosen environments provide constant reward r at each time-step t until the enforced episode termination at time M, it is straightforward to compute the baseline discounted return expected from state t as $G_t = \sum_{k=t}^{M} \gamma^k r$. Focusing on the first state s_0 of each new episode as given by $\max_{a \in \mathcal{A}} Q(s_0, a; \theta)$, the evaluation-time progression of Q values was recorded. It should be noted that one evaluation iteration of 1000 time-steps occurred every 5 training iterations; by interleaving training and testing, we are still able to analyze the Q estimates evolution as a temporal sequence, and to assess the effects of bootstrapping error accumulation. Results are presented in Figure 3.1. Offline DQN is used as a baseline; the full lines correspond to each agent's Q estimates at evaluation time, while the dashed line is the environment's actual return G_t at an episode's first state s_0.

2.4 Ensemble DQV and DQV-Max

To investigate the efficacy of ensembling methods for bootstrapping error prevention, an averaging ensemble technique inspired by Ensemble-DQN (Agarwal et al., 2020) was implemented. In Ensemble-DQN, the Q function is approximated by an ensemble of K heads parameterized by a weight vector θ, with corresponding target weights θ^-; each Q-value prediction is then optimized with respect to its own target, similarly to Bootstrapped-DQN (Osband et al., 2016). Each head is initialized with different parameters and trained on all data using identical mini-batches; although bootstrap ensembles actually require that each head is trained on a different sample drawn with replacement from the dataset $\mathcal{D}$, it is well known in the deep learning literature that initializing a model with different parameters provides enough diversity to obtain reliable uncertainty estimates (Osband et al., 2016; Levine et al., 2020). Finally, each head optimizes a global loss which is the average of the ensemble total loss; the objective function for Ensemble-DQN thus takes the form

$$\begin{aligned} \mathcal{L}(\theta) = & \frac{1}{K} \sum_{k=0}^{k-1} \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim \mathcal{D}} \\ & \left[\left(y^{\text{TD}}_{\text{Ens-DQN}} - Q(s_t, a_t; \theta_k) \right)^2 \right], \end{aligned} \tag{2.8}$$

where

$$y^{\text{TD}}_{\text{Ens-DQN}} = r_t + \gamma \max_{a \in \mathcal{A}} Q\left(s_{t+1}, a; \theta^-_k\right). \tag{2.9}$$

When translating the ensemble loss objective to DQV and DQV-Max, we decided to use ensembles for the TD-targets computation. The rationale for this choice is to mitigate the bootstrapping error occurring precisely at this step; we expect that the compound estimate computed by the ensemble is close to the true TD-target $\hat{y}$. As a result, Ensemble-DQV employs only one ensemble on the V function to compute the common TD-target; by contrast, since DQV-Max requires two different TD-targets, Ensemble-DQV-Max uses ensembles to estimate both the Q and V function, yet the V heads lack target networks as in the original DQV-Max algorithm. The modified objectives of Ensemble-DQV and Ensemble-DQV-Max respectively become

$$\begin{aligned} \mathcal{L}(\phi) = & \frac{1}{K} \sum_{k=0}^{k-1} \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim \mathcal{D}} \\ & \left[\left(y^{\text{TD}}_{\text{Ens-DQV}} - V(s_t; \phi_k) \right)^2 \right] \end{aligned} \tag{2.10}$$

$$\begin{aligned} \mathcal{L}(\theta) = & \frac{1}{K} \sum_{k=0}^{k-1} \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim \mathcal{D}} \\ & \left[\left(y^{\text{TD}}_{\text{Ens-DQV}} - Q(s_t, a_t; \theta) \right)^2 \right], \end{aligned} \tag{2.11}$$

where

$$y^{\text{TD}}_{\text{Ens-DQV}} = r_t + \gamma V\left(s_{t+1}; \phi^-_k\right) \tag{2.12}$$

and

$$\begin{aligned}\mathcal{L}(\phi) = &\frac{1}{K}\sum_{k=0}^{k-1}\mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle\sim\mathcal{D}} \\ &\left[\left(v^{\text{TD}}_{\text{Ens-DQV-Max}} - V(s_t;\phi_k)\right)^2\right]\end{aligned} \tag{2.13}$$

$$\begin{aligned}\mathcal{L}(\theta) = &\frac{1}{K}\sum_{k=0}^{k-1}\mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle\sim\mathcal{D}} \\ &\left[\left(q^{\text{TD}}_{\text{Ens-DQV-Max}} - Q(s_t, a_t;\theta_k)\right)^2\right],\end{aligned} \tag{2.14}$$

where

$$v^{\text{TD}}_{\text{Ens-DQV-Max}} = r_t + \gamma \max_{a\in\mathcal{A}} Q\left(s_{t+1}, a; \theta^-_k\right) \tag{2.15}$$

$$q^{\text{TD}}_{\text{Ens-DQV-Max}} = r_t + \gamma V(s_{t+1};\phi_k)\,. \tag{2.16}$$

The ensembles were implemented using a multi-head architecture: each head shares the same body of layers except for a final fully connected layer, initialized with different parameters across heads; this architecture is also employed by REM (Agarwal et al., 2020). The experiments concerning the ensemble version of DQV and DQV-Max follow the same setup outlined in Section 2.3, however in this case $\max_{a\in\mathcal{A}} \frac{1}{K}\sum_{k=0}^{k-1} Q(s_0, a; \theta_k)$ was recorded to track the evolution of value estimates. For each experiment on the aforementioned environments a number of heads $K = 4$ was used; although other examples in the literature use a greater number of heads (e.g. $K = 10$ for Bootstrapped-DQN (Osband et al., 2016)), we settled on 4 due to computational limitations. Results are presented in Figure 3.2, where offline Ensemble-DQN is used as the baseline.

3 Results

The learning curves in Figure 3.1 show the Q-values evolution for the standard agents as a result of offline training, while the ones in Figure 3.2 show the same metric for their respective ensemble version. Every proposed offline agent learns to solve both classic control environments, and the reward curves are presented in Appendix A.1. The reward signals for the `CartPole-v1` environment are generally noisier compared to the ones for `Acrobot-v1`; this is in line with the performance of the online behavioral DQN agent π_{DQN}, which also produced unstable learning curves on this problem.

3.1 Offline bootstrapping error in the DQV family

As seen in Figure 3.1, the experiments confirmed previous findings for Q-learning-based agents run offline on continuous domain problems (Fujimoto et al., 2019; Kumar et al., 2019): the Q function incurs in a heavy overestimation bias produced by bootstrapping errors. This is most evident on the `CartPole-v1` environment, where DQN's Q-value estimates quickly escalate above the true value for s_0. Since the offline agent has no access to ground truth values due to lack of exploration, it cannot adjust the Q function estimates during training and the whole estimation process diverges. Offline DQN suffers overestimation on the `Acrobot-v1` problem too, but no divergence in Q estimates is observed here; this might be because the behavioral data for this environment are less noisy compared to those of `CartPole-v1`.

Among the studied algorithms, offline DQV is the most robust one to the bootstrapping error. On the `CartPole-v1` environment it is almost able to correctly estimate the true value of s_0 for each episode, never incurring in overoptimistic estimates. However, on the `Acrobot-v1` problem, offline DQV still suffers from stable overestimation, despite coming closest to the true value of s_0. DQV avoids the bootstrapping error because it is an *on-policy* algorithm. Although theoretically it should not be able to learn in the offline setting, its strong performance compared to the other agents is probably due to efficient usage of the large offline dataset, which enables it to learn on-policy discovering effective behaviors in the data. Moreover, DQV forms its TD-target using only the state-value function V, therefore it cannot possibly base predictions on those very out-of-distribution actions which are responsible for bootstrapping errors.

Offline DQV-Max is also more resilient to the bootstrapping error than offline DQN. On the `CartPole-v1` environment, it estimates the true value for s_0 nearly perfectly, showing no detrimental effects due to misaligned bootstrap estimates.

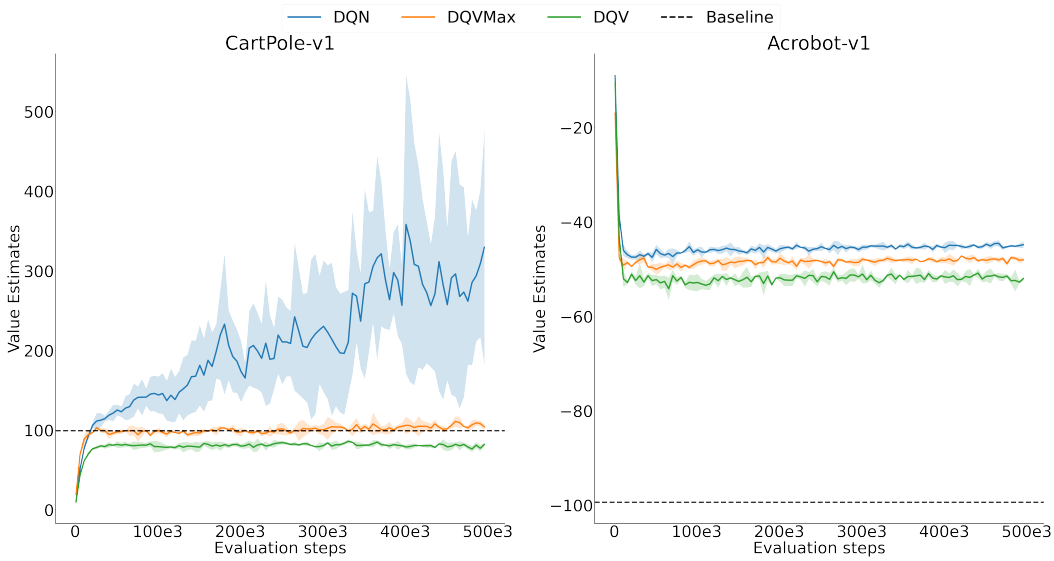

Figure 3.1: The Q-value estimates of offline DQN, DQV and DQV-Max at evaluation time. The shaded areas are ± 1 standard deviation from the mean of 3 different simulations.

As it is the case for offline DQV and DQN, it still overestimates the real Q-value for s_0 on the `Acrobot-v1` problem, positioning in between the estimates of DQN and DQV. The low Q-values on the `CartPole-v1` environment are most likely in virtue of DQV-Max's decoupling of *selection* and *evaluation* (Van Hasselt et al., 2016). DQV-Max forms its temporal difference regression targets (selection) from a model different than the one it uses to compute value estimates (evaluation). This separation is especially important for DQV-Max's TD-targets for the V function of Equation 2.6, where evaluating out-of-distribution actions could disrupt the function's convergence to the true V^*. Sabatelli et al. (2020) note that this disentanglement makes DQV-Max less prone to the overestimation bias in the online setting, and these experiments confirm the results in the offline one.

3.2 Offline bootstrapping error on the ensemble variants

On the `CartPole-v1` environment, Offline Ensemble-DQN suffers from a milder overestimation bias than offline DQN. The Q-value estimates on this problem decreased significantly with the implementation of the ensemble strategy ($t = 7.40, p < .01$); interestingly, in line with the theoretical analysis of Anschel et al. (2017), the observed decrease in Q estimates variance was proportional to the ensemble number of heads K (8755.07 vs. 2330.90 for Ensemble-DQN and DQN, respectively). The Q-value for s_0 still diverges from the true baseline, as seen in Figure 3.2; this is symptomatic that a simple ensemble of Q functions alone does not prevent the DQN bootstrapping error. Since the behavior policy π_{DQN} produced a noisy reward signal on the `CartPole-v1` environment, the lower Q estimates compared to base DQN are actually desirable and better capture the ensemble's uncertainty about the true value of s_0. This naturally results in a significant drop in performance for Ensemble-DQN ($t = 2.62, p < .01$), showing that the agent uses the Q-values as a proxy for predicted reward; full results can be found in Table 3.1. Again, the Q estimates on the `Acrobot-v1` environment for Ensemble-DQN are stable and overoptimistic, but they do not significantly differ from those of base offline DQN.

Concerning the offline ensemble versions of DQV and DQV-Max, no significant change in value estimates from their standard counterparts were observed on both environments. The Q-values for

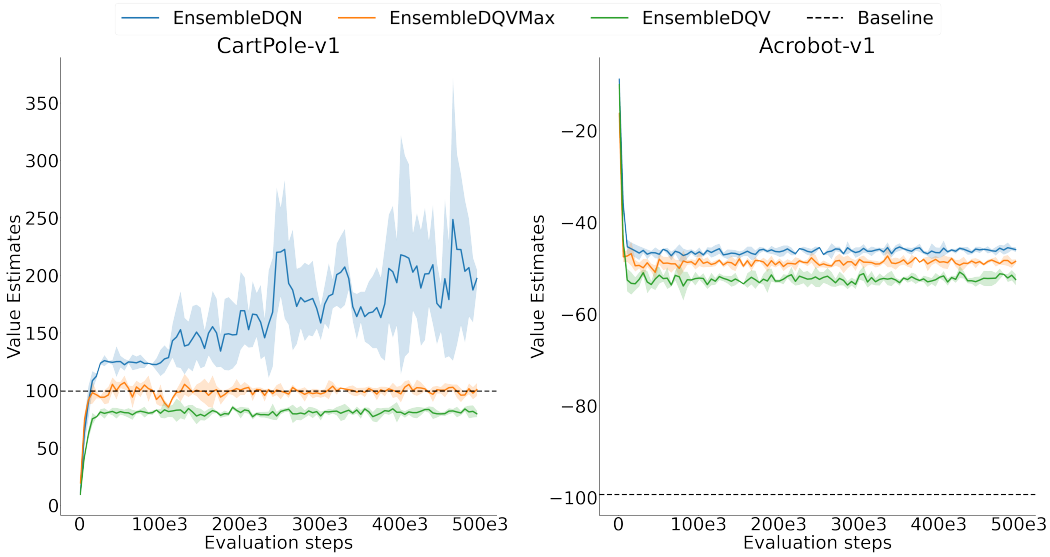

Figure 3.2: Evaluation time Q-value estimates of the ensemble version of offline DQN, DQV and DQV-Max. The shaded areas are ± 1 standard deviation from the mean of 3 different simulations.

these agents in Figure 3.2 appear very similar to the non-ensemble variants in Figure 3.1. One notable exception is the performance of offline Ensemble-DQV on the `CartPole-v1` environment, where a significant drop was registered ($t = 2.40, p < .01$). However, the Q estimates distributions for s_0 between this agent and offline DQV almost perfectly overlap as seen in Figure 3.3. Both offline DQV and Ensemble-DQV converge to basically the same Q-value for s_0, and both cannot incur in an overestimation bias since they only use the V function to compute the TD-target. Therefore, one plausible reason for this performance drop is the combination of DQV's "on-policyness" with the ensemble technique. Theoretically, on-policy algorithms should not be able to learn from off-policy data, yet offline DQV is still able to solve the problems correctly due to their relative ease. However, it is likely that the incorrect estimations derived from being on-policy compound together from each head of the ensemble, producing a sub-optimal policy. To this regard, it should be noted that the reward accumulated by Ensemble-DQV on `CartPole-v1` is lower than that of base DQV mostly in the early stages of learning (first 50 000 steps) across each run, subsequently stabilizing at the maximum for the environment.

3.3 Additional study: Ensemble-DQV-Max ablations

Since no change was found when using ensembles to estimate both Q and V in DQV-Max, two additional experiments were performed where either the Q or the V function were approximated by an ensemble, respectively. This further investigation is motivated by the fact that, like DQN, DQV-Max is an off-policy algorithm, hence of interest for offline RL. Moreover, since DQV-Max already decouples selection and evaluation, we want to assess whether either value function involved in the computation of DQV-Max's TD-targets drives more bootstrapping error than the other, based on the assumption that ensemble techniques should dampen Q-values. As seen in Figure 3.4, results for these experiments are clear: ensembling the Q function (EnsembleDQV-MaxOnQ) or the V function (EnsembleDQVMax-OnV) results in fundamentally the same Q estimates for s_0 as produced by ensembling both functions. Looking at the DQV-Max temporal difference targets for V and Q (Equation 2.6 and 2.7, respectively) the reason is evident: the Q function regresses towards targets computed by V, which cannot suffer from the action distributional shift, and updates to Q are thus in-distribution with re-

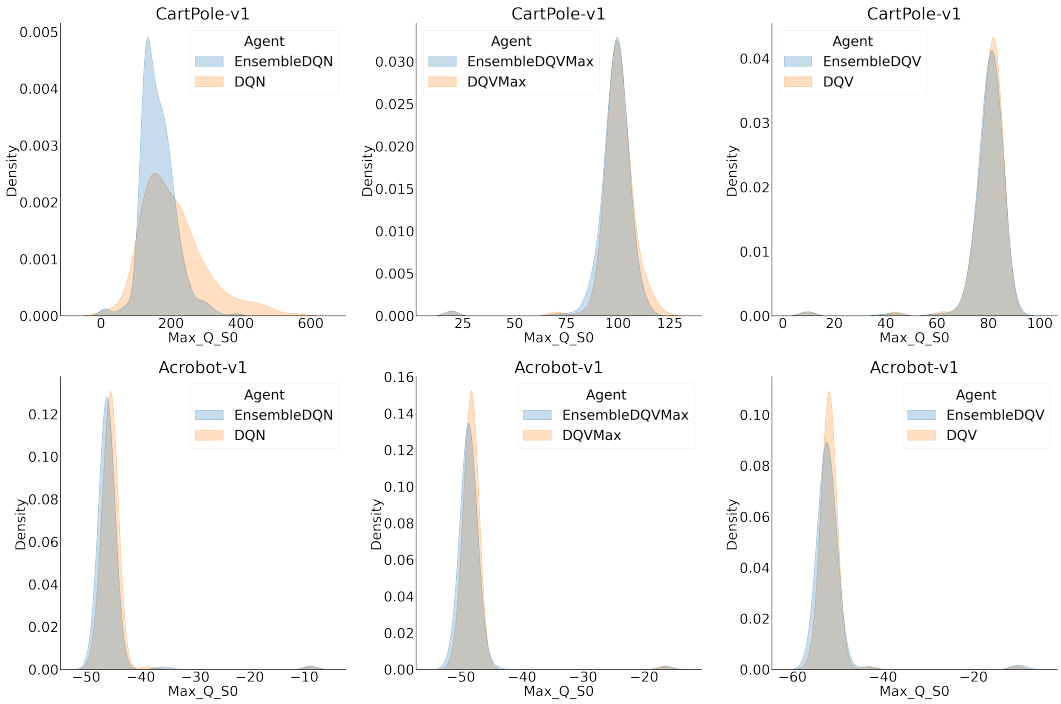

Figure 3.3: Distribution of maximum Q-values for s_0 for each agent and its ensemble variant

spect to the training data. Ultimately, this is probably enough to correct bootstrapping errors arising when computing the targets for V, and the two value functions are able to balance their respective estimates such that using an ensemble on either one results in no significant change.

4 Conclusion

The experiments presented in the previous section provide an answer to the question of whether the DQV algorithmic family benefits from ensemble techniques in offline RL. According to the empirical results, simply re-framing the learning problem of offline RL to use multiple copies of the same agent is not enough to prevent the bootstrapping error. Ensembles of function approximators are a well-known performance boost in machine learning, but it is crucial to properly exploit the additional information they provide compared to a single learner. If this point is not correctly addressed, the possible uncertainty estimation gains are overlooked, and such information is critical for an offline RL agent.

When testing whether the simple ensemble technique of Ensemble-DQN can be generalized to other algorithms, translating it to DQV and DQV-Max becomes a limitation of this study. In fact, as already mentioned, these algorithms disentangle *selection* – the choice of the regression targets for the state or state-action value function – from *evaluation* – the estimation of a state or state-action pair's value. This is one important factor in the infamous *deadly triad* of off-policy RL (Sutton & Barto, 2018), and the main focus of this research under the form of bootstrapping error. Where such decoupling is absent, as in the DQN algorithm, using a simple ensemble was in fact enough to observe gains in terms of variance reduction, which is directly related to the overestimation bias (Anschel et al., 2017) and is caused by misaligned value estimates – also known as bootstrapping errors. This was the case for the experiments with offline Ensemble-DQN on the `CartPole-v1` environment. When it comes to the DQV algorithmic family and controlling the bootstrapping error with ensembles, these algorithms' strength thus becomes

Table 3.1: Response metrics summary. Each agent is compared to its ensemble version, and colored cell-pairs highlight significant differences

Agent	CartPole-v1				Acrobot-v1			
	Reward		Q-Value		Reward		Q-Value	
	Mean	Var	Mean	Var	Mean	Var	Mean	Var
DQN	440.36	15540.87	208.71	8755.07	-70.48	2533.65	-45.22	14.57
Ensemble-DQN	410.55	23157.11	163.70	2330.90	-76.02	2949.21	-45.78	15.70
DQV	489.09	2473.61	79.68	79.68	-70.86	2032.89	-51.45	19.04
Ensemble-DQV	475.88	6657.30	79.56	81.74	-82.43	5457.53	-51.60	31.53
DQV-Max	445.77	14386.91	99.70	109.31	-69.90	1718.89	-48.06	10.45
Ensemble-DQV-Max	430.85	18011.71	98.12	98.26	-77.73	3544.01	-48.42	11.61

an architectural weakness for the proposed experiments. In fact, base DQV and DQVMax show a robust performance in the offline setting that is unaffected by the addition of more heads to estimate uncertainty. By using a function to form the TD-targets different from the one dedicated to evaluate current states, these algorithms already take significant steps to prevent the bootstrapping error, which makes them unfit candidates for the simple ensemble strategy of Ensemble-DQN.

Another limitation is the naive ensemble technique employed throughout the experiments adapted from Ensemble-DQN. Given the empirical results, it is not sufficient to increase the number of prediction heads in DQV and DQV-Max, to apply standard value-based regression methods (i.e. temporal difference learning) individually on each, and finally to train every head on the average of the ensemble total loss. As previously stated, the former are two strong off-policy algorithms; yet for offline RL they could still benefit from the uncertainty estimated by an ensemble of Q or V functions, if this information is properly integrated in their learning formulation. The simple averaging technique implemented in this research does not fully exploit the uncertainty information provided by the ensembles of Q and V functions. It would be interesting to see what happens when conservative estimates are formed in the face of uncertainty, for example, by down-scaling the TD-targets by the variance of the ensemble predicted Q-values as discussed in Levine et al. (2020).

For future work with a setup similar to this research, different lines of experimentation are possible. For example, the size and diversity of the offline dataset $\mathcal{B}$ collected by π_β could be manipulated to asses the generalization capabilities of offline DQV and DQV-Max. As seen in some of the experiments for the REM agent (Agarwal et al., 2020), the size of the dataset $\mathcal{B}$ could be reduced to find the minimum amount of trajectories needed to obtain an acceptable performance level. Alternatively, offline DQV and DQV-Max could be given only expert or quasi-random data, thus decreasing $\mathcal{B}$'s diversity. The offline agents in this research learned on the full set of policies encountered during online training of π_{DQN}, in lieu of the importance of training datasets' diverse composition highlighted by the REM results. Learning on data produced by a small number of policies or by highly suboptimal ones matters for offline DQV and DQV-Max because it resembles what is available from many settings in the real world, where behavioral data are collected by a handful of static policies – or even a single one.

Finally, regarding uncertainty estimation in offline RL, the ensemble component could be extracted from the single algorithm scope and applied to different learners. This means having a multitude of agents (e.g. both DQV and DQV-Max) learn and collaborate on the same problem, implementing a voting procedure to decide, for example, on the TD-targets for the whole ensemble. The same information relating to uncertainty estimation purposes available from individual heads in the current setup could then come from different RL algorithms that inform the ensemble decisions. Voting

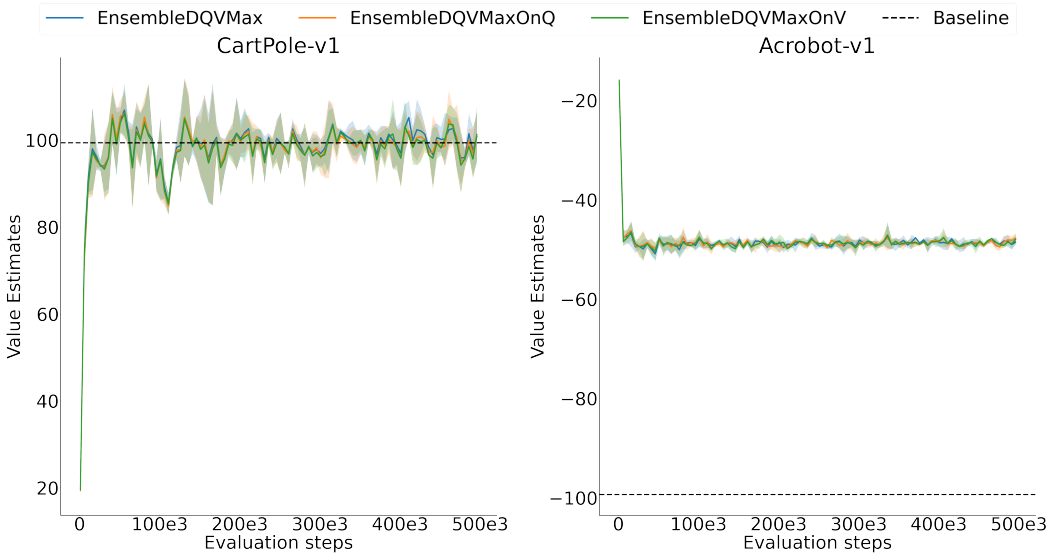

Figure 3.4: Evaluation time Q-value estimates of the ablated variants of offline Ensemble-DQV-Max. The shaded areas are ± 1 standard deviation from the mean of 3 different simulations.

ensembles are a well-defined concepts in machine learning, and in the case of DQV and DQV-Max it could be of interest to assess if the relative strength of each agent taken individually – lower variance for the first on-policy algorithm, greater learning generality for the second off-policy one – can be combined in a meaningful way in the offline RL setting.

References

Agarwal, R., Schuurmans, D., & Norouzi, M. (2020). An optimistic perspective on offline reinforcement learning. In *International conference on machine learning* (pp. 104–114).

Anschel, O., Baram, N., & Shimkin, N. (2017). Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International conference on machine learning* (pp. 176–185).

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Bellman, R. (1957). Dynamic programming, princeton univ. *Press Princeton, New Jersey*.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., ... Zhang, Q. (2018). *JAX: composable transformations of Python+NumPy programs.* Retrieved from `http://github.com/google/jax`

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *Openai gym.* arXiv. Retrieved from `https://arxiv.org/abs/1606.01540` doi: 10.48550/ARXIV.1606.01540

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., & Bellemare, M. G. (2018). Dopamine: A Research Framework for Deep Reinforcement Learning. Retrieved from `http://arxiv.org/abs/1812.06110`

Colas, C., Sigaud, O., & Oudeyer, P.-Y. (2019). *A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms.* arXiv. Retrieved from `https://arxiv.org/abs/1904.06979` doi: 10.48550/ARXIV.1904.06979

Fujimoto, S., Meger, D., & Precup, D. (2019, 09–15 Jun). Off-policy deep reinforcement learning without exploration. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (Vol. 97, pp. 2052–2062). PMLR. Retrieved from https://proceedings.mlr.press/v97/fujimoto19a.html

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2017). Deep reinforcement learning that matters. *CoRR*, *abs/1709.06560*. Retrieved from http://arxiv.org/abs/1709.06560

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kumar, A., Fu, J., Soh, M., Tucker, G., & Levine, S. (2019). Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, *32*.

Kumar, Aviral. (2019). *Data-Driven Deep Reinforcement Learning.* https://bair.berkeley.edu/blog/2019/12/05/bear/. ([Online; accessed 11-July-2022])

Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, *8*(3), 293–321.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Osband, I., Blundell, C., Pritzel, A., & Van Roy, B. (2016). Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, *29*.

Sabatelli, M., Louppe, G., Geurts, P., & Wiering, M. A. (2020). The deep quality-value family of deep reinforcement learning algorithms. In *2020 international joint conference on neural networks (ijcnn)* (pp. 1–8).

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

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Thrun, S., & Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 connectionist models summer school hillsdale, nj. lawrence erlbaum* (Vol. 6, pp. 1–9).

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 30).

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. doi: 10.1038/s41592-019-0686-2

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3), 279–292.

A Appendix

Following the practice from Henderson et al. (2017) and Colas et al. (2019), Welch's t-test was used to test whether each ensemble variant of the analyzed algorithms performed worse than its respective base version in terms of the defined response metrics. Significant results are highlighted in Table 3.1. For the Ensemble-DQV-Max ablations experiment, a Kruskal-Wallis H-test was used to compare the former agent to its ablated variants. Both test were performed with the corresponding routines from the `SciPy` Python package (Virtanen et al., 2020).

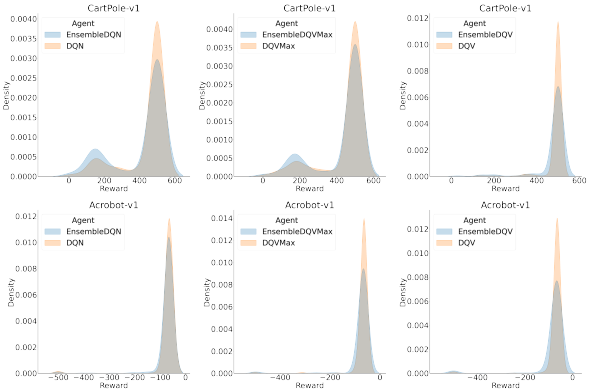

Figure A.3: Distribution of rewards: each offline agent is compared against its ensemble variant

A.1 Additional plots

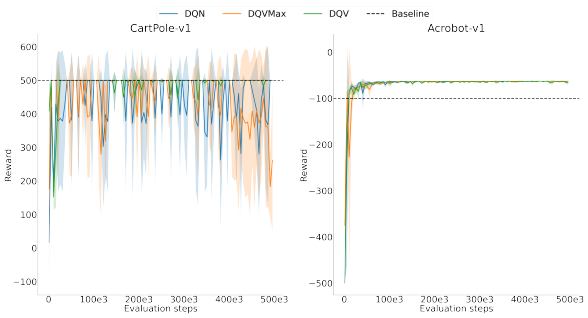

Figure A.1: Evaluation time reward signal for offline DQN, DQV and DQV-Max, averaged over 3 runs

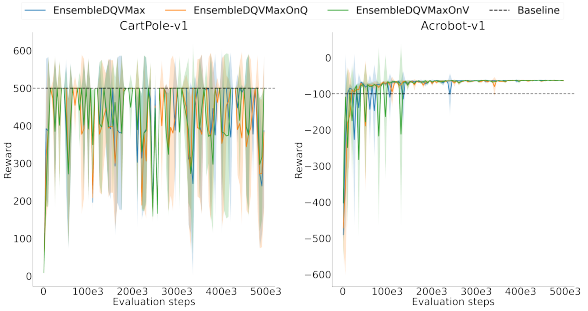

Figure A.4: Ensemble-DQV-Max reward signals for the ablations experiment, averaged over 3 runs

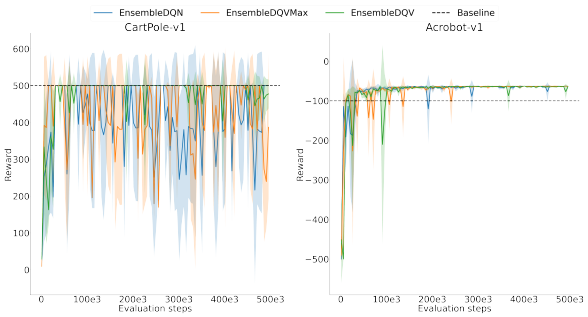

Figure A.2: Evaluation time reward signal for the ensemble variants of offline DQN, DQV and DQV-Max, averaged over 3 runs

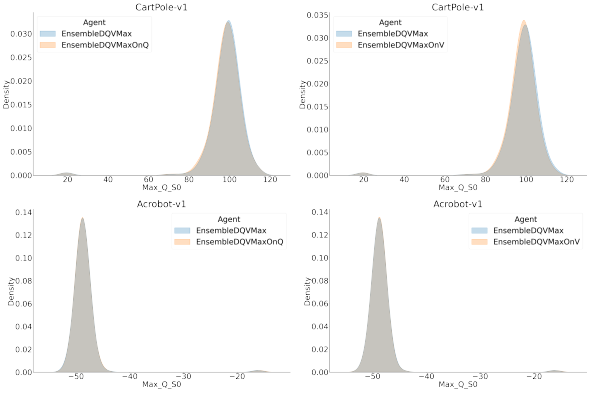

Figure A.5: Distribution of maximum Q-values for s_0 for Ensemble-DQV-Max compared against its ablations

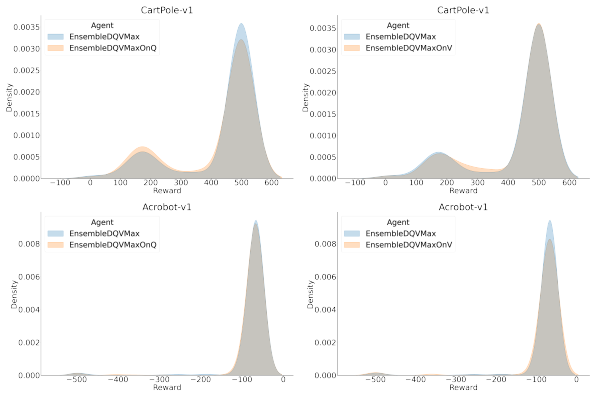

Figure A.6: Distribution of rewards for Ensemble-DQV-Max compared to its ablations

A.2 Hyper-parameters

Table A.1: Hyper-parameters used for online data collection and in the experiments

Hyper-parameter	Value (online and offline)	
(Training/Evaluation) steps	1000	
Iterations	500	
Redundancy	3	
Reward clipping	$[-1, 1]$	
Target network update period (steps)	100	
Discount factor γ	0.99	
Exploration strategy	ϵ-greedy	
Evaluation ϵ	0.001	
Replay memory size	500 000 trajectories	
Mini-batch size	32	
Replay scheme	Uniform	
Optimizer	Adam	
Adam ϵ	$3.125e-4$	
Adam learning rate	0.001	
Loss function	Mean Squared ErrorTraining ϵ	
Networks number of layers	2	
Hidden units per layer	512	
Hyper-parameter	Online	Offline
Training period (steps)	4	1
Min. memory size for replay	500	-
Training ϵ	0.01	-
Evaluation period (iterations)	-	5