Revisiting Bellman Errors for Offline Model Selection

Anonymous Author(s)
Affiliation
Address
email

1 Introduction

There has been a recent surge of reinforcement learning (RL) algorithms that can train an agent offline with varying success [11, 15]. Most such algorithms rely on online environment interactions to tune their own hyperparameters. However, applying offline RL algorithms in real-world settings necessitates tuning their hyperparameters fully offline (i.e. offline model selection). Prevalent offline model selection approaches are typically based on offline policy evaluation (OPE) algorithms that estimate the expected returns under a given policy [28, 34, 38]. Unfortunately, accurately estimating the expected returns under one policy using data generated by a completely different behavioral policy can be very difficult. Other approaches that have been explored have been based on empirical Bellman errors as an approximation for the true Bellman errors [13, 21, 28]. These works have found empirical Bellman errors to be poor predictors of value model accuracy, leading to a current belief that Bellman errors are not useful in offline model selection [10, 14, 28].

Instead, our work argues that empirical Bellman errors have been found unhelpful in model selection because they are poor proxies for the true Bellman errors. We propose a new algorithm, *Supervised Bellman Validation (SBV)*, that uses supervised learning to derive a Bellman error-based validation loss function and provide a better proxy for the true Bellman errors. We argue several theoretical benefits of SBV over competing methods and apply SBV to offline Atari datasets [3, 15] containing high-dimensional image observations. We explore conditions in which approaches based on Bellman errors can work well for model selection, and demonstrate that SBV performs competitively when these conditions are satisfied. Finally, we discuss implications of our results and suggest potential avenues for future work.

2 Preliminaries

In offline RL [24, 26], we have a static dataset $\mathcal{D} = \{(s, a, r, s')\}$ consisting of $|\mathcal{D}|$ transitions (s, a, r, s') where $r \in \mathbb{R}$ is the immediate reward observed after state $s \in \mathcal{S}$ was observed and action $a \in \mathcal{A}$ was taken, before observing the new state $s' \in \mathcal{S}$. We assume the data comes from a Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, d_0, \gamma)$ [30] with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $T(s'|a, s) = \Pr(S_{t+1} = s'|A_t = a, S_t = s)$, reward function $R(S_t, A_t, S_{t+1}) = R_t$, initial state probabilities $d_0(s) = \Pr(S_0 = s)$, and discount factor $\gamma \in (0, 1)$. We can think of the observed state-action pairs in $\mathcal{D}$ as being sampled from (or being identically distributed as) $P^{\mu}_{SA}(s, a) = d^{\mu}(s)\mu(a|s)$ where μ is the *behavioral policy* and d^{μ} is the marginal distribution of states over time points induced by policy μ and MDP $\mathcal{M}$ (see section A of the Appendix for a more thorough definition of the behavioral policy). Likewise, we can think of the observed transitions (s, a, r, s') in $\mathcal{D}$ as being sampled from (or being identically distributed as) $P^{\mu}_{T}(s, a, r, s') = d^{\mu}(s)\mu(a|s)T(s'|a, s)$ where $r = R(s, a, s')$.

Define the *action-value function* for policy π as $Q^{\pi}(s, a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s, A_0 = a]$ where $\mathbb{E}_{\pi}$ denotes expectation over MDP $\mathcal{M}$ and policy π. The optimal policy π^* is the policy whose action-value function equals the optimal action-value function $Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$. For any real-valued function of state-action pairs Q (oftentimes called a *Q-function*), the *Bellman operator* $\mathcal{B}^*$ satisfies:

$$\mathcal{B}^* Q(s, a) = \mathbb{E}[R_t + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') | S_t = s, A_t = a] \tag{1}$$

It can be shown that $\pi^* = \pi_{Q^*}$, where $\pi_Q(s) = \text{argmax}_a Q(s, a)$ is the greedy policy of Q-function Q, and that $Q = \mathcal{B}^* Q$ if and only if $Q = Q^*$ [33].

For any Q-function Q, density P of state-action pairs and dataset of transitions $\mathcal{S} = \{(s, a, r, s')\}$, let $||Q||^2_P = \mathbb{E}_{(s,a)\sim P}[Q(s, a)^2]$ and $||Q||^2_{\mathcal{S}} = |\mathcal{S}|^{-1} \sum_{(s,a)\in\mathcal{S}} [Q(s, a)^2]$. Define the *Bellman backup function* for Q-function Q as $(\mathcal{B}^* Q)(s, a)$ and its *Bellman error function* as $(Q - \mathcal{B}^* Q)(s, a)$ (see Equation 1). Likewise, define the *empirical*

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Bellman backups for Q-function Q and dataset $\mathcal{S}$ as $(\mathcal{B}_{\mathcal{S}}Q)(s,a) = r + \gamma \max_{a'} Q(s',a'),\ (s,a,r,s') \in \mathcal{S}$ and their *empirical Bellman errors* as $Q(s,a) - (\mathcal{B}_{\mathcal{S}}Q)(s,a),\ (s,a) \in \mathcal{S}$.

3 Related Work

Throughout we assume all estimated value functions and policies are fitted by Q-learning algorithms (Q-learning is discussed briefly in section A of the Appendix). In the *offline model selection* problem, we have a set of candidate value models $\mathcal{Q} = \{Q_1, ..., Q_M\}$, or estimators for Q^*, and our goal is to choose the "best" among them based on some criterion. Here each candidate in $\mathcal{Q}$ was estimated via a Q-learning algorithm with a different hyperparameter configuration. For example, candidates in $\mathcal{Q}$ could include Q-functions obtained from running a single DQN algorithm and stopping at different numbers of training iterations, or it could include Q-functions obtained from running DQN with different Q-network architectures. The most popular approach for offline model selection is to use an *offline policy evaluation (OPE)* algorithm, which estimates the marginal expectation of returns $J(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_t]$ under policies of interest $\pi \in \{\pi_{Q_1}, \pi_{Q_2}, ..., \pi_{Q_M}\}$ [12]. For example, *importance sampling* estimators directly estimate $J(\pi)$ from $\mathcal{D}$ by using importance weights to adjust for the distribution shift [29, 34]. *Fitted Q-evaluation* indirectly estimates $J(\pi)$ by estimating Q^π using another off-policy RL algorithm [25, 28]. Finally, *model-based* approaches indirectly estimates $J(\pi)$ by estimating the transition kernel $T(\cdot|a,s)$ for all $(a,s) \in \mathcal{A} \times \mathcal{S}$ using standard density estimation techniques [7, 32, 38].

Existing OPE approaches often have difficulties with accurately estimating $J(\pi)$. In particular, importance sampling usually has prohibitively-large estimation variance, FQE introduces its own hyperparameters that cannot be easily tuned offline and model-based approaches can have great difficulty modelling the entire underlying MDP in complex and high-dimensional settings. For these reasons, all of these methods have achieved highly sub-optimal performance on certain environments and most of them have not even been attempted on environments with pixel-valued states [12]. In contrast to estimating the expected returns associated with the greedy policies implicit in $\mathcal{Q}$, another approach is to assess candidates $Q_m, 1 \leq m \leq M$ via their *mean squared Bellman error (MSBE)*:

$$||Q_m - \mathcal{B}^* Q_m||^2_{P^\mu_{SA}} = \mathbb{E}_{(s,a)\sim P^\mu_{SA}} \left[\left(Q_m(s,a) - (\mathcal{B}^* Q_m)(s,a)\right)^2 \right] \tag{2}$$

We discuss theoretical justifications for using the Bellman error for model selection in section 5. To estimate the true MSBE (Equation 2), previous literature has used the *empirical mean square Bellman error (EMSBE)* [13, 21, 28]:

$$|Q_m - \mathcal{B}_{\mathcal{D}} Q_m||^2_{\mathcal{D}} = \frac{1}{|\mathcal{D}|} \sum_{(s,a,r,s')\in\mathcal{D}} \left[\left(r + \max_{a'\in\mathcal{A}} Q_m(s',a') - Q_m(s,a) \right)^2 \right] \tag{3}$$

The problem is that the EMSBE (Equation 3) is a **biased** estimator of the true MSBE and its minimizer can be an arbitrarily poor estimator of Q^* even in the asymptotic case (see section 5 for details). It is likely for this reason that previous literature has found this to be ineffective at hyperparameter tuning [13, 21].

The work by Farahmand and Szepesvari [10] is more related to our approach in that they propose supervised learning to approximate the true Bellman errors. There are however two major distinctions of Farahmand and Szepesvari [10] over our work: 1) their algorithm is much more sample inefficient as they require partitioning the data into thirds as well as tight probabilistic upper bounds on the *excess risk* of the estimates of $\mathcal{B}^* Q_m$ [35]; 2) they did not investigate the empirical performance of their algorithm or of the MSBE more generally.

4 Supervised Bellman Validation

To understand *Supervised Bellman Validation (SBV)*, consider first the case where $Q^*(s,a)$ is actually **known** for state-action pairs $(s,a) \in \mathcal{D}$ and we wish to select the candidates $Q_m \in \mathcal{Q}$ that best estimate Q^*. An obvious criterion in this case would be the *value function estimation mean squared error (MSE)*:

$$||Q^* - Q_m||^2_{P^\mu_{SA}} = \mathbb{E}_{(s,a)\sim P^\mu_{SA}} \left[\left(Q^*(s,a) - Q_m(s,a)\right)^2 \right]. \tag{4}$$

While P^μ_{SA} is unknown, we can estimate the associated expectation by randomly partition 80% of the trajectories present in $\mathcal{D}$ into a training set $\mathcal{D}_T$ and reserving the remaining 20% of trajectories as a validation set $\mathcal{D}_V$. We can then generate candidates $\mathcal{Q} = \{Q_1, ..., Q_M\}$ by running deep Q-learning algorithms on $\mathcal{D}_T$ with M different hyperparameter configurations, and use $\mathcal{D}_V$ to estimate the value function estimation MSE (Equation 4) as $||Q^* - Q_m||^2_{\mathcal{D}_V}$, for each $1 \leq m \leq M$.

In real-world RL, the targets $Q^*(s,a), (s,a) \in \mathcal{D}$ are not known: this is what separates supervised learning from offline RL. Instead of a criterion based on Equation 4, SBV uses a surrogate criterion based on the MSBE $||Q_m - \mathcal{B}^* Q_m||^2_{P^\mu_{SA}}$ (Equation 2). The relationship between estimation error and Bellman error is discussed more in section 5. Similar to what

Algorithm 1: Supervised Bellman Validation (SBV)

Input: Set of observed transitions $\mathcal{D} = \{(s, a, r, s')\}$, hyperparameter configurations to evaluate $\mathcal{H} = \{H_1, ..., H_m\}$
Randomly partition trajectories in $\mathcal{D}$ to training set $\mathcal{D}_T$ and validation set $\mathcal{D}_V$
for *configuration* $m \in \{1, ..., M\}$ **do**
  Estimate Q^* as Q_m by running a Q-learning algorithm on $\mathcal{D}_T$ with configuration H_m
  Estimate $\mathcal{B}^* Q_m$ as $\widehat{\mathcal{B}}^* Q_m$ by minimizing $||\mathcal{B}_{\mathcal{D}_T} Q_m - f||^2_{\mathcal{D}_T}$ wrt f (see Equation 5)
  Estimate the MSBE of Q_m as $||Q_m - \widehat{\mathcal{B}}^* Q_m||^2_{\mathcal{D}_V}$ (see Equation 6).
end
Output Q_{m^*} as our estimate of Q^* where $m^* = \text{argmin}_{1 \leq m \leq M} ||Q_m - \widehat{\mathcal{B}}^* Q_m||^2_{\mathcal{D}_V}$

we would do in supervised learning, SBV creates a training set $\mathcal{D}_T$ and a validation set $\mathcal{D}_V$ by randomly partitioning trajectories from $\mathcal{D}$. SBV then trains M Q-functions $\mathcal{Q} = \{Q_1, ..., Q_M\}$ on $\mathcal{D}_T$. Note that the MSBE actually contains **two** unknown quantities: the population density P^{μ}_{SA}, and the M Bellman backup functions $\mathcal{B}^* Q_m, 1 \leq m \leq M$. From Equation 1, we see that each $(\mathcal{B}^* Q_m)(s, a)$ is just a conditional expectation. Therefore, the M Bellman backup functions can be estimated by running M regression algorithms on $\mathcal{D}_T$, with the mth such algorithm estimating $\mathcal{B}^* Q_m$ by fitting a function f to minimize the *Bellman backup MSE*:

$$||\mathcal{B}_{\mathcal{D}_T} Q_m - f||^2_{\mathcal{D}_T} = \frac{1}{|\mathcal{D}_T|} \sum_{(s,a,r,s') \in \mathcal{D}_T} \left[\left(r + \gamma \max_{a'} Q_m(s', a') - f(s, a) \right)^2 \right]. \tag{5}$$

Denote the fitted models from our regression algorithms as $\widehat{\mathcal{B}}^* Q_1, ..., \widehat{\mathcal{B}}^* Q_M$. The MSBE for each candidate $Q_m \in \mathcal{Q}$ can then be estimated as:

$$||Q_m - \widehat{\mathcal{B}}^* Q_m||^2_{\mathcal{D}_V} = \frac{1}{|\mathcal{D}_V|} \sum_{(s,a) \in \mathcal{D}_V} \left[\left(Q_m(s, a) - (\widehat{\mathcal{B}}^* Q_m)(s, a) \right)^2 \right]. \tag{6}$$

Our method is summarized in Algorithm 1. Here a hyperparameter configuration H_m specifies all relevant hyperparameters of the Q-learning algorithm, such as the specific training algorithm (e.g. double DQN [16] or duelling DQN [36]), the Q-network architecture and the number of training iterations.

To instantiate SBV, Algorithm 2 provides a practical implementation of SBV for tuning the number of training iterations used by DQN. Here the *Q-Network* is a neural network Q-function Q_θ with trainable parameters θ that is trained by DQN to approximate Q^*, and $Q_{\theta_{k+1}}$ denotes the trained Q-Network after $k + 1$ iterations of the DQN algorithm. Moreover, the *Bellman Network* is a neural network model $\mathcal{B}_\phi$ with trainable parameters ϕ that is updated by SBV after every DQN training iteration to approximate $\mathcal{B}^* Q_\theta$ for the current Q-Network Q_θ. We let $\mathcal{B}_{\phi_{k+1}}$ denote the Bellman network updated by SBV in order to approximate $\mathcal{B}^* Q_{\theta_{k+1}}$.

While the Q-network hyperparameters can be tuned via SBV, the Bellman network itself has hyperparameters which must be tuned. As the Bellman network is solving a series of supervised learning problems, the relevant hyperparameters can easily be tuned offline by minimizing validation Bellman backup MSE on $\mathcal{D}_V$ across the various sets of targets. The ability of SBV to fully tune its own hyperparameters offline gives it real-world applicability and sets it apart from other approaches like fitted Q-evaluation, whose hyperparameters have traditionally been chosen based on online evaluations or previous literature on the environments of interest [28].

Algorithm 2: Applying Early Stopping to DQN with SBV

Input: Set of observed transitions $\mathcal{D} = \{(s, a, r, s')\}$
Randomly partition trajectories in $\mathcal{D}$ to training set $\mathcal{D}_T$ and validation set $\mathcal{D}_V$
Initialize deep Q-network Q_{θ_0} with trainable parameters θ_0 and Bellman network $\mathcal{B}_{\phi_0}$ with trainable parameters ϕ_0
for *iteration* $k \in \{0, ..., K - 1\}$ **do**
  Update θ_{k+1} from θ_k by running DQN on $\mathcal{D}_T$ for one iteration
  Update ϕ_{k+1} from ϕ_k via gradient descent to minimize $||\mathcal{B}_{\mathcal{D}_T} Q_{\theta_{k+1}} - \mathcal{B}_\phi||^2_{\mathcal{D}_T}$ wrt ϕ (see 5)
  Estimate of the MSBE of $Q_{\theta_{k+1}}$ as $||Q_{\theta_{k+1}} - \mathcal{B}_{\phi_{k+1}}||^2_{\mathcal{D}_V}$ (see 6)
end
Estimate Q^* as Q_{k^*} and π^* as $\pi_{Q_{k^*}}$ where $k^* = \text{argmin}_{1 \leq k \leq K} ||Q_{\theta_k} - \mathcal{B}_{\phi_k}||^2_{\mathcal{D}_V}$

5 Theoretical Results

We begin by introducing some novel theoretical results to support our algorithm, the proofs of which can be found in section B of the Appendix. Our first theorem, given below, implies that a candidate value model Q_m with sufficiently small Bellman errors should be sufficiently accurate and have a greedy policy that gives sufficiently high return. Moreover, if the Bellman errors $Q_m - \mathcal{B}^* Q_m$ are unknown but are estimated with sufficient accuracy, the estimated Bellman errors will still be informative in lower-bounding the estimation accuracy and policy value.

Theorem 1. *Let $\widehat{\mathcal{B}}^* Q_m$ be an estimate of $\mathcal{B}^* Q_m$ and assume that $||Q_m - \widehat{\mathcal{B}}^* Q_m||_\infty \leq \epsilon$ and $||\widehat{\mathcal{B}}^* Q_m - \mathcal{B}^* Q_m||_\infty \leq \delta$. Then $||Q - Q^*||_\infty \leq \frac{1}{1-\gamma}(\epsilon + \delta)$ and $||V^{\pi^*} - V^{\pi_Q}||_\infty \leq \frac{2}{(1-\gamma)^2}(\epsilon + \delta)$ where $V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)}\left[Q^\pi(s,a)\right]$.*

Let F^μ be the cumulative distribution function (CDF) associated with P_T^μ and let $F^{\mathcal{D}}$ be the *empirical distribution function (EDF)* of transitions associated with $\mathcal{D}$ [23]. Under general conditions, $F^{\mathcal{D}}$ converges to F^μ as $|\mathcal{D}|$ grows large. To study the bias and asymptotic properties of the EMSBE (Equation 3) and SBV (Algorithm 1), we focus on the setting where $|\mathcal{D}| = \infty$ and $F^{\mathcal{D}} = F^\mu$. Theorem 2 states that the EMSBE has non-negligible bias and that its minimum need not be close to Q^* even with infinite samples. In contrast, Theorem 3 states that SBV correctly recovers the true MSBE and correctly selects the true optimal action-value function Q^*, at least in the asymptotic case. These theorems help explain why SBV performs well empirically, both overall and relative to the EMSBE (see section 6).

Theorem 2. *Assume that $F^{\mathcal{D}} = F^\mu$. Then $\text{argmin}_{Q:\mathcal{S}\times\mathcal{A}\to\mathbb{R}} ||Q - \mathcal{B}_{\mathcal{D}} Q||^2_{\mathcal{D}} \neq Q^*$ in general and:*

$$||Q_m - \mathcal{B}_{\mathcal{D}} Q_m||^2_{\mathcal{D}} - ||Q_m - \mathcal{B}^* Q_m||^2_{P^\mu_{SA}} = \mathbb{E}_{(S_t,A_t)\sim P^\mu_{SA}}\left\{ \textit{Var}\left[R_t + \max_{a' \in \mathcal{A}} Q_m(S_{t+1}, a') | S_t, A_t \right] \right\}$$

.

Theorem 3. *Assume the following: (1) $F^{\mathcal{D}} = F^\mu$; (2) $\mathcal{D}_T$ and $\mathcal{D}_V$ partition the trajectories present in $\mathcal{D}$; (3) $\mathcal{B}^* Q_m$ is estimated as $\widehat{\mathcal{B}}^* Q_m = \text{argmin}_{f:\mathcal{S}\times\mathcal{A}\to\mathbb{R}} ||\mathcal{B}_{\mathcal{D}_T} Q_m - f||^2_{\mathcal{D}_T}$. Then under mild technical conditions and with probability one, $\widehat{\mathcal{B}}^* Q_m = \mathcal{B}^* Q_m$, $||Q_m - \widehat{\mathcal{B}}^* Q_m||^2_{\mathcal{D}_V} = ||Q_m - \mathcal{B}^* Q_m||^2_{P^\mu_{SA}}$ and $||Q_m - \widehat{\mathcal{B}}^* Q_m||^2_{\mathcal{D}_V} = 0$ if and only if $Q_m = Q^*$.*

6 Experiments

We evaluated Supervised Bellman Validation (SBV; Algorithm 1) on four offline DQN-Replay datasets[3] associated with the following Atari games [4]: Pong, Breakout, Asterix and Seaquest. See section A of the Appendix for details on why we chose these datasets. Following [3], we uniformly sub-sampled 25% of our DQN-replay datasets to obtain four training datasets with 10M transitions and four associated validation datasets with 2.5M transitions. For each training dataset, we evaluate a Q-network after each training iteration across two deep Q-learning configurations, which we dub DQN (S) and DDQN (L). DQN (S) uses the standard DQN architecture [27] and the Adam optimizer following Agarwal et al. [3]. The DDQN (L) configuration is described in section C of the Appendix, and involves a deeper network architecture and double Q-learning target updates [16]. Each training configuration was run for 50 iterations, resulting in 100 candidate Q-networks to evaluate for each training dataset.

We implemented SBV in a manner similar to Algorithm 2 to improve computational efficiency. Moreover, the Bellman network training configuration was tuned offline so as to minimize validation error across the 400 evaluated Q-functions and four Atari games, as discussed in section 4. To demonstrate the efficacy of SBV, we compare it to EMSBE (Equation 3) and *weighted per-decision importance sampling (WIS)* [29]. Full implementation details of SBV and WSE can be found in section D of the Appendix. Due to the computational demands of implementing Fitted Q-evaluation (FQE) and the inability to tune its hyperparameters offline, we have omitted results for FQE for now. We also do not implement model-based approaches here: accurately estimating the underlying MDP of Atari environments offline would be completely novel and extremely challenging, warranting a paper in and of itself.

In Table 1, we calculate the mean return of the top-5 policies according to each method for each environment, similar to Fu et al. [12], and standardize the mean returns to a $[0, 1]$ range. We can see here that SBV consistently performs better than competing methods with respect to top-5 policy returns. EMSBE does not perform as well as SBV, but it still performs better than WIS.

In Figure 1, we plot learning curves for the best configuration for each environment, and indicate the iteration where training was stopped by each method. Again we see that SBV performs significantly better than competing methods on average. Moreover, only SBV performs as well or better than applying no early stopping across all environments.

Theorem 1 and our empirical investigations suggest that the MSBE provides a tighter lower bound on estimation error and a looser lower bound on policy value. Consequently, the lower the MSBE value, the more informative it will be. For example, if $J(\pi^*) = 10$, value models with an MSBE of 0.1 may have universally high estimation error but may have associated policy values in the wide range of $[1, 10]$. On the other hand, value models with an MSBE of only 0.01 could

Table 1: Standardized Top-5 Policy Return Means. A standardized return of 0% implies that the method choose the worst five policies possible on the given dataset (based on online evaluations), while a return of 100% implies that the method choose the best five policies possible.

Method	Pong	Breakout	Asterix	Seaquest
WIS [29]	41%	40%	50%	13%
EMSBE (Equation 3)	77%	43%	51%	44%
SBV (Ours)	**94%**	**79%**	**76%**	**65%**

have policy values residing in the much tighter range of $[9, 10]$ (they will also all have much lower estimation error). SBV (and any other accurate estimator of the MSBE) will thus be an effective model selector when the set of candidate value functions $\mathcal{Q}$ include value functions that have sufficiently low Bellman error. This condition can be satisfied in practice by exploring a sufficiently large number of training configurations and/or by exploring training configurations which have been shown in previous work to achieve robust performance in a variety of scenarios. See section E of the Appendix for a more in-depth discussion of this condition as well as other conditions of SBV performance.

7 Discussion and Future Work

Our theoretical and empirical results challenge the current belief in the RL community that Bellman errors are not helpful for offline model selection. SBV only requires learning a univariate regression function, making it 1) more statistically efficient than either offline RL or model-based learning; 2) straightforward to tune SBV's own hyperparameters to the dataset at hand; and 3) less biased and more sample-efficient than using no function approximation whatsoever. Our empirical results on Atari games show that Bellman error-based model selection can perform well in real-world settings provided certain conditions are met. We hope our work will change existing attitudes and conventions in offline RL and facilitate the development of better methodology for offline model selection.

Our work opens up several avenues for future research. For example, one avenue for future work is to investigate how to achieve sufficient performance with SBV while reducing computational demands. Reducing the number of configurations we have to explore before achieving sufficient performance with SBV would be helpful here, as would modifying the Bellman network training configuration to improve computational efficiency. Moreover, our results focus on discrete control problems and Q-learning algorithms. In contrast, continuous control problems and deep actor-critic algorithms require two neural network models: a Q-network Q_θ and a *policy model* π_α. The γ-contraction properties of the Bellman operator and policy iteration theory [5] suggests that a high-performing policy model π_α requires that both $Q_\theta \approx \mathcal{B}^{\pi_\alpha} Q_\theta$ and that $\pi_\alpha \approx \pi_{Q_\theta}$. While SBV could help ensure that the first condition is satisfied, extending SBV to deal with the second condition is left for future work. Finally, recall that a major shortcoming of FQE is that it has its own hyperparameters that cannot be easily tuned offline. in theory, one can use SBV to tune the hyperparameters of FQE and then use FQE to tune the hyperparameters of deep Q-learning or deep actor-critic. The main issue here is the computational demand: both SBV and FQE are already computationally demanding procedures in their own right, and combining them in a computationally feasible manner is nontrivial.

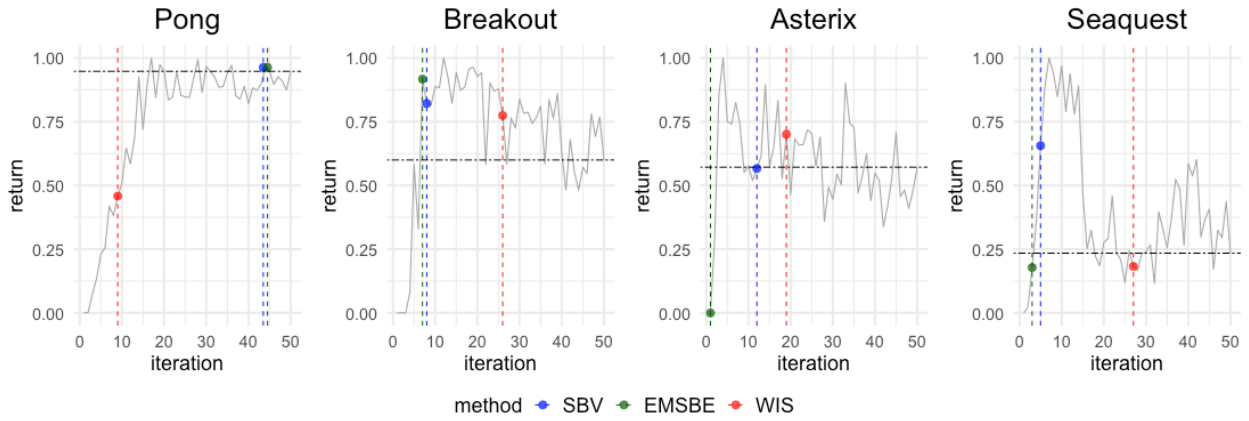

Figure 1: Learning Curves for the Best Configuration. Returns are standardized to a $[0, 1]$ range with zero being the lowest return observed for a given run and one being the highest return observed. The dashed horizontal line represents performance when no early stopping is applied. The vertical lines represent the iterations where training was stopped according to different methods.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Alekh Agarwal, Nan Jiang, Sham M. Kakade, and Wen Sun. Reinforcement learning: Theory and algorithms, 2022. URL `https://rltheorybook.github.io/`. A working draft.

[3] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *ICML*, 2020.

[4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. doi: 10.1613/jair.3912.

[5] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, 2010.

[6] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.

[7] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*, 2018.

[8] Timothy Dozat. Incorporating nesterov momentum into adam. In *ICLR*, 2016.

[9] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *ICML*, 2018.

[10] Amir Massoud Farahmand and Csaba Szepesvari. Model selection in reinforcement learning. *Machine Learning*, 85:299–332, 2010.

[11] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[12] Justin Fu, Mohammad Norouzi, Ofir Nachum, G. Tucker, Ziyun Wang, Alexander Novikov, Mengjiao Yang, Michael R. Zhang, Yutian Chen, Aviral Kumar, Cosmin Paduraru, Sergey Levine, and Tom Le Paine. Benchmarks for deep off-policy evaluation. In *ICLR*, 2021.

[13] Scott Fujimoto, David Meger, Doina Precup, Ofir Nachum, and Shixiang Shane Gu. Why should i trust you, bellman? the bellman error is a poor replacement for value error. *ArXiV*, abs/2201.12417, 2022.

[14] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow*. O'Reilly, second edition, 2019.

[15] Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Thomas Paine, Sergio Gómez, Konrad Zolna, Rishabh Agarwal, Josh S Merel, Daniel J Mankowitz, Cosmin Paduraru, et al. Rl unplugged: A suite of benchmarks for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:7248–7259, 2020.

[16] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, page 2094–2100, 2016.

[17] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009. doi: 10.1007/978-0-387-84858-7.

[18] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[19] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42:2011–2023, 2020.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[21] Alex Irpan, Kanishka Rao, Konstantinos Bousmalis, Chris Harris, Julian Ibarz, and Sergey Levine. Off-policy evaluation via off-policy classification. In *NeurIPS*, 2019.

[22] Steven Kapturowski, Georg Ostrovski, John Quan, Rémi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *ICLR*, 2019.

[23] Samuel Kotz, Campbell B. Read, N. Balakrishnan, Brani Vidakovic, and Norman L. Johnson. *Encyclopedia of Statistical Sciences*. John Wiley & Sons, 2006.

[24] Michail G. Lagoudakis and Ronald E. Parr. Least-squares policy iteration. *J. Mach. Learn. Res.*, 4:1107–1149, 2003.

[25] Hoang Minh Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *ICML*, 2019.

[26] Sergey Levine, Aviral Kumar, G. Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *ArXiv*, abs/2005.01643, 2020.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[28] Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyun Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning. *ArXiv*, abs/2007.09055, 2020.

[29] Doina Precup, Richard S. Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. In *ICML*, 2000.

[30] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[32] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML*, 1990.

[33] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[34] Philip S. Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *ICML*, 2016.

[35] Vladimir N Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[36] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. In *ICML*, 2016.

[37] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992. doi: 10.1007/bf00992698.

[38] Michael R Zhang, Thomas Paine, Ofir Nachum, Cosmin Paduraru, George Tucker, ziyu wang, and Mohammad Norouzi. Autoregressive dynamics models for offline policy evaluation and optimization. In *ICLR*, 2021.

Appendix

A Supplementary Preliminaries

Q-learning algorithms update a working model based on its empirical Bellman backups so as to approximate Q^* [37]. When this model is a deep neural network, such algorithms are called *deep Q-learning algorithms* [27]. Q-learning algorithms are usually used to solve discrete control problems, whereby $\max_{a \in \mathcal{A}} Q_\theta(s,a)$ can be solved exactly and quickly for neural network Q-function Q_θ.

The *offline DQN-Replay datasets* [3] were generated by running online DQN agents on Atari game environments [4] and saving the intermediate transitions observed during training as $\mathcal{D}$. Atari environments are widely studied in deep RL and possess high-dimensional states, stochastic nonlinear transition dynamics and long-term consequences from taking particular actions. They also possess discrete action spaces amenable to Q-learning algorithms. These properties makes Atari environments challenging and applicable candidates for benchmarking real-world performance of SBV.

We can think of the observed state-action pairs in $\mathcal{D}$ as being sampled from (or being identically distributed as) $P^\mu_{SA}(s,a) = d^\mu(s)\mu(a|s)$ where μ is the behavioral policy and d^μ is the marginal distribution of states over time points induced by policy μ and MDP $\mathcal{M}$. In the event that $\mathcal{D}$ was generated by a single stationary policy, μ is simply equal to this policy. However, for datasets such as the DQN Replay Datasets consisting of the intermediate transitions observed from training an online RL algorithm, $\mathcal{D}$ can be thought of as being generated from many different policies and μ becomes a mixture distribution of those policies.

B Mathematical Proofs

Theorem 1. *Let $\widehat{\mathcal{B}}^* Q_m$ be an estimate of $\mathcal{B}^* Q_m$ and assume that $||Q_m - \widehat{\mathcal{B}}^* Q_m||_\infty \leq \epsilon$ and $||\widehat{\mathcal{B}}^* Q_m - \mathcal{B}^* Q_m||_\infty \leq \delta$. Then $||Q - Q^*||_\infty \leq \frac{1}{1-\gamma}(\epsilon + \delta)$ and $||V^{\pi^*} - V^{\pi_Q}||_\infty \leq \frac{2}{(1-\gamma)^2}(\epsilon+\delta)$ where $V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)} Q^\pi(s,a)$.*

Proof. Suppose $||Q_m - \widehat{\mathcal{B}}^* Q_m||_\infty \leq \epsilon$ and $||\widehat{\mathcal{B}}^* Q_m - \mathcal{B}^* Q_m||_\infty \leq \delta$. Then:

$$
\begin{aligned}
||Q_m - Q^*||_\infty =& ||Q_m - \mathcal{B}^* Q_m + \mathcal{B}^* Q_m - Q^*||_\infty \\
\leq& ||Q_m - \mathcal{B}^* Q_m||_\infty + ||\mathcal{B}^* Q_m - Q^*||_\infty && \text{(Subadditivity of } L_\infty \text{ norm)} \\
=& ||Q_m - \mathcal{B}^* Q_m||_\infty + ||\mathcal{B}^* Q_m - \mathcal{B}^* Q^*||_\infty && (Q^* \text{ is a fixed point of } \mathcal{B}^*) \\
\leq& ||Q_m - \mathcal{B}^* Q_m||_\infty + \gamma ||Q_m - Q^*||_\infty && (\mathcal{B}^* \text{ is a } \gamma\text{-contraction}) \\
\Rightarrow ||Q_m - Q^*||_\infty \leq& \frac{1}{1-\gamma} ||Q_m - \mathcal{B}^* Q_m||_\infty \\
=& \frac{1}{1-\gamma} ||Q_m - \widehat{\mathcal{B}}^* Q_m + \widehat{\mathcal{B}}^* Q_m - \mathcal{B}^* Q_m||_\infty \\
\leq& \frac{1}{1-\gamma} \left(||Q_m - \widehat{\mathcal{B}}^* Q_m||_\infty + ||\widehat{\mathcal{B}}^* Q_m - \mathcal{B}^* Q_m||_\infty \right) && \text{(Subadditivity of } L_\infty \text{ norm)} \\
\leq& \frac{1}{1-\gamma} (\epsilon + \delta)
\end{aligned}
$$

Moreover, it was proven in Lemma 1.11 of [2] that $||V^* - V^{\pi_{Q_m}}||_\infty \leq \frac{2}{1-\gamma} ||Q - Q^*||_\infty$. This concludes the proof. □

Theorem 2. *Assume that $F^{\mathcal{D}} = F^\mu$. Then $\text{argmin}_{Q:\mathcal{S}\times\mathcal{A}\to\mathbb{R}} ||Q - \mathcal{B}_{\mathcal{D}} Q||^2_{\mathcal{D}} \neq Q^*$ and:*

$$
||Q_m - \mathcal{B}_{\mathcal{D}} Q_m||^2_{\mathcal{D}} - ||Q_m - \mathcal{B}^* Q_m||^2_{P^\mu_{SA}} = \mathbb{E}_{(S_t,A_t)\sim P^\mu_{SA}} \left\{ Var\left[R_t + \max_{a' \in \mathcal{A}} Q_m(S_{t+1}, a') | S_t, A_t \right] \right\}.
$$

Proof. Suppose $F^{\mathcal{D}} = F^\mu$. Then:

$$
\begin{aligned}
||\mathcal{B}_{\mathcal{D}}Q_m - Q_m||^2_{\mathcal{D}} =& \mathbb{E}_{(s,a,r,s')\sim P^\mu}\left[\left(r + \max_{a'} Q_m(s',a') - Q_m(s,a)\right)^2\right] \\
=& \mathbb{E}_{(s,a,r,s')\sim P^\mu}\left[\left(r + \max_{a'} Q_m(s',a') - \mathcal{B}^*Q_m(s,a) + \mathcal{B}^*Q_m(s,a) - Q_m(s,a)\right)^2\right] \\
=& ||\mathcal{B}^*Q_m - Q_m||^2_{P^\mu} \\
&+ \mathbb{E}_{(s,a,r,s')\sim P^\mu}\left[\left(r + \max_{a'} Q_m(s',a') - \mathcal{B}^*Q_m(s,a)\right)^2\right] \\
&+ 2\mathbb{E}_{(s,a,r,s')\sim P^\mu}\left[\left(\mathcal{B}^*Q_m(s,a) - Q_m(s,a)\right)\left(r + \max_{a'} Q_m(s',a') - \mathcal{B}^*Q_m(s,a)\right)\right]
\end{aligned}
$$

Recalling that $\mathcal{B}^*Q_m(s,a) = \mathbb{E}_{s'\sim T(\cdot|s,a)}[R(s,a,s') + \max_{a'} Q_m(s',a')]$ and $P^\mu(s,a,r,s') = P^\mu(s,a)T(s'|s,a)$:

$$
\begin{aligned}
&\mathbb{E}_{(s,a,r,s')\sim P^\mu}\left[\left(r + \max_{a'} Q_m(s',a') - \mathcal{B}^*Q_m(s,a\right)^2\right] \\
=& \mathbb{E}_{(s,a)\sim P^\mu}\left\{\mathbb{E}_{s'\sim T(\cdot|s,a)}\left[\left(R(s,a,s') + \max_{a'} Q_m(s',a') - \mathcal{B}^*Q_m(s,a)\right)^2\right]\right\} \\
=& \mathbb{E}_{(S_t,A_t)\sim P^\mu}\left\{\text{Var}\left[R_t + \max_{a'} Q_m(S_{t+1},a')|S_t,A_t\right]\right\}
\end{aligned}
$$

Moreover:

$$
\begin{aligned}
&\mathbb{E}_{(s,a,r,s')\sim P^\mu}\left[\left(\mathcal{B}^*Q_m(s,a) - Q_m(s,a)\right)\left(r + \max_{a'} Q_m(s',a') - \mathcal{B}^*Q_m(s,a\right)\right] \\
=& \mathbb{E}_{(s,a)\sim P^\mu}\left\{\mathbb{E}_{s'\sim T(\cdot|s,a)}\left[\left(\mathcal{B}^*Q_m(s,a) - Q_m(s,a)\right)\left(R(s,a,s') + \max_{a'} Q_m(s',a') - \mathcal{B}^*Q_m(s,a\right)\right]\right\} \\
=& \mathbb{E}_{(s,a)\sim P^\mu}\left\{\left(\mathcal{B}^*Q_m(s,a) - Q_m(s,a)\right)\mathbb{E}_{s'\sim T(\cdot|s,a)}\left[\left(R(s,a,s') + \max_{a'} Q_m(s',a') - \mathcal{B}^*Q_m(s,a\right)\right]\right\} \\
=& \mathbb{E}_{(s,a)\sim P^\mu}\left\{\left(\mathcal{B}^*Q_m(s,a) - Q_m(s,a)\right)\times 0\right\} = 0
\end{aligned}
$$

Therefore:

$$
||Q_m - \mathcal{B}_{\mathcal{D}}Q_m||^2_{\mathcal{D}} = ||Q_m - \mathcal{B}^*Q_m||^2_{P^\mu} + \mathbb{E}_{(S_t,A_t)\sim P^\mu}\left\{\text{Var}\left[R_t + \max_{a'} Q_m(S_{t+1},a')|S_t,A_t\right]\right\}
$$

Finally, we need to show that $\text{argmin}_Q ||Q - \mathcal{B}_{\mathcal{D}}Q||^2_{\mathcal{D}} \neq 0$ in general. In the proof of Theorem 3, we show that $\text{argmin}_Q ||Q - \mathcal{B}^*Q||^2_{P^\mu} = Q^*$ (with probability one) under the condition that the behavioral policy μ is uniformly bounded away from zero. Unless state transitions are deterministic, $\mathbb{E}_{(S_t,A_t)\sim P^\mu}\left\{\text{Var}\left[R_t + \max_{a'} Q(S_{t+1},a')|S_t,A_t\right]\right\} \neq 0$ and thus it is clear that $\text{argmin}_Q \left[||Q - \mathcal{B}^*Q||^2_{P^\mu} + \mathbb{E}_{P^\mu}\left\{\text{Var}\left[R_t + \max_{a'} Q(S_{t+1},a')|S_t,A_t\right]\right\}\right]$ will not be the same as $\text{argmin}_Q ||Q - \mathcal{B}^*Q||^2_{P^\mu}$ in general. Specifically, the former will tend to be biased away from $\text{argmin}_Q ||Q - \mathcal{B}^*Q||^2_{P^\mu}$ towards lower-variance Q-functions. This concludes the proof.

□

Theorem 3. *Assume the following: (1) $F^{\mathcal{D}} = F^\mu$; (2) $\mathcal{D}_T$ and $\mathcal{D}_V$ partition the trajectories present in $\mathcal{D}$; (3) $\mathcal{B}^*Q_m$ is estimated as $\widehat{\mathcal{B}}^*Q_m = \text{argmin}_{f:\mathcal{S}\times\mathcal{A}\to\mathbb{R}} ||\mathcal{B}_{\mathcal{D}_T}Q_m - f||^2_{\mathcal{D}_T}$. Then under mild technical conditions and with probability one, $\widehat{\mathcal{B}}^*Q_m = \mathcal{B}^*Q_m$, $||Q_m - \widehat{\mathcal{B}}^*Q_m||^2_{\mathcal{D}_V} = ||Q_m - \mathcal{B}^*Q_m||^2_{P^\mu_{SA}}$ and $||Q_m - \widehat{\mathcal{B}}^*Q_m||^2_{\mathcal{D}_V} = 0$ if and only if $Q_m = Q^*$.*

Proof. (Sketch) In addition to the assumptions given above, we make the following additional technical assumptions: (T1) $\mathcal{D}$ consists of an infinite number of finite-length trajectories $\tau = (s_0, a_0, r_0, s_1, ..., a_T, r_T, s_{T+1})$ distributed iid as $P^{\mu}_{\text{Traj}}(\tau) = \Pr(T|s_0, a_0, ..., a_T, s_{T+1}) d_0(s_0) \prod_{t=0}^{T} \mu(a_t|s_t) T(s_{t+1}|a_t, s_t)$ (T2) There exists an $\epsilon > 0$ such that $\mu(a|s) > \epsilon$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ (T3) $\mathcal{S}$ is countable and $\mathcal{A}$ is finite

Assumption (T3) is made for mathematical convenience. The proofs for when $\mathcal{S}$ and $\mathcal{A}$ are both infinite or even uncountable are similar, but they would require a few minor technical changes and the use of measure theory. When our state-action space is countable, the equalities $\widehat{\mathcal{B}}^* Q_m = \mathcal{B}^* Q_m$ and $\text{argmin}_Q ||Q - \widehat{\mathcal{B}}^* Q||^2_{\mathcal{D}_V} = Q^*$ hold *surely*, i.e. for all state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$. For the uncountable case, there could still exist a set of state-action pairs with measure zero such that these equalities do not hold. However, the probability of observing these state-action pairs would be zero under $P^{\pi}(s, a) = \pi(a|s) d^{\pi}(s)$ for any policy π, and thus these equalities still hold *almost surely*, i.e. with probability one (this difference is mathematically important but practically meaningless).

As $\mathcal{D}_T$ and $\mathcal{D}_V$ are constructed by randomly partitioning the trajectories in $\mathcal{D}$ and $\mathcal{D}$ consists of an infinite number of finite-length trajectories with distribution P^{μ}_{Traj}, it is easy to see that $\mathcal{D}_T$ and $\mathcal{D}_V$ would themselves consist of an infinite number of trajectories with the same distribution. It is also easy to see that a population distribution of trajectories P^{μ}_{Traj} induces the population distribution of transitions P^{μ}_{T}. Thus as the trajectories in $\mathcal{D}_T$ and $\mathcal{D}_V$ are both distributed as P^{μ}_{Traj}, $F^{\mathcal{D}_T} = F^{\mathcal{D}_V} = F^{\mu}$.

Let $P_Y(\cdot|x)$ be the distribution of targets $R_t + \max_{a'} Q_m(S_{t+1}, a')$ when $(S_t, A_t) = x$, or when $S_{t+1} \sim T(\cdot|(S_t, A_t) = x)$. As $F^{\mathcal{D}_T} = F^{\mu}$, we have:

$$
\begin{aligned}
||\mathcal{B}_{\mathcal{D}_T} Q_m - f||^2_{\mathcal{D}_T} =& \mathbb{E}_{(s,a,r,s') \sim P^{\mu}} \left[\left(r + \max_{a'} Q_m(s', a') - f(s, a) \right)^2 \right] \\
=& \mathbb{E}_{x \sim P^{\mu}_{SA}, y \sim P_Y(\cdot|x)} \left[(y - f(x))^2 \right]
\end{aligned}
$$

This is just a population MSE loss function, and it is well-known that the function f minimizing this expectation satisfies $f(x) = \mathbb{E}_{y \sim P_Y(\cdot|x)}[y]$ except possibly for some set $\mathcal{X}^C$ such that $P^{\mu}_{SA}(x \in \mathcal{X}^C) = 0$ (see for example [17]). However, as $\mu(a|s) > \epsilon > 0$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$, it is clear that $P^{\mu}_{SA}(s, a) > 0$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$, and as $\mathcal{S} \times \mathcal{A}$ is countable, this means that $Pr(X = x) > 0$ for any $x \in \mathcal{S} \times \mathcal{A}$, Therefore, the only such $\mathcal{X}^C$ would be the empty set and $f(x) = \mathbb{E}[Y|X = x]$ for all $x \in \mathcal{S} \times \mathcal{A}$.

As $\mathbb{E}[Y|X] = \mathbb{E}[R_t + \max_{a'} Q_m(S_{t+1}, a')|S_t, A_t] = \mathcal{B}^* Q_m$, we thus have that $\widehat{\mathcal{B}}^* Q_m = \mathcal{B}^* Q_m$. And as $F^{\mathcal{D}_V} = F^{\mu}$, $||Q_m - \widehat{\mathcal{B}}^* Q_m||^2_{\mathcal{D}_V} = ||Q_m - \mathcal{B}^* Q_m||^2_{P^{\mu}_{SA}}$. It remains to show that $\text{argmin}_Q ||Q - \widehat{\mathcal{B}}^* Q||^2_{\mathcal{D}_V} = Q^*$, or that $\text{argmin}_Q ||Q - \mathcal{B}^* Q||^2_{P^{\mu}} = Q^*$. It is easy to show that for any random variable X with distribution P, $\mathbb{E}_{x \sim P}[X^2] = 0$ if and only if $\Pr_{X \sim P}(X = 0) = 1$. Therefore $||Q - \mathcal{B}^* Q||^2_{P^{\mu}} = 0$ if and only if $(Q - \mathcal{B}^* Q)(s, a) = 0$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ except possibly some subset $\mathcal{X}^C$ such that $\Pr_{(s,a) \sim P^{\mu}}((s, a) \in \mathcal{X}^C) = 0$. However, as discussed before, under our assumptions the only subset $\mathcal{X}^C$ that would apply here is the empty set. Therefore $||Q - \mathcal{B}^* Q||^2_{P^{\mu}} = 0$ if and only if $(Q - \mathcal{B}^* Q)(s, a) = 0$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. As Q^* is the unique fixed point of $\mathcal{B}^*$ in L_{∞} space, this means that $||Q - \mathcal{B}^* Q||^2_{P^{\mu}} = 0$ if and only if $Q = Q^*$. As $||Q - \mathcal{B}^* Q||^2_{P^{\mu}} \geq 0$ for all Q-functions Q, it thus holds that any $\text{argmin}_Q ||Q - \mathcal{B}^* Q||^2_{P^{\mu}} = Q^*$. This concludes the proof.

□

C Implementation Details of Q-Learning for Atari

There were two training configurations explored when running Q-learning: DQN (S) and DDQN (L). DQN (S) uses the standard Nature DQN architecture and training algorithm [27] and the Adam optimizer following Agarwal et al. [3]. Most of the details of this configuration can be found in Agarwal et al. [3] (they call this configuration "Offline DQN (Adam)") and thus we do not repeat them here. We did however change the batch size from 32 to 128 to speed-up training. We should also mention that our definition of "training iteration" differs from that of Agarwal et al. [3] in that it involves more weight updates to the Q-network overall (640,000 training steps instead of 250,000 training steps) as well as more frequent re-loading of data from disk into memory to reduce the correlation of the mini-batches sampled for training (a new subset of data is loaded into memory every 16,000 training steps instead of every 250,000 training steps). These changes apply to both DQN (S) and DDQN (L).

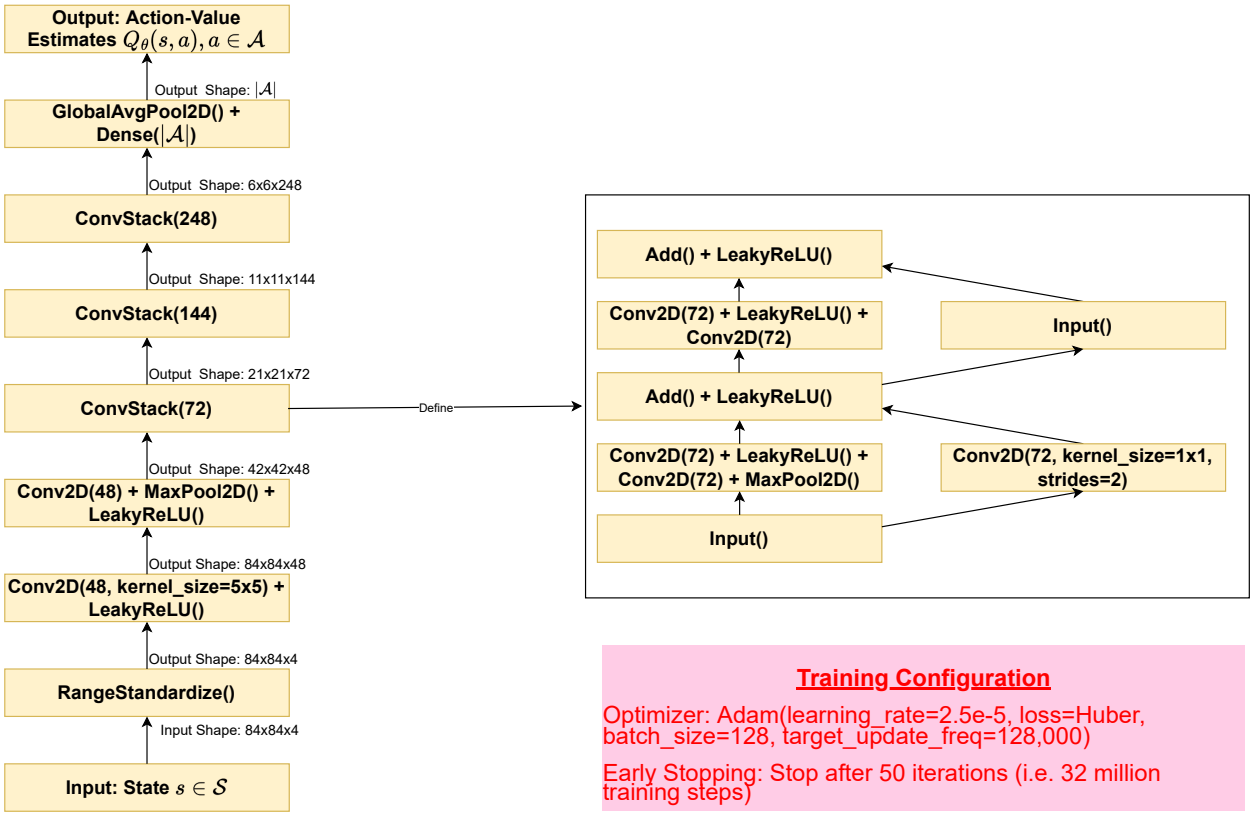

Figure C.1: Network Graph of Q-Network [DDQN (L)]. Unlabeled arrows represent feed-forward connections. Unless otherwise specified, all layers use the default parameters specified by TensorFlow [1], with the following exceptions: 1) convolutional layers use 3x3 kernels and zero padding (padding="SAME"); 2) max pooling layers use 3x3 kernels, a vertical and horizontal stride of 2 and apply zero padding (padding="SAME").

DDQN (L) uses a deeper and more complex Q-Network architecture that features 16 convolution layers as well max pooling layers, leaky ReLU activation layers [14] and residual connections [18]. The architecture here is simply the best-performing architecture for the Bellman network according to validation Bellman backup MSE that doesn't use batch normalization [20] and is described in more detail below. While there were architectures that performed better as Bellman networks, these all used batch normalization and we found that batch normalization greatly degraded Q-learning performance when added to Q-Networks. We also used double Q-learning targets [16] to reduce overestimation bias. Finally, we tweaked the learning rate and target update frequency so as to improve training stability (though otherwise the optimizer remained unchanged from the DQN (S) configuration). In theory, the learning rate and target update frequency could have been tuned by SBV as well, but to simplify and reduce the computational burden of our experiments, we just fixed these value to 2.5e-5 and 128,000, respectively.

The network graph of the DDQN (L) architecture can be found in Figure C.1. We use Leaky ReLU activation layers in place of the standard ReLU activation layers to fight issues with vanishing gradients and dying ReLUs [14]. A pre-processing layer first scales the input state $s \in \mathcal{S}$ to the $[0, 1]$ range where $\mathcal{S} = \{0, 1, ..., 255\}^{84\times84\times4}$. After the pre-processing layer, the network starts with two convolutional layers and a pooling layer, with the first convolutional layer having a kernel size of 5×5 and both convolutional layers having a filter size of 48. After this is three units which we will call *(DDQN) convolutional stacks*. Each DDQN convolutional stack consists of four convolution layers, all of which have the same number of filters, with a pooling layer inserted after the first two convolution layers. Moreover, the input to the stack is connected to the output of the pooling layer of the stack via a skip connection similar to other ResNet architectures [6], and the input to the third convolution layer is connected to the output of the final convolution layer of the stack via a skip connection as well. The three convolution layers have filter sizes 72, 144 and 248, in ascending order. After the three convolution stacks is a global average pooling layer and a dense layer with units equal to the number of possible actions $|\mathcal{A}|$. The output of this layer gives the Q-network estimates $Q_\theta(s, a), a \in \mathcal{A}$ of the true optimal action-values $Q^*(s, a), a \in \mathcal{A}$ for the given input state $s \in \mathcal{S}$.

D Implementation Details of SBV and WIS for Atari

D.1 Bellman Network Architecture, Optimizer and Regularizer

We used two training configurations for the Bellman network: A simpler configuration for Pong and a more complex one for the other three games. The more complex architecture was tuned to minimize validation MSE across the Breakout, Asterix and Seaquest datasets and performs significantly better (in terms of validation MSE) than previously-used architectures and configurations such as Nature [27] and IMPALA [9]. In theory, we could have used a different training configuration for every dataset, or even for every Bellman backup function, but we avoided doing this to simplify the experiments. It is more reminiscent of deep neural networks applied to modern image classification problems such as ImageNet [31] than the small networks employed in deep RL and includes commonly-used design choices in deep learning, such as batch normalization [20] and residual connections [18], to maximize generalization. The simpler configuration applied to Pong was utilized because it achieved almost the same performance as the more complex configuration while requiring significantly less computational resources. The more complex configuration is described below. Details of the simpler configuration are not as important and are not described in this version of the paper.

The network graph can be found in Figure D.1. As is standard in supervised deep convolutional neural networks, We follow every convolutional layer with a batch normalization layer except for those preceding pooling layers: these convolutional layers are followed first by a pooling layer and then by a batch normalization layer. We use Leaky ReLU activation layers in place of the standard ReLU activation layers to fight vanishing gradient and dying ReLU problems [14].

A pre-processing layer first scales the input state $s \in \mathcal{S}$ to the $[0, 1]$ range where $\mathcal{S} = \{0, 1, ..., 255\}^{84\times84\times4}$. After the pre-processing layer, the network starts with three convolutional layers and a pooling layer. After this is four units which we will call *(Bellman) convolutional stacks*. Each Bellman convolutional stack consists of three convolutional layers, a pooling layer and an SE block [19]. The SE block in this case consists of a global average pooling layer, a hidden dense bottleneck layer with $1/4$th as many hidden neurons as the input width (+ batch norm + leaky ReLU), and an output layer with softplus activation. The input signal is connected to the output of the SE block via a skip connection similar to other SE-ResNet architectures. After the four convolutional stacks are two convolutional layers that use depthwise separable convolutions [6] followed by an SE block. A skip connection connects the input of the first depthwise separable layer to the output of this final SE block.

The number of feature maps per layer increases with depth, increasing from 48 feature maps for the first convolutional layer to 240 for the final depthwise separable layers. Following the final SE block is a global average pooling to reduce the number of parameters and a dense layer with units equal to the number of possible actions $|\mathcal{A}|$. The output of the network $\mathcal{B}_\phi(s, a)$ is the dot product between the output of this dense layer and a one-hot transformation of the inputted action $a \in \mathcal{A}$, and is an estimate of the Bellman backup $(\mathcal{B}^* Q_m)(s, a)$ where Q_m is a candidate Q-function that we are trying to evaluate.

The network weights are trained on $\mathcal{D}_T$ to minimize mean square error via the NAdam optimization algorithm [8]. To speed up computations, we enable the mixed precision feature of TensorFlow [1] for the computations and use a batch size of 512. For the learning rate scheduler, we used an initial learning rate of 5×10^{-4} and multiplied the learning rate by a factor of 0.33 whenever training loss did not improve by at least $0.001 \times \text{var}(\mathcal{B}_\mathcal{D} Q_m(s_t, a_t)))$ over 2 consecutive iterations where $\text{var}(\mathcal{B}_\mathcal{D} Q_m(s_t, a_t))$ is the sample variance of the targets within $\mathcal{D}_T$ being used to train $\mathcal{B}_\phi$. Training was terminated after the first iteration with the learning rate equal to $5 \times 10^{-4} \times 0.33^2$. The performance of the network after each epoch was evaluated by calculating the mean squared error from validation set $\mathcal{D}_V$, and only the weights from the iteration with the lowest validation loss was saved.

D.2 Other Implementation Details of SBV

Denote Q_{θ_k} as the trained Q-network after the kth iteration for a particular configuration. We wish to estimate the Bellman backup functions $\mathcal{B}^* Q_{\theta_k}, 1 \leq k \leq 50$. Let $\mathcal{B}_{\phi_k}$ denote the trained Bellman network for estimating the $\mathcal{B}^* Q_{\theta_k}$. We perform two additional tricks to speed up computations. First, for most iterations, we initialize the trained weights of $\mathcal{B}_{\phi_{k+1}}$ as ϕ_k before conducting training, similar to Algorithm 2. This can be thought of as a kind of transfer learning and greatly speeds up computation. We do, however, occasionally re-randomize initial weights for an iteration via Hu and Xavier initialization schemes to avoid overfitting. The number of consecutive iterations I where weights are transferred from previous iterations before re-randomizing the weights is a hyperparameter that we tuned to minimize validation MSE. In our case, we set $I = 3$ for Seaquest and Breakout datasets, and $I = 1$ for Asterix datasets. The second trick we apply is stopping training once the top-5 policies according to SBV fails to change for five consecutive iterations. By stopping after it is clear that the optimal iteration has been found, we avoid wasting computational resources evaluating Q-functions that aren't going to be selected anyway.

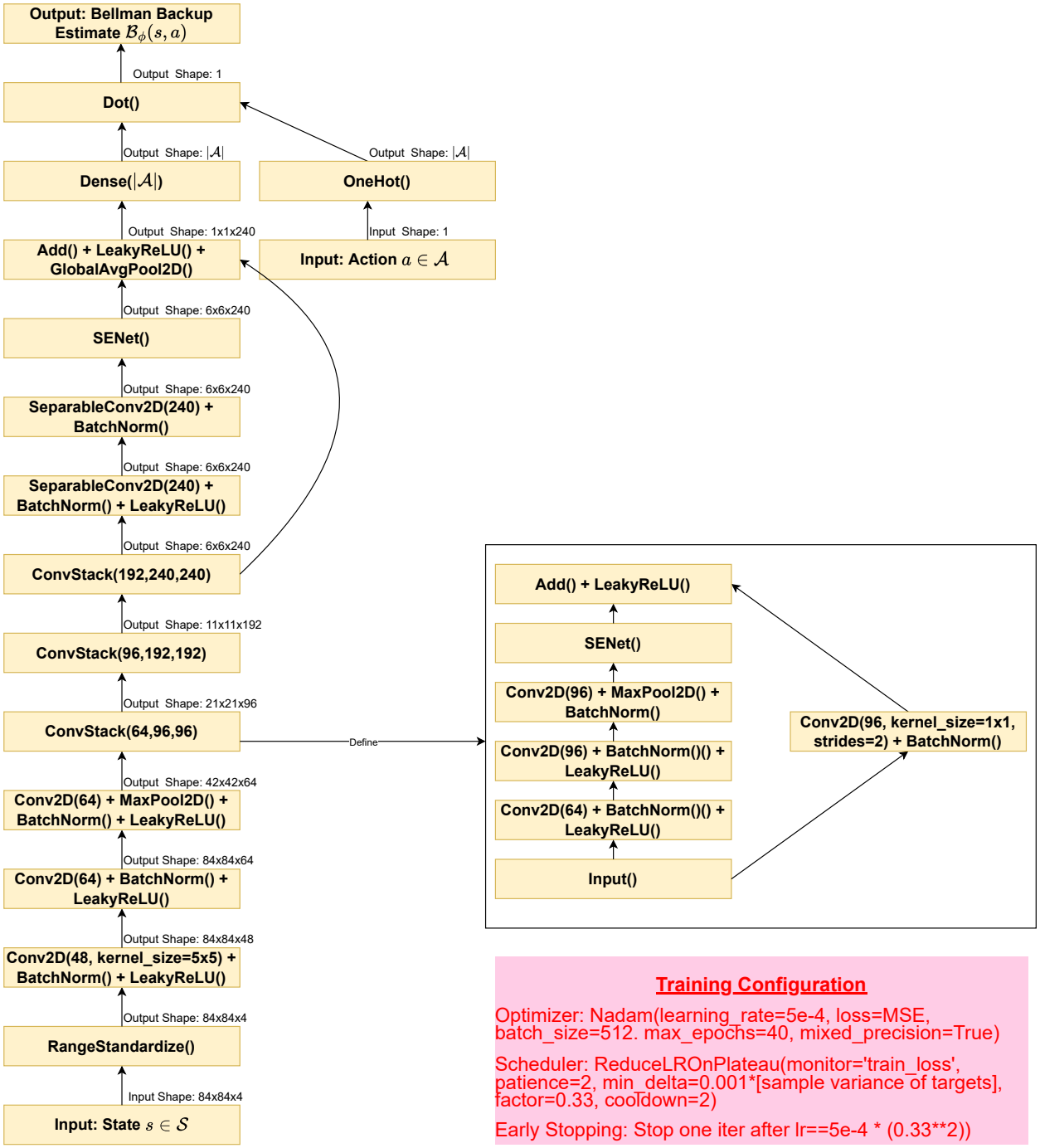

Figure D.1: Network Graph of Bellman Network. Unlabeled arrows represent feed-forward connections. Unless otherwise specified, all layers use the default parameters specified by TensorFlow [1], with the following exceptions: 1) convolutional layers use 3x3 kernels, zero padding (padding="SAME") and no bias; 2) max pooling layers use 3x3 kernels, a vertical and horizontal stride of 2 and apply zero padding (padding="SAME"); 3) SENet() uses an identical architecture to the squeeze-and-excitation units in [19], except the bottleneck layer uses a reduction factor of 4 instead of 16 and a leaky ReLU activation instead of the standard ReLU, and the output layer uses a softplus activation instead of sigmoid.

D.3 Implementation Details of WIS

As WIS requires knowledge of the unknown behavioral policy μ, we estimated it using a neural network approximator μ_β with trainable parameter vector β which we dub our *propensity network*. μ_β was trained on $\mathcal{D}_T$ so as to maximize log-likelihood $\sum_{(a,s)\in\mathcal{D}}[\log \mu_\theta(a|s)]$ while its log-likelihood on $\mathcal{D}_V$ was used to tune hyperparameters and evaluate performance. We ended up using the same configuration for the propensity network as that of the Bellman network (Figure D.1), minus the obvious changes (e.g. MSE loss replaced with categorical cross-entropy loss). We found that this achieved higher validation log-likelihood than other custom configurations explored as well as architectures and configurations used in previous literature such as IMPALA [9].

E Conditions for SBV to Perform Sufficiently

As we've discussed throughout the paper, model selection algorithms based on the MSBE such as the EMSBE will not perform well if they do not accurately estimate the MSBE. For this reason, SBV performance is dependent on estimating $||Q_m - \mathcal{B}^* Q_m||^2_{P^\mu}$, and specifically $\mathcal{B}^* Q_m$, with sufficient accuracy for all $1 \leq m \leq M$. Such algorithms will also fail if the MSBE itself is a poor model selector. For example, for MSBE to perform well as a model selector, we require there to be sufficient diversity in the observed state-action pairs (see the proof for Theorem 3 given in section B). Theorem 1 and our empirical investigations also suggest that for the MSBE to select high-quality policies, we require that our candidate set $\mathcal{Q}$ include estimators of Q^* are sufficiently accurate (i.e. have sufficiently small Bellman errors). To clarify, it is acceptable for $\mathcal{Q}$ to include mostly inaccurate estimates, so long as at least some estimates in $\mathcal{Q}$ are accurate.

For example, when excluding Q-functions generated by DDQN (L) and analyzing only Q-functions generated from DQN (S), SBV fails to properly select the best-performing policies for three of four environments analyzed. However, once Q-functions generated by DDQN (L) are included in the candidate set, SBV not only correctly identifies the Q-functions generated by DDQN (L) as better, but it is also effective at tuning the number of training iterations for DDQN (L) (i.e. applying early stopping). The issue here is that the Q-network trained by DQN (S) is quite inaccurate at every training iteration, whereas DDQN (L) estimates Q^* with much better accuracy provided we stop training early enough. Additional experiments on toy environments further suggest that the MSBE performs poorly when all Q-functions being evaluated have poor accuracy, even if some of the associated policy values are large (results omitted for this version of the paper).

It is also worth noting that while the candidate value functions need to include estimators with reasonable accuracy, the estimation does not have to be perfect either. For example, state-of-the-art online RL methods have managed to achieve policies with returns of over 100,000 for Seaquest [22]. As the Q-functions generated by DDQN (L) for Seaquest give values of less than 25,000, they must have decent estimation error. Despite this, they are accurate enough for SBV to select a high-performing policy.

In real-world settings, we can ensure that our candidate set includes sufficiently accurate estimates by exploring a sufficiently large number of training configurations and/or by exploring training configurations which have been shown in previous work to achieve robust performance in a variety of scenarios.