Value Bonuses using Ensemble Errors for Exploration in Reinforcement Learning

Abdul Wahab, Raksha Kumaraswamy, Martha White

Keywords: Reinforcement Learning, Exploration, Value bonuses, Ensembles, Uncertainty estimates

Summary

Optimistic value estimation can be useful to direct exploration and improve sample efficiency in Reinforcement Learning. Despite many such methods in literature, simpler, undirected approaches like ϵ-greedy still continue to be widely used. One potential reason for this is that many existing methods can be onerous to use as they may not be compatible with the base learning algorithm, or can be hard-to-use as many design choices need to be made to make them effective in practice. This paper proposes a simple approach to address these limitations. Building on ideas that utilize an ensemble for optimistic value estimation, this work proposes an algorithm called Value Bonuses using Ensemble Errors (VBE) that is easy to use and compatible with any base Reinforcement Learning algorithm, with a small additional computational foot-print. VBE's similarity and difference to existing approaches is discussed, and the algorithm is evaluated extensively. Our code is available at: https://github.com/mirzaabdulwahab1612/VBE

Contribution(s)

1. Proposes a new approach for estimating Value Bonuses using Ensemble Errors that allows for first-visit optimism and deep exploration.
   Context: Many prior works utilize the idea of *value bonuses* to estimate optimistic values for exploration (Osband et al., 2019). Typically, many estimate an additional value function that propagates *reward bonuses* in order to estimate the value bonus (Burda et al., 2019). This work proposes a variant of value bonuses that does not rely on propagating additional reward bonuses; this allows for desirable features like first-visit optimism. We show our framework allows for Optimistic Initial Values with high probability. Additionally, the value bonuses have a similar timescale of learning as the main value function, therefore, potentially allowing for deep exploration.
2. Provides insight into how our proposed value bonuses are similar to and different from some relevant widely-used approaches.
   Context: In specific, we contrast how the proposed bonuses capture MDP-specific properties like transition dynamics, unlike those of RND (Burda et al., 2019). Additionally, we highlight the similarity between quantities estimated by the proposed algorithm and BDQN (Osband et al., 2019), despite the difference that BDQN utilizes a Thompson sampling approach to induce optimism.
3. Empirically evaluate the utility of the proposed value bonuses for inducing exploration in classic-control problems, and demonstrate scalability through Atari.
   Context: We demonstrate how existing methods lack first-visit optimism in a controlled setting designed to test state coverage. We show that the proposed algorithm can extend to more complex environments like Atari without design choices that alter the underlying algorithm.

Value Bonuses using Ensemble Errors for Exploration in Reinforcement Learning

Abdul Wahab[1], **Raksha Kumaraswamy**[2], **Martha White**[1,3]
{wahab1, whitem}@ualberta.ca, raksha.kumaraswamy@sony.com

[1]**Department of Computing Science, University of Alberta, Canada**
[2]**Sony AI**
[3]**Alberta Machine Intelligence Institute (Amii)**

Abstract

Optimistic value estimates provide one mechanism for directed exploration in reinforcement learning (RL). The agent acts greedily with respect to an estimate of the value plus what can be seen as a *value bonus*. The value bonus can be learned by estimating a value function on *reward bonuses*, propagating local uncertainties around rewards. However, this approach only increases the value bonus for an action retroactively, after seeing a higher reward bonus from that state and action. Such an approach does not encourage the agent to visit a state and action for the first time. In this work, we introduce an algorithm for exploration called Value Bonuses with Ensemble errors (VBE), that maintains an ensemble of random action-value functions (RQFs). VBE uses the errors in the estimation of these RQFs to design value bonuses that provide first-visit optimism and deep exploration. The key idea is to design the rewards for these RQFs in such a way that the value bonus can decrease to zero. We show that VBE outperforms Bootstrap DQN and two reward bonus approaches (RND and ACB) on several classic environments used to test exploration and provide demonstrative experiments that it can scale easily to more complex environments like Atari. We provide code for VBE here.

1 Introduction

A typical approach to incorporate exploration into a value-based reinforcement learning (RL) agent is to obtain optimistic value estimates. The agent takes greedy actions according to this optimistic value estimate, leading it to take actions that look good either because they have high uncertainty or because the action is actually high value. This approach has been well-developed for the contextual bandit setting, with a variety of algorithms and theoretical results on optimality (Li et al., 2010; Abbasi-Yadkori et al., 2011). Understanding is growing about how to soundly extend these ideas to reinforcement learning, though the theoretical results on estimating and using optimistic values are limited to the linear function approximation setting (Grande et al., 2014; Osband et al., 2016a; Abbasi-Yadkori et al., 2019; Wang et al., 2019).

Though the theory is difficult to extend, there has been a concerted effort to develop and empirically evaluate such optimistic value estimation approaches for the deep RL setting. Bootstrap DQN with priors, for example, maintains an ensemble of action-values, which reflect uncertainty in the value estimates (Osband et al., 2018; 2019). It takes a Thompson sampling approach—which can be seen as optimistic—by sampling one action-value in the ensemble and following it for an entire episode. Another common approach to obtain optimistic value estimates employs the usage of *reward bonuses* (Bellemare et al., 2016; Ostrovski et al., 2017; Burda et al., 2019; Ash et al., 2022). A reward bonus, reflecting uncertainty with respect to the transition, is added to the reward, increasing the estimated value proportionally for the corresponding states and action.

Most works, however, eschew these directed exploration approaches in favor of simpler, undirected exploration approaches like ϵ-greedy. One potential reason for this is that reward bonus approaches do not encourage *first-visit optimism*. They encourage revisiting a state, if the reward bonus was high in that state; namely, they retroactively reason about uncertainty of states they have seen. The reward bonus cannot encourage visiting a state for the first time. Bootstrap DQN with priors (BDQN), on the other hand, does not have this issue, using fixed additive priors to provide first-visit optimism. Unlike reward bonuses, though, BDQN is more onerous to use. It requires completely changing the algorithm to one that maintains and updates an ensemble, and making key choices like how often to follow one of the value functions in the ensemble before switching. Recent work suggests it is key to have a large ensemble for BDQN (Janz et al., 2019; Osband et al., 2023). Epinets (Osband et al., 2023) can match the performance of BDQN with much less compute, but are arguably even more onerous to implement than BDQN. Our goal is to develop an easy-to-use exploration approach for deep RL, that can easily be added to an existing algorithm, making it less onerous to displace the default ϵ-greedy approach.

To do so, we explore how to directly estimate a *value bonus*. The agent acts greedily according to the value estimate plus this separate value bonus b, namely $\operatorname{argmax}_a q(s,a) + b(s,a)$. The value bonus should ideally represent the uncertainty for that state and action. Though this may be the first time this term is used,[1] there are some works that estimate value bonuses. One simple approach is to separate out the reward bonuses and learn them with a second value function, as was proposed for RND (Burda et al., 2019) and later adopted by ACB (Ash et al., 2022). This approach, however, still suffers from the fact that reward bonuses are only retroactive, and the resulting b is unlikely to be high for unvisited states and actions. For the contextual bandit setting, the ACB algorithm actually directly estimates the value bonus using the maximum over an ensemble of functions, which is high for unvisited states and actions; but the extension to deep RL with reward bonuses loses this first-visit optimism. UCLS (Kumaraswamy et al., 2018) and UBE (O'Donoghue et al., 2018; Janz et al., 2019) both directly estimate value bonuses, but are limited to linear function approximation. Dora (Choshen et al., 2018) uses value bonuses that are inversely proportional to visitation counts, which is again difficult to extend to the general function approximation setting.

In this work, we introduce a new approach to obtain value bonuses for reinforcement learning, with an algorithm we call Value Bonuses with Ensemble errors (VBE). Similarly to ACB, we use a maximum over an ensemble, but directly use that maximum as the value bonus, rather than indirectly through reward bonuses. The idea is to sample a random action-value function (RQF)—such as a random neural network—and extract the implicit random reward function underlying this RQF target. The RQF predictor in the ensemble is updated using temporal difference learning on this random reward. Because the RQF target is sampled from the same function class as the RQF predictor, the error can eventually reduce to zero, allowing the value bonus to shrink to zero. These value bonuses are learned separately from the main action-values, and so can be layered on top of many algorithms. In our experiments, for example, we simply use Double DQN (Van Hasselt et al., 2016), and modify the step where the agent selects an action from ϵ-greedy to instead taking the greedy action in the value estimate plus the value bonus. We show that this simple approach is an effective, and scalable method for exploration that improves sample efficiency of learning in a range of domains: from hard exploration gridworlds, to image-based Atari domains.

2 Background

We focus on the problem of an agent learning optimal behaviour in an environment, whose interaction process is modelled as a Markov Decision Process (MDP). A MDP consists of $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$ where $\mathcal{S}$ is the set of states; $\mathcal{A}$ is the set of actions; $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty)$ provides the transition probabilities; $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function; and $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition-based discount function which enables either continuing or episodic problems to be

[1] Usually, b would be called a confidence interval, with $q(s,a) + b(s,a)$ an upper confidence bound. However, we do not use that term here, because for the heuristics we use, it is not clear we get a valid upper confidence bound. Instead, it is a bonus added to the value when deciding which action looks promising.

specified (White, 2017). At each step, the agent selects action A_t in state S_t, and transitions to S_{t+1}, according to P, receiving reward $R_{t+1} \stackrel{\text{def}}{=} r(S_t, A_t, S_{t+1})$ and discount $\gamma_{t+1} \stackrel{\text{def}}{=} \gamma(S_t, A_t, S_{t+1})$.

For a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, \infty]$, the value for taking action a in state s is the expected discounted sum of future rewards, with actions selected according to π in the future,

$$q^\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma_{t+1} q^\pi(S_{t+1}, A_{t+1}) \middle| S_t = s, A_t = a \right]$$

where $\mathbb{E}_\pi$ means that actions are selected according to π in the expectation. The policy π can be progressively improved by making it greedy in $q^\pi(s, a)$, then updating the action-values for the new policy, then repeating until convergence.

In practice, these steps are approximated. The action-values q^π are approximated using q_w parameterized by $w \in \mathcal{W} \subset \mathbb{R}^d$. One algorithm to estimate q_w is Double DQN (DDQN). DDQN is an off-policy algorithm, meaning that it uses a different behavior policy π_b to select actions from the policy it evaluates, which is greedy in q_w. This algorithm uses a target network $q_{\tilde{w}}$ for bootstrapping, giving the following update for one transition (s, a, r, s', γ):

$$w \leftarrow w + \eta \delta \nabla q_w(s, a) \quad \text{for } \delta \stackrel{\text{def}}{=} r + \gamma q_{\tilde{w}}(s', \operatorname*{argmax}_{a'} q_w(s', a')) - q_w(s, a) \tag{1}$$

The behavior policy is typically defined to be ϵ-greedy in q_w, but can be any policy that promotes exploration. In this work, we consider an alternative choice for the behavior policy: one that uses a value bonus b, $\pi_b(s) = \operatorname{argmax}_a q_w(s, a) + b(s, a)$. The value bonus should reflect uncertainty in the action-value estimate, encouraging the behavior policy to take an action in a state if it has high uncertainty. It might have high uncertainty if (s, a) is quite different from what it has seen before—meaning it has never been visited—or because the agent has not yet visited it sufficiently often to be certain about its value. The focus of this work is a new approach for obtaining b for the deep RL setting.

3 Value Bonuses with Ensemble Errors

In this section, we first motivate why we use an ensemble of value functions, rather than simply using supervised learning for the ensemble. We then discuss how to appropriately define the rewards for the ensemble value functions, and contrast the unique property of the bonuses produced by these ensemble of value functions.

The most straightforward approach to get an error from an ensemble is to use a random target, as is done in RND. For an ensemble of size k, we can generate random neural networks $f_1, \ldots, f_k$ and update the learned functions $\hat{f}_1, \ldots, \hat{f}_k$ in the ensemble using a squared error: for each (s, a), update each $\hat{f}_i$ using loss $(f_i(s, a) - \hat{f}_i(s, a))^2$. The value bonus for any (s, a) can be set to

$$b(s, a) \doteq \max_{i \in [k]} |\hat{f}_i(s, a) - f_i(s, a)| \tag{2}$$

Ciosek et al. (2020) show that fitting random prior functions serve as a computationally tractable approach towards estimating uncertainty in the supervised learning setting. Unfortunately, in the reinforcement learning setting, this is likely to concentrate too quickly, and will not do what has been called deep exploration (Osband et al., 2019). We want the agent to reason not just about uncertainty for this state and action, but also about the uncertainty of the state that it leads into.[2]

Instead, we want an ensemble of value functions that are more likely to promote deep exploration. More specifically, we want to generate random rewards r_i for each f_{w_i}, where the f_{w_i} are updated using standard temporal difference learning bootstrapping approaches. We want the learning dynamics for these value functions to resemble the primary value function, so that they learn at a similar timescale and are more likely converge to zero once the primary value function has also converged.

[2]Note that RND do not use these errors directly for exploration. Instead, they used them as reward bonuses, which can retroactively promote deep exploration, with the issue that they do not promote first-visit optimism.

We need to define rewards and target functions that are consistent with each other and that allow us to easily measure the errors. Consider if we again do the simplest thing: generate a random neural network r_i for each f_{w_i}. Let us assume for now that we have a fixed policy, π. First, it is not clear how we would actually measure the error since we do not know the true value function f_i, namely the expected return using r_i under policy π. Further, this true value function may not be representable by f_{w_i}.

Instead, our proposed approach is to generate a random action-value function (RQF) f_i, and then define rewards consistent with that f_i. Define the stochastic ensemble reward from (S_t, A_t) to be

$$R_{i,t+1} \stackrel{\text{def}}{=} f_i(S_t, A_t) - \gamma_{t+1} f_i(S_{t+1}, A_{t+1}), \tag{3}$$

where $A_{t+1} \sim \pi(\cdot|S_{t+1})$ and $\gamma_{t+1} \stackrel{\text{def}}{=} \gamma(S_t, A_t, S_{t+1})$ is defined by the environment. Further, by definition, the action-values of the random prediction function is:

$$q_i^\pi(s,a) \stackrel{\text{def}}{=} \mathbb{E}_\pi \left[R_{i,t+1} + \gamma_{t+1} q_i^\pi(S_{t+1}, A_{t+1}) \middle| S_t = s, A_t = a \right]. \tag{4}$$

We show in the following proposition that $q_i^\pi = f_i$.

Proposition 1 *For all $i \in [k]$, we have $q_i^\pi = f_i$.*
Proof:

$$\begin{aligned}
q_i^\pi(s,a) &= \mathbb{E}_\pi \left[R_{i,t+1} + \gamma_{t+1} q_i^\pi(S_{t+1}, A_{t+1}) \middle| S_t = s, A_t = a \right] \\
&= \mathbb{E}_\pi \left[R_{i,t+1} + \gamma_{t+1} R_{i,t+2} + \gamma_{t+1}\gamma_{t+2} q_i^\pi(S_{t+2}, A_{t+2}) \middle| S_t = s, A_t = a \right] \\
&= \mathbb{E}_\pi \Big[[f_i(s,a) - \gamma_{t+1} f_i(S_{t+1}, A_{t+1})] + \gamma_{t+1}[f_i(S_{t+1}, A_{t+1}) - \gamma_{t+2} f_i(S_{t+2}, A_{t+2})] \\
&\qquad + \gamma_{t+1}\gamma_{t+2} q_i^\pi(S_{t+2}, A_{t+2}) \Big| S_t = s, A_t = a \Big] \\
&= \mathbb{E}_\pi \Big[[f_i(s,a) \underbrace{- \gamma_{t+1} f_i(S_{t+1}, A_{t+1})]}_{\text{cancels}} \underbrace{+ \gamma_{t+1} f_i(S_{t+1}, A_{t+1})}_{\text{cancels}} \\
&\qquad - \gamma_{t+1}\gamma_{t+2} f_i(S_{t+2}, A_{t+2})] + \gamma_{t+1}\gamma_{t+2} q_i^\pi(S_{t+2}, A_{t+2}) \Big| S_t = s, A_t = a \Big].
\end{aligned}$$

We can keep unrolling this, and these terms will continue to telescope, leaving only the first term $f_i(s,a)$, completing the proof. ■

Therefore, updating f_{w_i} with rewards r_i should converge to q_i^π—and so to f_i—because f_i is in the function class of f_{w_i}. This convergence ensures the value bonuses go to zero, which is desired if we want the agent to stop exploring and converge to the greedy policy. Even with a fixed policy, however, this convergence will only occur under certain conditions. Primarily, the failure would be that f_{w_i} gets stuck in a local minima or even that it diverges, due to know issues with temporal difference (TD) learning algorithms combined with neural networks and with off-policy update.

There is fortunately a large (and growing) literature understanding the convergence behavior of TD algorithms. Under linear function approximation, we know least-squares TD converges at a rate of $1/\sqrt{T}$ to the global solution, even under off-policy sampling (Tagorti & Scherrer, 2015). With the advent of theory for overparameterized networks, TD with a particular neural network function class has been shown to converge to the global solution, under on-policy sampling (Cai et al., 2019). In general, we know that a class of modified TD algorithms, called gradient TD methods, converge even under off-policy sampling and nonlinear function approximation (Dai et al., 2017; Patterson et al., 2022). Convergence under off-policy sampling is key in our setting, because the behavior policy is optimistic but the target policy may be greedy. We expect that under certain conditions on the neural network it might be possible to say that these gradient TD methods converge to global solutions, though to the best of our knowledge, no such work yet exists. We provide a more complete discussion in Appendix 6 of how this existing theory on convergence of TD applies to our setting.

3.1 Bonuses that reflect MDP-specific properties

While the form of the bonus proposed above in Equation 2 looks similar in principle to RND, the main difference as mentioned previously is that we consider the ensemble to be composed of random value functions, in contrast to methods like RND which consider the ensemble to be composed of random functions. This simple change provides an interesting property to the bonuses derived from this ensemble: they can reflect the stochasticity in the transition dynamics of the MDP.

Proposition 2 *Let* $\hat{f}_i(s,a) \stackrel{\text{def}}{=} \mathbb{E}[R_{i,t+1} + \gamma_{t+1}\hat{f}_i(S_{t+1}, A_{t+1})|S_t = s, A_t = a] + \epsilon_i(s,a)$*, where* $\epsilon_i(s,a)$ *denotes the Bellman error for* (s,a) *in* $\hat{f}_i$*. Then*

$$b(s,a) = \max_{i\in[k]} \big|\mathbb{E}[\gamma_{t+1}(\hat{f}_i(S_{t+1}, A_{t+1}) - f_i(S_{t+1}, A_{t+1}))|S_t = s, A_t = a] + \epsilon_i(s,a)\big|.$$

Proof: By the assumption that f_i is a value-function, we know that

$$f_i(s,a) \stackrel{\text{def}}{=} \mathbb{E}[R_{i,t+1} + \gamma_{t+1} f_i(S_{t+1}, A_{t+1})|S_t = s, A_t = a].$$

Plugging the definitions into Equation 2, we get

$$\begin{aligned}
b(s,a) &= \max_{i\in[k]} |\hat{f}_i(s,a) - f_i(s,a)| \\
&= \max_{i\in[k]} \big|\mathbb{E}[R_{i,t+1} + \gamma_{t+1}\hat{f}_i(S_{t+1}, A_{t+1})|S_t = s, A_t = a] + \epsilon_i(s,a) \\
&\qquad - \mathbb{E}[R_{i,t+1} + \gamma_{t+1} f_i(S_{t+1}, A_{t+1})|S_t = s, A_t = a]\big| \\
&= \max_{i\in[k]} \big|\mathbb{E}[\gamma_{t+1}(\hat{f}_i(S_{t+1}, A_{t+1}) - f_i(S_{t+1}, A_{t+1}))|S_t = s, A_t = a] + \epsilon_i(s,a)\big|.
\end{aligned}$$

■

4 Using the Ensemble of Value Functions

We provide pseudocode in Algorithm 1, for the case where the base algorithm is Double DQN, but it is possible to swap in many different off-policy value-based algorithms. Even actor-critic, which explicitly maintains a critic q_w, could easily incorporate the value bonuses by using an optimistic critic. For the purposes of this paper, however, we restrict our focus to Double DQN.

The ensemble value functions are updated on the same target policy as Double DQN, namely the greedy policy in q_w. This choice comes from the fact that we want to understand uncertainty in the values for the target policy. The update is similar to Double DQN, except the actions are sampled according to q_w rather than f_{w_i}, and we use the ensemble reward r_i defined above in Equation (3):

$$w_i \leftarrow w_i + \eta\delta_i \nabla f_{w_i}(s,a) \quad \text{for } \delta \stackrel{\text{def}}{=} r_i + \gamma f_{\tilde{w}_i}(s', \operatorname*{argmax}_{a'} q_w(s',a')) - f_{w_i}(s,a) \tag{5}$$

On each step, we only update one RQF predictor. Updating the entire ensemble is expensive, and arguably unnecessary. There are multiple ways to control the magnitude of the value bonus, and how quickly it decays. One way is the size of the ensemble, where the larger the ensemble, the more slowly this bonus should decay. Updating each RQF predictor less frequently, however, will also cause the bonus to decay more slowly. It both allows us to make the ensemble smaller, and ensure that regardless of the ensemble size, the computation per-step is simply double that of Double DQN: one update to the main value function and one update to an RQF predictor.

4.1 Guaranteeing Optimistic Initial Values

In this section we show the ensemble size and bonus scale c can be set to obtain optimistic initial values with high probability. The result motivates that value bonuses with ensemble errors provide sufficient first-visit optimism.

Algorithm 1 Value Bonuses with Ensemble Errors (VBE)

Parameters: ensemble size k, bonus scale c, target net update frequency τ, batch size m
Initialize empty buffer: $B \leftarrow \emptyset$, action-value function: q_w, target RQFs: $f_i, \ldots f_k$, predictor RQFs: $f_{w_1}, \ldots, f_{w_k}$, and target networks: $q_{\tilde{w}}, f_{\tilde{w}_1}, \ldots, f_{\tilde{w}_k}$
Optimistic behavior policy:
$\pi_b(s) \leftarrow \text{argmax}_{a \in \mathcal{A}} q_w(s,a) + c\, b(s,a)$
where $b(s,a) \leftarrow \max_{i \in [k]} |f_{w_i}(s,a) - f_i(s,a)|$
Get the initial state s_0
for environment interactions $t = 0, 1, \ldots$ **do**
  Take action $a \leftarrow \pi(s_t)$ and observe $r_{t+1}, s_{t+1}, \gamma_{t+1}$
  Add $(s_t, a_t, r_{t+1}, s_{t+1}, \gamma_{t+1})$ to the buffer B
  // Update action-values using DDQN update
  Sample mini-batch from B, update q_w using Eq. (1)
  // Update one randomly selected RQF
  Sample i from $[k]$ uniform randomly
  Sample mini-batch from B, update f_{w_i} using Eq. (5) where
  for each (s, a, r, s', γ) replace r with
  $r_i \stackrel{\text{def}}{=} f_i(s,a) - \gamma f_i(s', \text{argmax}_{a' \in \mathcal{A}} q_w(s', a'))$
  if $t + 1 \mod \tau == 0$ **then**
    $\tilde{q}_w \leftarrow q_w$ and for all i, $f_{\tilde{w}_i} \leftarrow f_{w_i}$
  end if
end for

We assume that we initialize our action-values q and target and predictor RQFs with a standard random neural network initialization. Given a fixed initialization, we can jointly reason about the deep setting and the linear setting. In the linear setting, for a given state, we have n-dimensional features $\phi(s), \phi_1, \ldots, \phi_k(s) \in \mathbb{R}^n$ for the q and RQFs respectively. In the deep setting, the last layer of the fixed neural network specifies the features. We additionally assume that the features from this last layer are all normalized, $\|\phi_i(s)\|_2 = 1$ for all s, to simplify the proof; but this is not strictly necessary and a similar result can be obtained without normalization.

Assumption 1 *The feature functions $\phi(s), \phi_1, \ldots, \phi_k(s) \in \mathbb{R}^n$ are all unit length—have an ℓ_2 norm of 1 for every state. All weights are iid sampled from $\mathcal{N}(0, 1/n)$, to get initializations $q(s,a) = \phi(s)^\top w_a$, $f_i(s,a) = \phi_i(s)^\top w^*_{a,i}$ and $f_{w_i}(s,a) = \phi_i(s)^\top w_{a,i}$*

Proposition 3 *Let $z(\delta)$ be the z-value for a Gaussian $X \sim \mathcal{N}(0,1)$ where Pr$(X > z(\delta)) = 1 - \delta$ for $\delta \in (0,1)$. Take any $q_{max} \in \mathbb{R}$ and any $\delta \in (0,1)$. Then for any s, a, if*

$$c \geq \sqrt{\tfrac{n}{\pi}} \left(q_{max} - z(\delta/2)/\sqrt{n}\right) / \left(\log(k/2) - \log\log(2/\delta)\right)$$

then

$$q(s,a) + c \max_{i \in 1,\ldots,k} |f_i(s,a) - f_{w_i}(s,a)| > q_{max}$$

with probability $1 - \delta$.

Proof: Pick any $s \in \mathcal{S}, a \in \mathcal{A}$. Notice that because $w^*_{a,i,j} \sim \mathcal{N}(0, 1/n)$ for all $j \in \{1, \ldots, n\}$, that the linear combination of these Gaussians $f_i(s,a) = \phi_i(s)^\top w^*_{a,i}$ is distributed as $\mathcal{N}(0, \|\phi_i(s)\|_2^2/n) = \mathcal{N}(0, 1/n)$ because $\|\phi(s)\|_2^2 = 1$. Define $\delta_i \doteq f_i(s,a) - f_{w_i}(s,a) = \phi_i(s)^\top w^*_{a,i} - \phi_i(s)^\top w_{a,i}$, which is the difference of two $\mathcal{N}(0, 1/n)$ random variables and itself is again Gaussian with double the variance, $\mathcal{N}(0, \frac{2}{n})$. We can rewrite $\delta_i = \sqrt{\frac{2}{n}} Y_i$ where $Y_i \sim \mathcal{N}(0,1)$. Then we can leverage the result from (Ash et al., 2022, Lemma 1) that gives a lower bound on the

maximum of the absolute value of k standard normal Gaussians to get that with probability $1-\delta/2$

$$\begin{aligned}
&\max_{i\in\{1,\ldots,k\}} |\delta_i| = \sqrt{\tfrac{2}{n}} \max_{i\in\{1,\ldots,k\}} |Y_i| \\
&\geq \sqrt{\tfrac{2}{n}}\sqrt{\tfrac{\pi}{2}}\sqrt{\log(k/2) - \log\log(2/\delta)} \\
&= \sqrt{\tfrac{\pi}{n}}\sqrt{\log(k/2) - \log\log(2/\delta)}
\end{aligned}$$

Similarly to the RQFs, because $w_{a,j} \sim \mathcal{N}(0, 1/n)$, we also have that $q(s,a) = \phi(s)^\top w_a$ is Gaussian with distribution $\mathcal{N}(0, \|\phi(s)\|_2^2/n) = \mathcal{N}(0, 1/n)$. We can therefore say with probability $1-\delta/2$ that $q(s,a) > z(\delta/2)/\sqrt{n}$. Taking the union over these two high probability events, we get that with probability $1-\delta$,

$$\begin{aligned}
&q(s,a) + c \max_{i\in 1,\ldots,k} |f_i(s,a) - f_{w_i}(s,a)| > \\
&z(\delta/2)/\sqrt{n} + c\sqrt{\tfrac{\pi}{n}}\sqrt{\log(k/2) - \log\log(2/\delta)}
\end{aligned}$$

where plugging in the above c makes this second term equal q_{max}, giving us the desired result. ■

We can also ask what happens to the bonuses in VBE after initialization. Ideally, they eventually converge to zero, with the action-values converging and the behavior and target policies both converging to a greedy policy. This scenario goes beyond the convergence conditions discussed above in Section 3 for fixed policies. In VBE, both our behavior policy and target policy are changing with time. Unfortunately, theory around TD does not address this scenario. There are some results for a fixed behavior policy for double Q-learning under linear function approximation (Zhao et al., 2021), or for a variant of DQN with a fixed dataset (Wang & Ueda, 2022). The issue with a changing behavior policy is that it changes the relative importance of states in the objective, and so the best value function may change as it changes how it trades off errors across states. In our realizable setting, this changing importance may be less important, because our RQF predictor can perfectly represent the target. In our own experiments, we found the value bonuses did always converge to zero. Nonetheless, we know of no theory that would allow us to guarantee this.

Connection to BDQN: Though not obvious at first glance, there is a connection between RQFs and random prior functions in BDQN. In BDQN, the value function is $q_\theta = f_\theta + cp$ for a random prior function p that is not updated, prior scale c, and a learned function f_θ. Random priors were developed for stationary state distributions—though then applied to control—so let us consider the update for a fixed policy π. The update uses $a' \sim \pi(\cdot|s')$, giving

$$r - c\underbrace{(p(s,a) - \gamma p(s',a'))}_{\text{RQF's reward}} + \gamma f_\theta(s',a') - f_\theta(s,a)$$

This is a standard update with reward bonus $c(p(s,a) - \gamma p(s',a'))$, and this bonus is a scaled negation of our reward in Equation (3). With a fixed policy, we can separate the value function learning into q^π that estimates the values for the rewards and b^π that estimates the values for the reward bonuses. Namely, f_θ consists of $q^\pi + b^\pi$. As these functions converge, $b^\pi(s,a)$ approaches $-cp(s,a)$ using the exact same argument to the one in our Proposition 1, just negating the function p. Consequently, $f_\theta(s,a) + cp(s,a) = q^\pi(s,a) + b^\pi(s,a) + cp(s,a) = q^\pi(s,a) + (b^\pi(s,a) + cp(s,a))$ goes to q^π since $b^\pi(s,a) + cp(s,a)$ eventually cancels.

This argument is not how randomized priors are presented, but provides another intuitive interpretation. Further, it highlights a key difference between BDQN and VBE: BDQN takes a Thompson sampling approach to induce optimism, whereas VBE acts greedily with respect to optimistic value estimates. Another key point is that BDQN's prior-based bonus is scaled by the c parameter. The prior-based bonus can be seen as adding a fixed noise to the targets in the updates. Scaling the bonus term with c would increase the variance in the targets by a factor of c^2. This can make the optimization problem harder and may cause BDQN to be sensitive towards higher learning rates. Unlike BDQN, VBE does not incorporate the c parameter when estimating the RQFs, and only uses the bonus scale in the behavior policy.

5 Experiments

We evaluate our proposed algorithm on four classic exploration environments and six Atari environments, particularly in comparison to BDQN and the reward bonus approaches ACB and RND. We first investigate the algorithms in a pure exploration setting, on Deepsea, where we evaluate state coverage. Then we compare performance on the classic environments, and conclude with experiments in Atari, highlighting that VBE scales successfully to this setting. We provide more analysis of VBE's sensitivity to its two parameters – ensemble size, and bonus scale in Appendix E.

5.1 Experimental Settings

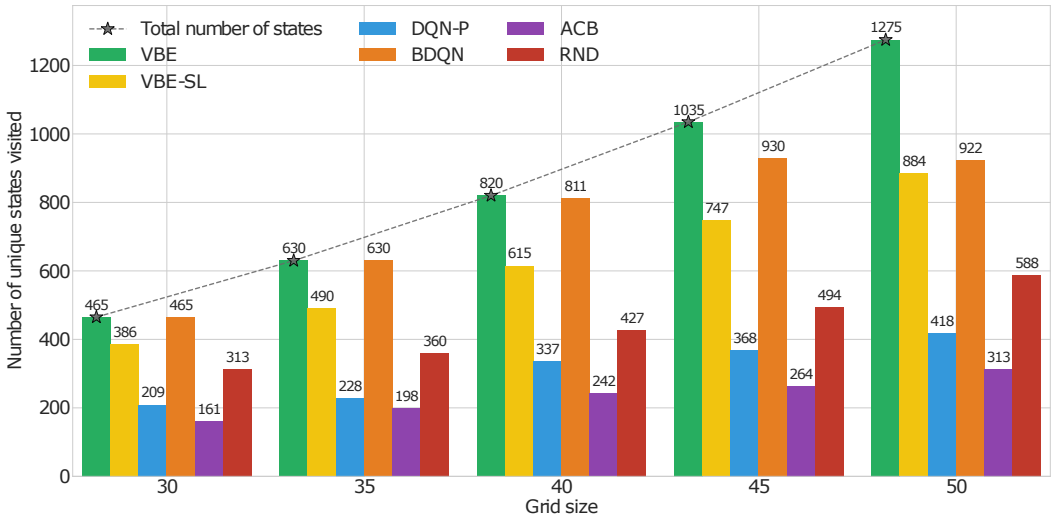
The four classic exploration environments are Sparse Mountain Car, Puddle World, River Swim and Deepsea. Full details are in Appendix A, but we list a few key details here. Mountain Car has two-dimensional continuous inputs with sparse rewards: the agent only receives a reward of 1 at the goal and 0 otherwise. Puddle World also has two-dimensional continuous inputs, noisy actions and highly negative rewards in puddles along the way to the goal. River Swim resembles a problem where a fish tries to swim upriver, with high reward (+1) upstream which is difficult to reach and, a lower but still positive reward (+0.005), which is easily reachable downstream. This environment has a single continuous state dimension in $[0, 1]$, with stochastic displacement when taking actions left or right.[3] Deepsea is similar to River Swim, but is a two-dimensional grid world. Reaching the high-reward state requires the agent to take the action to go right every time. There is a penalty of $\frac{0.01}{N}$ for taking the action right, except in the bottom right corner where there is a reward of 1 for taking the action right. A policy that explores uniform randomly has the probability of 2^{-N} of reaching the goal state in each episode.

We use slightly different evaluation metrics for the various environments. River Swim is continuing, so we report accumulated reward over learning. For both Deepsea and Puddle World, we report the undiscounted episodic return. For Mountain Car, we report the discounted return, because for every successful episode, the undiscounted return is 1 and so not meaningful in this sparse variant. For all episodic environments, we report steps on the x-axis and the corresponding episodic return on y-axis. All results in the classic environments use 50000 steps and 30 runs, except Deepsea which uses 10000 episodes and 5 runs.

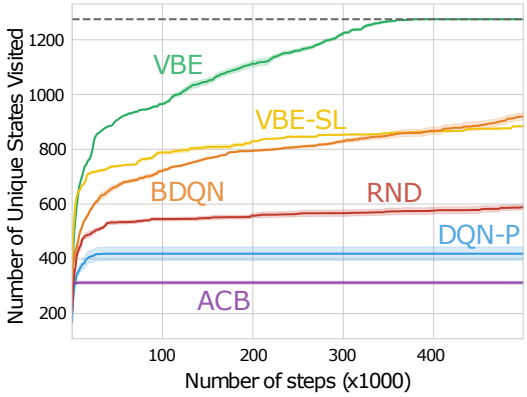
Across problems we compare VBE with DDQN-based variants of ACB and RND, DQN with additive priors (DQN-P) and BDQN. ACB, RND and VBE only differ in their value bonuses; we use the reward bonuses underlying ACB and RND to learn their respective value bonuses. As originally proposed, we make the reward-bonus value function non-episodic for ACB and RND. We also compared VBE with the released variants of ACB and RND that use PPO in Appendix B; in general, we find this PPO version to be less sample efficient than the DDQN versions. DQN-P simply adds an additive prior to DQN, like BDQN; it can be seen as BDQN with one value function in the ensemble. We evaluate the algorithms using 1, 2, 8 and 20 value functions in the ensembles and bonus scales of 1, 3 and 10. To match their original implementation RND uses two deep neural networks with multiple (64) nodes in the final layer as the target and predictor network for the reward bonus. All methods use the same neural network architectures, detailed in Appendix A.2.

We also include VBE-SL, that uses supervised learning instead of a TD update for the RQFs, to ablate this component of VBE. We discussed in Section 3 that the errors for VBE-SL likely reduce too quickly, resulting in insufficient exploration; we test that hypothesis here. Note that both VBE and VBE-SL only update their ensemble, with errors defining their value bonus, whereas ACB and RND both have to update their ensembles to get reward bonuses and learn a second value function to get the value bonus.

[3] One seemingly innocuous but important point to highlight is that we flipped the observation such that the high reward is at observation 0 and the lower reward is at observation 1. We did so because the standard random initialization and ReLU activation often results in a higher value for a higher input, thus favouring the correct action in the standard variant and making algorithms like BDQN look artificially good. Our modified variant removes this inadvertent bias without changing the problem structure or difficulty in any way.



(a) State coverage in Deepsea of different grid sizes



(b) Progression of unique states visited (grid size 50)

Figure 1: Contrasting the state coverage abilities of exploration algorithms in DeepSea. In (a) each bar corresponds to the total number of unique states visited by an agent after completing 10,000 episodes. The black stars indicate the total number of unique states for each grid size. Notably, VBE covers the entire state space, even for the larger grid sizes. (b) displays the progression of unique states visited by agents over the course of learning for Deepsea with grid size 50. The dotted line represents the total number of unique states (1275) in this environment. It provides evidence that VBE consistently explores new states at a significantly higher rate.

5.2 Pure Exploration

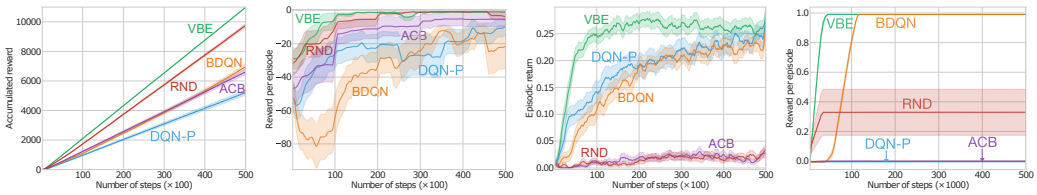
We first test how effectively the agents cover the state space in Tabular Deepsea with increasing grid sizes. In this setting, the agents observe no reward from the environment; thus, there are no harder to reach states, as in the original DeepSea environment. This allows us to evaluate agent's ability to do both directed deep exploration and employ first-visit optimism. For this tabular setting, the agents are otherwise the same as the other experiments, except the function approximation is linear in a one-hot encoding.

Figure 1a shows that VBE covers the entire state space for all the grid sizes. BDQN is able to cover the state space for a grid size of 30 and 35, but fails on bigger grids. Both ACB and RND fail to cover the state space, with ACB covering even less than DQN-P. This outcome is not surprising, given that neither approach ensures first-visit optimism. VBE-SL visits more unique states compared to ACB, RND and DQN-P. This is because VBE-SL provides first-visit optimism, encouraging the agent to take an action in a state if it has not done so before. But, as expected, it does not explore as much as VBE, likely as its value bonuses decay too quickly.

These suboptimal behaviors are emphasized in Figure 1b for a grid size of 50. All methods initially start exploring a similar number of states, easily reaching around 300 unique states. ACB, RND and DQN-P largely stop visiting new states very early in learning, though RND is slowly increasing the number of states it visits. BDQN and VBE-SL are similar across in their behavior, with VBE-SL

exploring more early, possibly due to better first-visit optimism. Over time, however, BDQN starts to catch up and then surpasses VBE-SL. VBE is the only algorithm that maintains a consistent increase until it has seen all states. It is interesting to note that VBE is able to cover the state-space with just 1 RQF in the ensemble, whereas the rest of the algorithms require much bigger ensembles and still fail to cover the state-space (Table 1). This highlights the ability of VBE to provide first-visit optimism and do deep exploration.

5.3 Comparison in Classic Environments



(a) River Swim (b) Puddle World (c) Mountain Car (d) Deepsea

Figure 2: Online performance in River Swim, Puddle World, Mountain Car, and Deepsea. Higher on the y-axis is better. The x-axis denotes the number of interaction steps with the environment. The shaded region corresponds to standard errors.

In this section we compare VBE with DQN-P, BDQN, ACB, and RND on the four classic control environments. In Figure 2, we see that VBE learns faster and reaches the best final performance in all four environments. Surprisingly, DQN-P is competitive with BDQN in three out of the four environments. In Deepsea, where persistent optimism is essential to reach the state with high reward, DQN-P fails. ACB and RND both fail to learn in the sparse reward domain Mountain Car, whereas, in Puddle World which has a denser reward structure, they perform better. RND is competitive in River Swim and Puddle World, however, it fails to learn the optimal policy in the majority of runs in Deepsea. ACB is competitive in Puddle World but fails in Deepsea. We also compare VBE with PPO-based variants of ACB and RND in the classic control environments in Appendix B.2

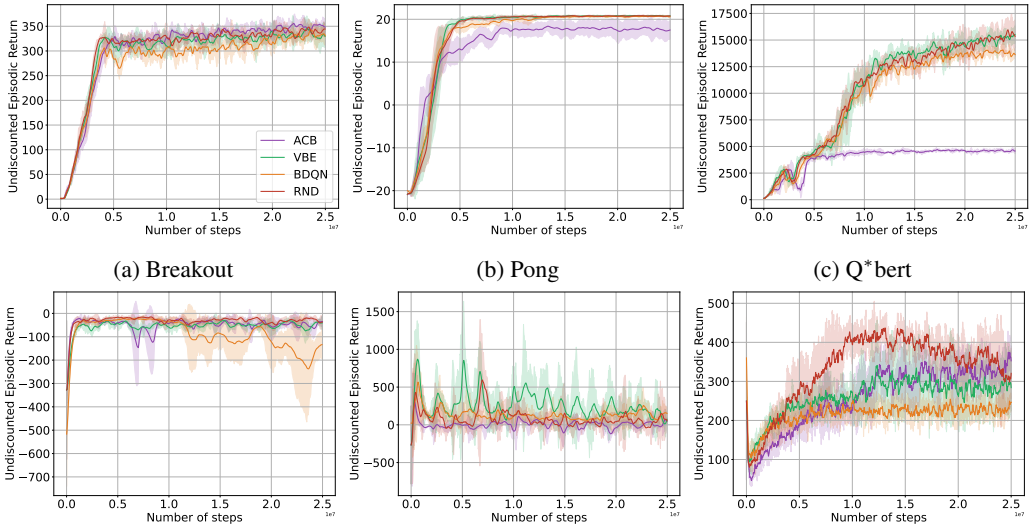
5.4 Atari

In this section we test VBE on several hard exploration Atari games, namely Private-Eye, Pitfall, Gravitar (Burda et al., 2019), and also on Breakout, Pong and Qbert. We chose this set to ensure a good mix of both hard and easy exploration environments Taiga et al. (2021). As is standard, we combine four consecutive frames to make the observation ($4 \times 84 \times 84$), and update agents ever four steps. We clip the rewards between $[-1, 1]$, and do 3 runs for all agents for 25 million steps.

For BDQN, we follow the choices from (Osband et al., 2016b), and use an ensemble size of 10, without bootstrapping.[4] As in the original BDQN implementation, each value function in the ensemble uses a shared representation network. We use a DDQN update for BDQN to be fair and consistent with the rest of the agents. For VBE, we also use an ensemble size of 10 and use a shared representation for the RQFs. To further improve the computational complexity we only update the RQF heads. ACB uses an ensemble size of $k = 128$ for computing the reward bonus, and RND uses a CNN-based target and predictor with 512 nodes in the final layer. We use $c = 10$ for all the agents in all the environments. Additional implementation details for Atari are mentioned in Appendix A.3

In Figure 3 we see that VBE performs consistently better than BDQN in all Atari environments. In Breakout and Pong, it takes longer for BDQN to achieve the same level of performance as the other agents. This could be attributed to random sampling of the value function for the behavior policy, thus requiring more time for all value functions to adjust. In Qbert and Gravitar, BDQN

[4]The original BDQN implementation for Atari does not use bootstrapping, that is, all members of the ensemble see the same data.



(a) Breakout (b) Pong (c) Q*bert

(d) Pitfall (e) Private-Eye (f) Gravitar

Figure 3: Online performance in six Atari games, with shaded regions corresponding to standard errors. The x-axis is the number of environment interaction steps in millions, and the y-axis is the online Undiscounted Episodic Return, where higher is better. The environments in the second row are considered to be more challenging in terms of exploration.

plateaus before the other agents, suggesting less exploration in these environments compared to the other agents. In Pitfall, BDQN does well in the beginning, but then fails to maintain a good policy in the later stages of training. ACB shows early learning in Pong, but then fails to converge to an optimal policy. Similarly in Qbert, ACB converges to a suboptimal policy, suggesting that ACB's value bonus is not suitable in these environments. In Breakout and Pitfall, ACB performs similarly to VBE, with ACB being slightly better in Breakout towards the end. In Gravitar, ACB is slow in the early stages of the training but then surpasses VBE in the later stages. RND performs very similar to VBE in Breakout, Pong, and Qbert, and is slightly better in Pitfall. However, in Gravitar, RND seems to do much better than all the other agents initially, but then fails to maintain that performance and drops down matching VBE. In Private-Eye none of the agents do well; however, VBE seems to be collecting a higher reward more often than the other agents.

The exact reason for an agent's performance in complex environments like Atari is hard to discern, and is something that requires additional understanding. However, after showcasing the efficacy of VBE in more controlled settings, the purpose of this experiment is to demonstrate that VBE can easily be extended to more complex environments and can even surpass or be at par with other exploratory baselines. In Appendix C we demonstrate that variants of VBE can further improve on these results.

6 Conclusion

In this work we introduced a new approach to do directed exploration in deep RL, called Value Bonuses with Ensemble errors (VBE). The utility of value bonuses is that it is simple to layer on top of an existing algorithm: the value bonuses are separately estimated and only impact the behavior policy. Improving how we estimate value bonuses, therefore, provides a promising path to replacing simple, but undirected exploration strategies like ϵ-greedy. To date, the primary way to estimate value bonuses has been to estimate a separate value function on reward bonuses, as was done for ACB and RND. This approach, however, does not encourage first-visit optimism; it only encourages revisiting an action once a reward bonuses was observed. We show that, in general, ACB and RND do not provide effective exploration, in classic environments and several Atari environments, and that VBE consistently outperforms BDQN.

Appendix: A Discussion on Convergence Criteria for Value Bonuses

First let us discuss how the theory for LSTD applies to our setting. The result from (Tagorti & Scherrer, 2015, Corollary 1) bounds the error of the value function learned under LSTD to the true value function, assuming features are linearly independent (Assumption 1) and a mixing assumption for the environment and behavior policy (Assumption 2). This bound includes an error to the best linear solution, for infinite data, and the error between the best linear solution and the true value function. Because we are in the realizable case and the objective is convex for linear function approximation, the best linear solution is the true value function and in the limit of data the LSTD solution will reach this best linear solution. We can write this as a corollary of their result. Note their result is written by value functions, but automatically extends to action-value function by considering state-action features and stationary distribution $\mu_b(s,a) = \mu(s)\pi_b(a|s)$.

Corollary 1 (Corollary following from [Theorem 1) *(Tagorti & Scherrer, 2015)] Assume we are given behavior policy π_b with stationary distribution μ and target policy π and the rewards are defined using a randomly sampled f_i from the set of linear functions on features $\phi(s,a)$ and the formula in Equation (3). Under Assumption 1 and 2 from (Tagorti & Scherrer, 2015), for a large enough number of samples T given by (Tagorti & Scherrer, 2015, Eq 6) (called n in their result), then f_{w_i} returned by LSTD satisfies*

$$\mathbb{E}_{s\sim\mu a\sim\pi_b(\cdot|s)}[(f_{w_i}(s,a) - f_i(s,a))^2] \leq O(1/\sqrt{T})$$

Now let us discuss how the work on neural TD applies to our setting (Cai et al., 2019). The result is proved for neural networks with a single hidden layer using a ReLU activation for the hidden layer, with the additional condition that the stationary distribution for the policy has a bounded density over states and the stepsizes decrease at a rate of $1/\sqrt{t}$. This result immediately implies that our f_{w_i} should converge to f_i, because the global solution for this problem is f_i because it is in the value function class. We state this as a corollary of their result here, to be clear about how it applies.

Corollary 2 (Corollary following from [Theorem 4.6) *(Cai et al., 2019)] Assume that 1) the policy π is fixed with stationary distribution μ, where $\mu(s)\pi(a|s)$ has bounded density across the space $x = (s,a)$ 2) the function class $\mathcal{F} = \{\frac{1}{\sqrt{m}}\sum_{j=1}^{m} b_j \max(x^\top w_j, 0)|W = (b_1,\ldots,b_m,w_1,\ldots,w_m), ||W - W(0)||_2 \leq B\}$ for $x = (s,a)$, $W(0)$ a point at which the weights are initialized in the algorithm and B some constant, 3) $\|x\|_2 = 1$ for all x and the rewards are defined using a randomly sampled f_i from $\mathcal{F}$ and the formula in Equation (3), and 4) the Neural TD algorithm (Algorithm 1 in (Cai et al., 2019)) is run for T steps with stepsize $\eta = \min((1-\gamma)/8, 1/\sqrt{T})$. Then the algorithm returns f_{w_i} that satisfies*

$$\mathbb{E}_{W\sim,\mu\pi}[(f_{w_i}(s,a) - f_i(s,a))^2] \leq \frac{O(B^2)}{(1-\gamma)^2\sqrt{T}} + O(B^2 m^{-1/2} + B^{5/2} m^{-1/4})$$

Proof: The result also requires that the reward magnitudes are all bounded, which they are by construction. Theorem 4.6 states that the outputted action-value function is bounded as above to the global optimum in the function class. Because $f_i(s,a)$ is in the function class, we know it is the global optimum. ■

References

Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári. Improved algorithms for linear stochastic bandits. *Advances in neural information processing systems*, 24, 2011.

Yasin Abbasi-Yadkori, Nevena Lazic, Csaba Szepesvari, and Gellert Weisz. Exploration-enhanced politex. *arXiv preprint arXiv:1908.10479*, 2019.

Jordan T. Ash, Cyril Zhang, Surbhi Goel, Akshay Krishnamurthy, and Sham M. Kakade. Anti-concentrated confidence bonuses for scalable exploration. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=RXQ-FPbQYVn.

Marc G Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems*, 2016.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1lJJnR5Ym.

Qi Cai, Zhuoran Yang, Jason D Lee, and Zhaoran Wang. Neural Temporal-Difference Learning Converges to Global Optima. In *Advances in Neural Information Processing Systems*, 2019.

Leshem Choshen, Lior Fox, and Yonatan Loewenstein. DORA the explorer: Directed outreaching reinforcement action-selection. *CoRR*, 2018.

Kamil Ciosek, Vincent Fortuin, Ryota Tomioka, Katja Hofmann, and Richard Turner. Conservative uncertainty estimation by fitting prior networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJlahxHYDS.

Bo Dai, Niao He, Yunpeng Pan, Byron Boots, and Le Song. Learning from conditional distributions via dual embeddings. In *Artificial Intelligence and Statistics*. PMLR, 2017.

R Grande, T Walsh, and J How. Sample efficient reinforcement learning with gaussian processes. In *International Conference on Machine Learning*, 2014.

David Janz, Jiri Hron, Przemysł aw Mazur, Katja Hofmann, José Miguel Hernández-Lobato, and Sebastian Tschiatschek. Successor uncertainties: Exploration and uncertainty in temporal difference learning. In *Advances in Neural Information Processing Systems*, 2019.

Raksha Kumaraswamy, Matthew Schlegel, Adam White, and Martha White. Context-dependent upper-confidence bounds for directed exploration. *arXiv preprint arXiv:1811.06629*, 2018.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *World Wide Web Conference*, 2010.

Brendan O'Donoghue, Ian Osband, Remi Munos, and Vlad Mnih. The Uncertainty Bellman Equation and Exploration. In *International Conference on Machine Learning*, 2018.

I Osband, B Van Roy, and Z Wen. Generalization and exploration via randomized value functions. In *International Conference on Machine Learning*, 2016a.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, 2016b.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized Prior Functions for Deep Reinforcement Learning. *NeurIPS*, 2018.

Ian Osband, Benjamin Van Roy, Daniel J Russo, Zheng Wen, et al. Deep exploration via randomized value functions. *J. Mach. Learn. Res.*, 20(124):1–62, 2019.

Ian Osband, Zheng Wen, Seyed Mohammad Asghari, Vikranth Dwaracherla, Morteza Ibrahimi, Xiuyuan Lu, and Benjamin Van Roy. Approximate Thompson Sampling via Epistemic Neural Networks. In *Uncertainty in Artificial Intelligence*, 2023.

Georg Ostrovski, Marc G Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *International Conference on Machine Learning*, 2017.

Andrew Patterson, Adam White, and Martha White. A generalized projected bellman error for off-policy value estimation in reinforcement learning. *The Journal of Machine Learning Research*, 2022.

Manel Tagorti and Bruno Scherrer. On the Rate of Convergence and Error Bounds for LSTD(λ). In *Proceedings of the 32nd International Conference on Machine Learning*. PMLR, 2015.

Adrien Ali Taiga, William Fedus, Marlos C Machado, Aaron Courville, and Marc G Bellemare. On bonus-based exploration methods in the arcade learning environment. *arXiv preprint arXiv:2109.11052*, 2021.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, 2016.

Yining Wang, Ruosong Wang, Simon S Du, and Akshay Krishnamurthy. Optimism in reinforcement learning with generalized linear function approximation. *arXiv preprint arXiv:1912.04136*, 2019.

Zhikang T Wang and Masahito Ueda. Convergent and efficient deep q network algorithm. In *International Conference on Learning Representations*, 2022.

Martha White. Unifying task specification in reinforcement learning. In *International Conference on Machine Learning*, 2017.

Lin Zhao, Huaqing Xiong, and Yingbin Liang. Faster non-asymptotic convergence for double q-learning. In *Advances in Neural Information Processing Systems*, 2021.

Supplementary Materials

The following content was not necessarily subject to peer review.

A Experiment Details

A.1 Environment Details

Mountain Car is classic control problem of driving an underpowered car up a mountain. The original problem is set up as cost-to-goal, and here to frame it as a challenging exploration problem we offset the reward by 1, making it a sparse reward problem. The start state is sampled from the range $[-0.6, -0.4]$, which is the valley between two mountains, and the car starts with velocity zero.

Puddle World is a continuous state 2-dimensional world with $(x, y) \in [0, 1]^2$ with 2 intersecting puddles: (1) $[0.45, 0.4]$ to $[0.45, 0.8]$, and (2) $[0.1, 0.75]$ to $[0.45, 0.75]$. The puddles have a radius of 0.1 and the goal is the region $(x, y) \in [0.95, 1.0], [0.95, 1.0]$. The problem is cost-to-goal with additional penalty for when the agent is either puddle. The penalty for being in a puddle is inversely proportional to the distance of the agent from the center of the puddle, i.e., higher negative reward for being closer to the center. The agent chooses a direction of movement, resulting in displacement equal to $0.005 + \zeta, \zeta \sim N(\mu = 0, \sigma = 0.1)$ in the chosen direction. The starting positions for episodes is uniformly sampled from $(x, y) \in [0.1, 0.3], [0.45, 0.65]$. High variance transitions coupled with high magnitude penalties make this a challenging exploration problem.

River Swim is a standard continuing exploration benchmark inspired by a fish trying to swim up-river, with high reward (+1) upstream which is difficult to reach and, a lower but still positive reward (+0.005), which is easily reachable downstream. The state space is continuous in $[0, 1]$, and the stochastic displacement is equal to $0.1 + \zeta, \zeta \sim N(\mu = 0, \sigma = 0.01)$ in the direction of the chosen action up or down. As swimming upstream is difficult, action up is stochastically switched to down. We also flip the observation such that the high reward is at observation 0 and the lower reward is at observation 1. We do this because we noticed that using random initialization with ReLU activations would mostly result in a higher value for a higher input thus favouring the correct action in this case. The starting position is sampled uniformly in $[0.9, 1.0]$.

Deepsea is a finite-horizon episodic grid world environment, which poses a hard exploration challenge. In each state the agent can take two actions, left or right. Every action moves the agent down one row with column change being controlled by the chosen action. Collisions to the grid edges are handled by the agent staying in the same column but moving down one row. Given the transition structure, the agent can never access the states in the top-right triangle of the grid. Therefore, the total number of states are $\frac{N \times (N+1)}{2}$. The most rewarding state is the state on the bottom-right corner of the grid. To reach this state successfully in an episode, the agent needs to take the action that moves it towards right at every step. However, there is a penalty of $\frac{0.01}{N}$ for taking this action in every state, except for in the high rewarding state where the agent gets a reward of 1 for taking the right action. This transition and reward structure make it a very challenging environment. A policy that explores uniform randomly has a probability of 2^{-N} of reaching the highly rewarding state in any episode.

A.2 Algorithm Details for Classics Control

In the classic control experiments (Section 5.3), every agent uses the same neural architecture, containing 2 hidden non-linear layers with 50 nodes each and ReLU activation, followed by a linear output-layer. DQN-P, BDQN, VBE, ACB, and RND all use target networks which are updated periodically after every τ steps. For DQN-P and BDQN $\tau = 4$ works best for all four classic environments, as used by the BDQN paper. VBE, ACB and RND use $\tau = 4$ for Mountain Car, Puddle World and River Swim, and $\tau = 64$ for Deepsea. We use a learning rate of $\alpha = 0.001$ and a discount factor of $\gamma = 0.99$. DQN-P, BDQN and VBE variants use an experience replay buffer that stores the

most recent 50K transitions. The agent's parameters are updated after every step using a randomly sampled mini-batch of 128. We sweep the agents on bonus scales $c = [1.0, 3.0, 10.0]$, and ensemble sizes $k = [1, 2, 8, 20]$. The PPO version of ACB uses an ensemble size of $k = 128$, and RND uses a multi-layer neural network instead of an ensemble. Tables 1, and 2 show the best performing sets of ensemble size k and bonus scale c for results in Sections 5.2, and 5.3, correspondingly.

	Deepsea
VBE	$k = 1, c = 1.0$
VBE-SL	$k = 20, c = 1.0$
DQN-P	$k = 1, c = 1.0$
BDQN	$k = 20, c = 1.0$
ACB	$k = 20, c = 1.0$
RND	$c = 1.0$

Table 1: Ensemble size k and bonus scale c for agents in Figure 1a.

	River Swim	Puddle World	Mountain Car	Deepsea
VBE	$k = 20, c = 1.0$	$k = 1, c = 10.0$	$k = 2, c = 1.0$	$k = 20, c = 1.0$
DQN-P	$k = 1, c = 10.0$	$k = 1, c = 3.0$	$k = 1, c = 1.0$	$k = 1, c = 10.0$
BDQN	$k = 8, c = 10.0$	$k = 2, c = 1.0$	$k = 20, c = 1.0$	$k = 20, c = 10.0$
ACB	$k = 20, c = 10.0$	$k = 1, c = 1.0$	$k = 8, c = 1.0$	$k = 20, c = 1.0$
RND	$c = 10.0$	$c = 1.0$	$c = 1.0$	$c = 1.0$

Table 2: Ensemble size k and bonus scale c for agents in Figure 2.

A.3 Algorithm Details for Atari

All the agents used in Atari experiments use the same architecture: A representation network with 3 CNN layers followed by a value function head containing one hidden layer with Relu activation and a final linear layer. We use a DDQN update for all agents and update every 4 steps. We update the target networks every 10000 steps. We use a replay buffer that stores 1 million of the most recent transitions and discount factor of 0.99. Finally, we use Adam optimizer with a learning rate of 0.0000625.

B Comparing VBE with PPO based variants of ACB and RND

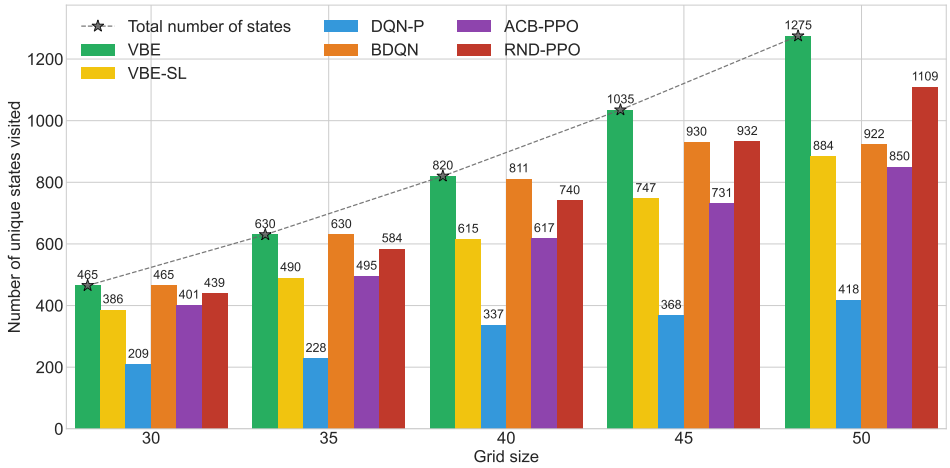
In this section we compare VBE with PPO-based variants of ACB and RND as they were originally implemented and evaluated – we call these algorithm ACB-PPO and RND-PPO, respectively. We use the opensource implementation of PPO-based ACB and RND as provided by Ash et al. (2022)[5]. To ensure that each algorithm uses the same amount of data, we use a single agent interacting with the environment for ACB-PPO and RND-PPO, instead of parallel agents interacting with parallel instances of the environment. Using a softmax policy and parallel agents interacting with the environments can also help with exploration, so to be fair we use only one instance of agent-environment interaction, as is done for VBE.

B.1 Pure Exploration

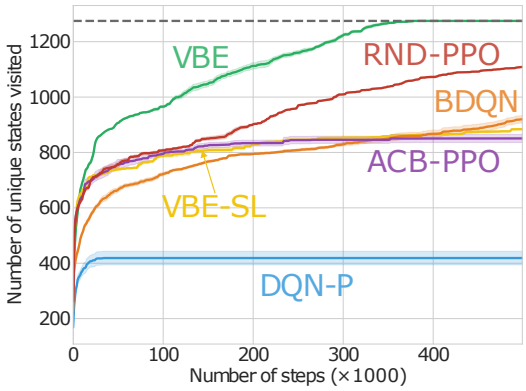
In this section we first compare VBE with PPO-based variants of ACB and RND in the pure exploration setting. Similar to in Section 5.2, we use one-hot encoding and linear function approximation for all the agents. The agents observe no environment reward and are behaving only with respect

[5] See https://github.com/JordanAsh/acb/tree/main

to the exploration strategy employed by each of these algorithms. In Figure 4 we see that the PPO-based variants of ACB and RND cover much more of the state-space compared to their Value-based (VB) counterparts in Figure 1a. This may be due to the fact that PPO-based variants of ACB and RND use a softmax behaviour policy, which can cause the agent to take the non-greedy actions at random. In Figure 4b we see that ACB-PPO almost flatlines and stops visiting new states after sometime. However, RND-PPO keeps on visiting new states. Although this is a significant improvement, however, both ACB-PPO and RND-PPO still fail to cover the state-space.



(a) State coverage in Deepsea of different grid sizes

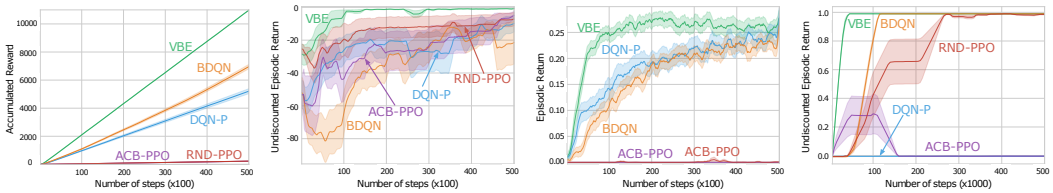


(b) Progression of unique states visited (grid size 50)

Figure 4: Contrasting the state coverage abilities of exploration algorithms in DeepSea. In (a) each bar corresponds to the total number of unique states visited by an agent after completing 10,000 episodes. The black stars indicate the total number of unique states for each grid size. Notably, VBE covers the entire state space, even for the larger grid sizes. (b) displays the progression of unique states visited by agents over the course of learning for Deepsea with grid size 50. The dotted line represents the total number of unique states (1275) in this environment. It provides evidence that VBE consistently explores new states at a significantly higher rate.

B.2 Classic Control

The chosen hyperparmaters of ACB-PPO and RND-PPO as compared to VBE for classic control domains are shown in Table 3. Figure 5 shows the result comparing these algorithms. In general, across the domains the PPO-variants' perform similar to, or poorer than, their VB variants in Figure 2. VBE, and BDQN, outperform them across domains. Even DQN-P is competitive with them, or better in all domains except Deepsea. Compared to their VB variants, the performance of the PPO variants drops in River Swim, and Puddle World, and continues to be the same in Mountain Car. In Deepsea RND-PPO improves upon its VB variant RND, while ACB-PPO continues to perform similarly to ACB.

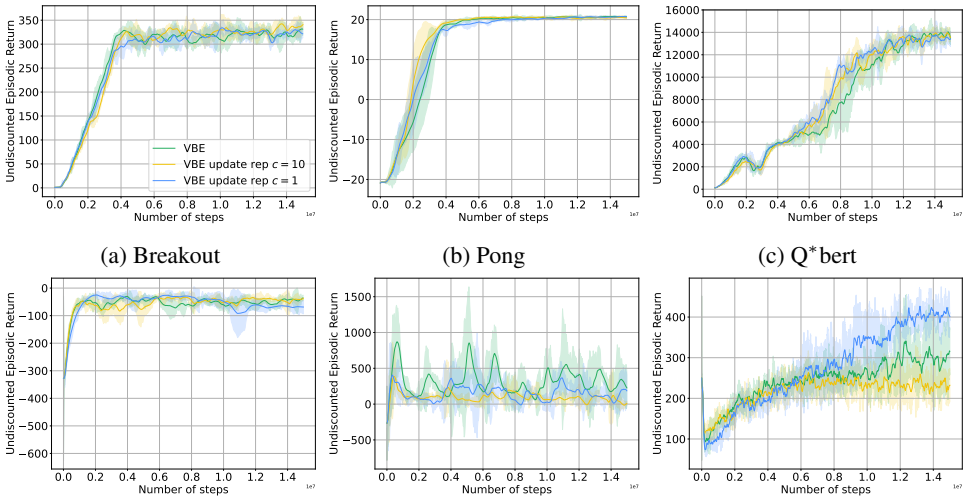


(a) River Swim (b) Puddle World (c) Mountain Car (d) Deepsea

Figure 5: Online performance of PPO-based variants of ACB and RND in River Swim, Puddle World, Mountain Car, and Deepsea. Higher on the y-axis is better. The x-axis denotes the number of interaction steps with the environment. The shaded region corresponds to standard errors.

	River Swim	Puddle World	Mountain Car	Deepsea
VBE	$k = 20, c = 1.0$	$k = 1, c = 10.0$	$k = 2, c = 1.0$	$k = 20, c = 1.0$
ACB ($k = 128$)	$c = 1.0$	$c = 3.0$	$c = 1.0$	$c = 10.0$
RND	$c = 1.0$	$c = 10.0$	$c = 10.0$	$c = 1.0$

Table 3: Ensemble size k and bonus scale c for agents in Figure 5.



(a) Breakout (b) Pong (c) Q*bert

(d) Pitfall (e) Private-Eye (f) Gravitar

Figure 6: Comparing the online performance of VBE and two of its variants on six Atari environments. We do 3 runs for each agent for 15 Million steps.

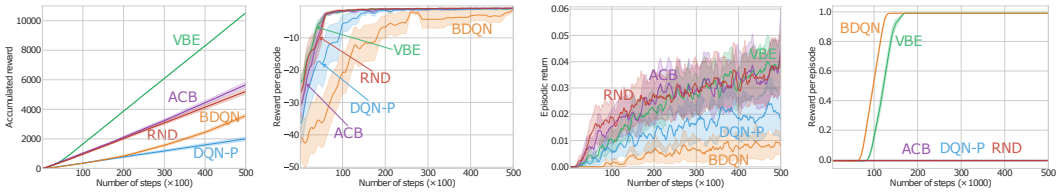
C An alternate variant of VBE for Atari

The VBE in Section 5.4 uses a shared representation for RQFs and only updates the RQF head to reduce computational complexity. In this section, we test another variant of VBE that has a separate representation network for each RQF and updates the entire network. We also test this new variant with a smaller bonus-scale $c = 1$. The purpose of this experiment was to test whether reducing the representational ability of VBE by using a shared representation and not updating it can hinder its learning capabilities. However, we observe that the base variant of VBE can keep up with its more parameterized counterpart. In Breakout and Pong, the variant with learnable representation networks shows slight improvements compared to the base VBE. We also note that using a smaller c can affect the performance in hard exploration environments. In Pitfall, VBE with $c = 1$ performs well initially but then takes a dip after 10 million steps. Consistent with all other agents, VBE's variants are unable to learn on Private Eye, however, base VBE and the variant with $c = 1$ collects higher rewards more often. In Gravitar, VBE with $c = 1$ does exceptionally well compared to the

other variants and matches the performance of RND in Figure 3f. This experiment reveals that our design choices for using a shared representation for RQFs and only updating the RQF heads is valid and allows the agent to explore. It further reveals that more investigation is needed to understand the role of bonus-scale in these environments. With more compute resources, and search over the hyperparameters can potentially identify a variant of VBE that can perform even better on these environments.

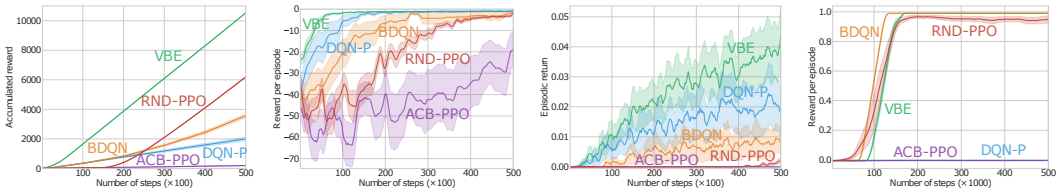
D Linear Function Approximation

In this section we test VBE and the baseline agents on the same four classic environments as in Section 5.3, with tile-coded features and linear function approximation. We use the following tile-coding parameters – River Swim :($tiles = 4, tiling = 32, features = 128$), Puddle World: ($tiles = 5, tiling = 5, features = 128$), and Mountain Car: ($tiles = 4, tiling = 16, features = 512$). The results in Figure 7 are similar to their neural network counterpart results in Figure 2, in that VBE does best or is competitive across all domains. RND and ACB perform similarly in all domains – being competitive with VBE in Puddle World and Mountain Car, but failing in River Swim and Deepsea. BDQN outperforms VBE in Deepsea marginally. DQN-P is competitive with BDQN in all domains except Deepsea – outperforming it in in Puddle World and Mountain Car. We also compare VBE, BDQN, and DQN-P to PPO-based variants of ACB and RND in Figure 8. ACB-PPO's performance drops with respect to ACB in all domains. RND-PPO on the other hand improves in Deepsea to be competitive with BDQN and VBE, has a high rate of improvement later towards later learning in River Swim, while dropping in perfomance in both Puddle World and Mountain Car with respect to its value-based variant.



(a) River Swim (b) Puddle World (c) Mountain Car (d) Deepsea

Figure 7: Online performance in River Swim, Puddle World, Mountain Car, and Deepsea, with tile-coded features and linear function approximation. Higher on the y-axis is better. The x-axis denotes the number of interaction steps with the environment. The shaded region corresponds to standard errors.

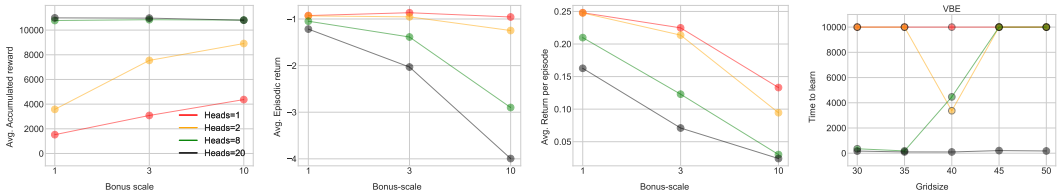


(a) River Swim (b) Puddle World (c) Mountain Car (d) Deepsea

Figure 8: Online performance of PPO-based variants of ACB and RND in River Swim, Puddle World, Mountain Car, and Deepsea, with tile-coded features and linear function approximation. Higher on the y-axis is better. The x-axis denotes the number of interaction steps with the environment. The shaded region corresponds to standard errors.

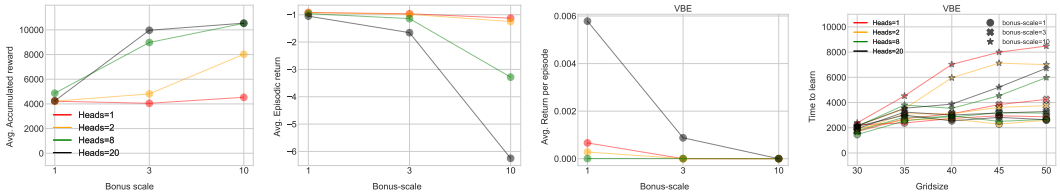
E Parameter Sensitivity of VBE: Ensemble Size × Bonus Scale

In this section we show the effect that bonus scales and ensemble sizes have on the performance of the VBE in each of the four classic control environments. In Figure 9 we show the sensitivity of VBE used in Section 5.3 to its two parameters – ensemble size and bonus scale — in each environment. For River Swim we see that the performance improves as the bonus scale and the ensemble size is increased. This makes sense as River Swim is a hard exploration environment and requires more aggressive exploration. In Puddle World and Mountain Car, we observe that increasing the bonus scale and the ensemble size harms the performance, since they do not require too much exploration. For Deepsea we only test a bonus scale of 1 with different ensemble sizes on different grid sizes. We can see that only an ensemble size of 20 works well on all grid sizes. Figure 10 shows the parameter sensitivity results for the linear function approximation case. We observe a very similar pattern to its neural network counterpart across all the environments.



(a) River Swim (b) Puddle World (c) Mountain Car (d) Deepsea

Figure 9: Shows the effect of different bonus scales and ensemble sizes across the classic control environments. For Deepsea, we only use a bonus scale of 1 and test different ensemble sizes.



(a) River Swim (b) Puddle World (c) Mountain Car (d) Deepsea

Figure 10: Shows the effect of different bonus scales and ensemble sizes across the classic control environments. These results correspond to the linear function approximation case.

F Target Policy Experiments

In this framework where we have access to two forms of value functions – the action-values, and their corresponding bonuses – there are two fundamental learning updates that we can employ – on-policy, or off-policy. As we are in the control setting, we choose the off-policy q-learning update that tries to estimate q^* directly in VBE. In this section we explore the impact of this choice by comparing the two updates: (1) on-policy updates where the target policy is the same as the behaviour policy, and (2) off-policy updates, which is the one employed by VBE, where the target policy is different from the behaviour policy. The former uses an optimistic target policy that maximizes over $q(s,.)+b(s,.)$, whereas the latter uses a greedy target policy that maximizes over $q(s,.)$ alone.

In Figure 11 we show the performance of the agents resulting from the two different updates, run on different grid sizes of Deepsea, with different ensemble size represented by the different colors, and different bonus scales represented by different shapes. The agents use linear function approximation for learning the action-value function and RQFs. We can see that the greedy agent generally performs better than the optimistic agent, which makes sense as the optimistic target policy can cause

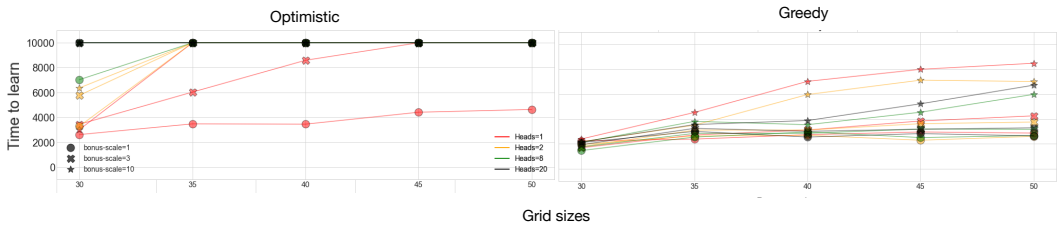
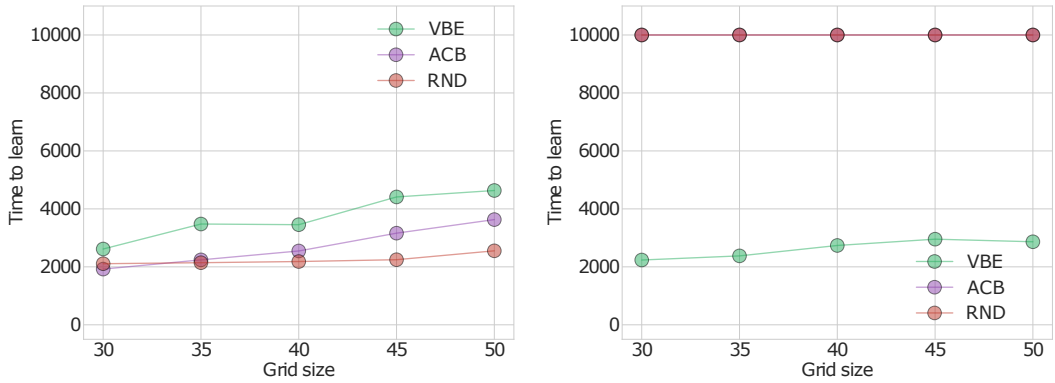


Figure 11: Comparing Optimistic target policy (On-policy) with Greedy target policy (Off-policy) on different grid sizes of Deepsea.



(a) Optimistic target policy

(b) Greedy target policy

Figure 12: This figure compare optimistic target policy to Greedy target policy: (a) the agents use an on-policy (optimistic) updates, and (b) the agents use an off-policy (greedy) updates. ACB and RND fail with the greedy policy, whereas with optimistic target policy they outperform VBE. VBE however, does well with a greedy policy compared to the optimistic one. These agents use a single linear-layer with bias term.

more exploration. The greedy agent performs well even with multiple RQFs, whereas optimistic agent fails to learn the optimal policy in this fixed budget of steps when the number of RQFs are increased.

Additionally, we investigated into the role of the target policy in ACB and RND. During these experiments with ACB and RND we noticed an interesting phenomenon: using an optimistic target policy allows ACB and RND to learn the optimal policy quickly on various grid-sizes of the Deepsea environment (Figure 12), and using a greedy target policy for ACB and RND would cause the agents to fail to learn the optimal policy (Figure 7d, 2d). This is interesting as in either case we do not expect ACB and RND to be able to cover the entire state space based on random initialization because these reward bonus methods do not provide optimism for unseen state-action pairs – that is, they do not provide first-visit optimism. Upon investigating the phenomenon, we found out that this happens because of the bias term in the linear layer, the momentum term in the optimizer and because the intrinsic value function in these methods are designed to be non-episodic. When the optimistic target policy is used along with momentum during optimization, the bias term of the approximator consistently increases. Since the bias-term is a shared parameter, the increase in its value causes the intrinsic action values to start increasing, providing the optimism for unseen action-values as well – allowing for the agent to cover the state space and thus learn the optimal policy. In case of the greedy target policy this phenomenon does not arise; the intrinsic values do not increase, and thus the agent fails to cover the state space. In Section 5.2, we show that if we use tabular features with a linear-layer without any bias term then ACB and RND fail to cover the state space, reflecting that they do not provide first-visit optimism.