VALUE BONUSES USING ENSEMBLE ERRORS FOR EXPLORATION IN REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

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. This approach, however, 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 for designing 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 learns faster in several Atari environments.

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., 2016; 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 $\text{argmax}_a\, q(s,a) + b(s,a)$. 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 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 specified (White, 2017). On 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.

[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.

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 finally provide the VBE algorithm that uses this ensemble.

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.

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 $q_i^\pi = f_i$.

[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.

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 A of how this existing theory on convergence of TD applies to our setting.

4 Using the Ensemble of Value Functions

We provide the Value Bonuses with Ensemble Errors (VBE) algorithm in this section. We provide pseudocode, in Algorithm 1, for the case where the base algorithms 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 instead 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.

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: $f_i, \dots f_k$, predictor RQFs: $f_{w_1}, \dots, f_{w_k}$, and target networks: $q_{\tilde{w}}$, $f_{\tilde{w}_1}, \dots, f_{\tilde{w}_k}$
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 $t = 0, 1, \dots$ **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 the action-values using the Double DQN update
    Sample a mini-batch and use update in Equation (1) to update q_w
    // Update one randomly select ensemble value function
    Sample i from $[k]$ uniform randomly
    Sample a mini-batch from B and update f_{w_i} using Equation (5) where
    for each (s, a, r, s', γ) we 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 can again ask what happens to our value bonuses in VBE. 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 $q_\theta = f_\theta + p$ for a random prior function p that is not updated and 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 $r + \gamma(f_\theta(s',a') + p(s',a')) - (f_\theta(s,a) + p(s,a)) = r - (p(s,a) - \gamma p(s',a')) + \gamma f_\theta(s',a') - f_\theta(s,a)$. This is a standard update with reward bonus $p(s,a) - \gamma p(s',a')$, and this bonus is the 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 $-p(s,a)$ using the exact same argument to the one in our Proposition 1, just negating the function p. Consequently, $f_\theta(s,a) + p(s,a) = q^\pi(s,a) + b^\pi(s,a) + p(s,a) = q^\pi(s,a) + (b^\pi(s,a) + p(s,a))$ goes to q^π since $b^\pi(s,a) + p(s,a)$ eventually cancels.

This argument is not how randomized priors are presented, but provides another intuitive interpretation. Further, it highlights a key difference BDQN and VBE: BDQN takes a Thompson sampling approach to induce optimism whereas VBE acts greedily with respect to optimistic value estimates.

5 Experiments

We evaluate our proposed algorithm on four classic exploration environments and six Atari environments, particularly in comparison to BDQN and the reward bonuses 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 investigate the impact of

the bonus scale and number of RQFs in the ensemble. We also compare variants of VBE which use ACB and RND's reward bonus strategy to to estimate the value bonuses (VB ACB and VB RND). We conclude with experiments in Atari, particularly highlighting how to scale VBE to this setting.

5.1 ENVIRONMENTS

The four classic exploration environments are Sparse Mountain Car, Puddle World, River Swim and DeepSea. These four environments have varying requirements for exploration: DeepSea and River Swim are considered hard exploration environments, whereas Puddle World and Mountain Car require less exploration. The full details for these environments are in Appendix B, but we list a few key details here.

Mountain Car has two-dimensional continuous inputs, with a sparse reward structure: 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 and DeepSea were both designed as hard exploration problems, requiring persistent behavior with likely failure under dithering, ϵ-greedy exploration. 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. One seemingly innocuous but important point for this environment 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 this case. This other variant removes this inadvertent bias without changing the problem structure or difficulty in any way.

DeepSea is similar to River Swim, but is a grid world environment. Reaching the high-reward state requires the agent to take the action to go right every time. However, there is a penalty of $\frac{0.01}{N}$ for taking the action right, except for the right most state where the agent gets a reward of 1 for taking the right action. A policy that explores uniform randomly has the probability of 2^{-N} of reaching the goal state in each episode.

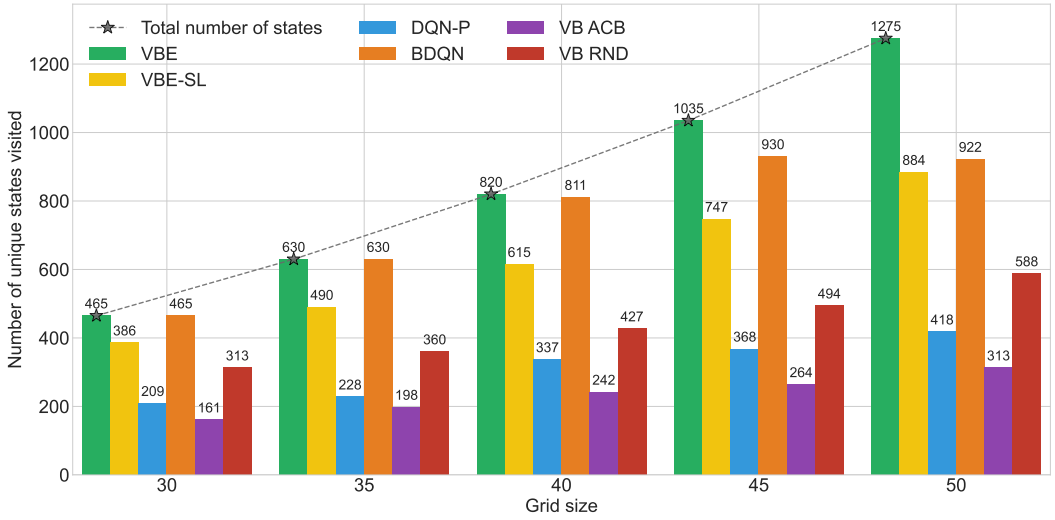
5.2 ALGORITHMS AND EXPERIMENTAL SETTINGS

The environments uses slightly different evaluation metrics. 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. The default grid size for Deepsea, unless mentioned, is 50, which is the largest grid size we experiment with.

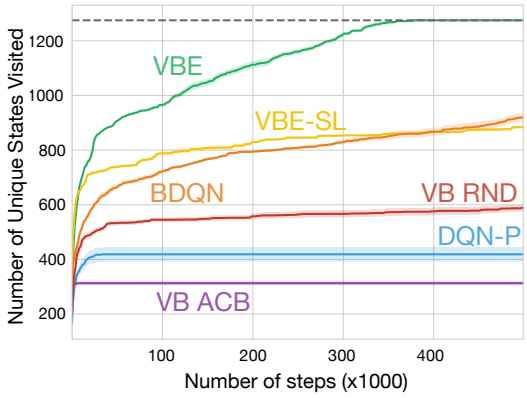
Across problems we compare VBE with DQN with additive priors (DQN-P), BDQN, the released variants of ACB and RND that use PPO [3], and their DDQN-based variants: VB ACB and VB RND. DQN-P simply adds an additive prior to DQN, like BDQN has; it can be seen as BDQN with one value function in the ensemble. For both VBE and BDQN, we test using 1, 2, 8 and 20 value functions in the ensembles and bonus scales of 1, 3 and 10. ACB and RND also use the same bonus scales. To match their original implementations ACB uses an ensemble of 128 to estimate the reward bonus, and RND uses two deep neural network 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 B.

We also include variants of VBE to provide evidence for the way we estimate the value bonuses with our ensemble errors. We include VBE-SL, meaning that instead of the TD update, we use a supervised learning update. We discussed in Section 3 that the errors for VBE-SL are likely to reduce too quickly, resulting in insufficient exploration; we test that hypothesis here. We also use the reward bonuses underlying ACB and RND to learn the value bonus and replace them with VBE's

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



(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 visited 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.

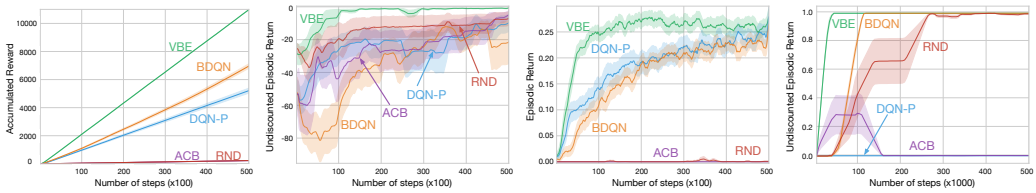
ensemble value bonus, i.e., VB ACB and VB RND. As proposed by the authors we make the reward-bonus value function non-episodic. VB ACB, VB RND and VBE-SL are otherwise exactly the same as VBE, including using DDQN, defining the value bonus using the ensemble error and using the value bonus in the same way.

5.3 Pure Exploration

We first test how effectively the agents cover the state space in Tabular DeepSea with increasing grid sizes. For this tabular setting, the agents are otherwise the same as the other experiments, except the function approximation is linear on a one-hot encoding.

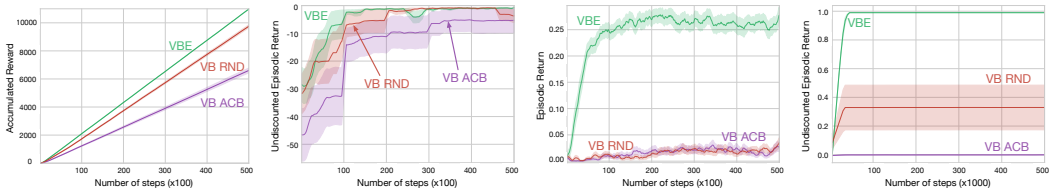
Figure 1a shows that VBE covers the entire state space for different grid sizes. BDQN is able to cover the state space for a grid size of 30 and 35, but starts to degrade after that. Both VB ACB and VB RND fail to cover the state space, with VB ACB covering even less than DQN-P. This outcome is not surprising, given that neither approach ensures first-visit optimism. VBE-SL at least includes 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. VB ACB, DQN-P and VB RND largely stop visiting new states very early in learning, though VB RND is slowly increasing the number of states it visits. BDQN and VBE-SL 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.



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

Figure 2: Comparison of online performance in River Swim, Puddle World, Mountain Car, and Deepsea. In all domains, higher on 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 3: Comparing online performance VBE to two alternative ways to estimate value bonuses, namely estimating a value function on the reward bonuses given by RND and ACB. The shaded regions corresponds to standard errors.

5.4 Comparison in Classic Environments

In this section we compare VBE with DQN-P, BDQN, ACB, and RND. ACB and RND here use PPO as originally proposed. In Figure 2, 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. ACB and RND fail to learn in both the sparse reward domains Riverswim and Mountain Car. In Puddle World and Deepsea which have a denser reward structure, they perform better. RND learns slowly in both, whereas ACB is competitive in Puddle World, but does poorly in Deepsea.

5.5 Alternative Choices for the Value Bonus

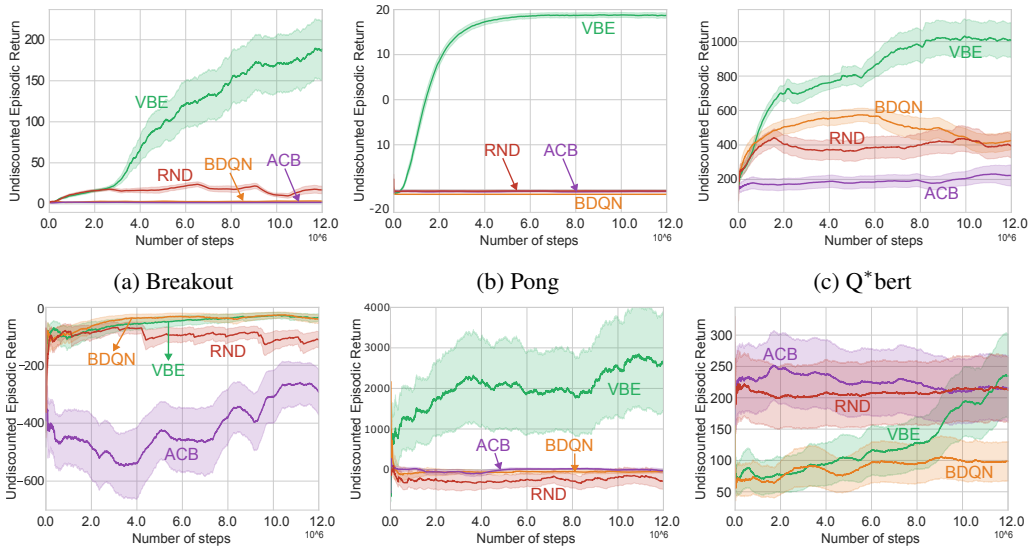
We compared VBE to VB ACB and VB RND for pure exploration; now we do so for the four classic environments. VB ACB and VB RND are another natural way to estimate value bonuses—albeit missing first-visit optimism—and help validate our new approach to estimating value bonuses.

In Figure 3 we see that VBE outperforms VB ACB and VB RND. Similar to their PPO versions in Figure 2, they fail in Mountain Car and are competitive in Puddle World. However, behavior is quite different in Deepsea and Riverswim. Now VB ACB almost matches VBE in Riverswim, and VB RND has significantly improved compared to its PPO version. But, in Deepsea, they both perform notably more poorly, especially VB RND fails here but succeeded with its PPO version. The primary difference between them is using DDQN as a base algorithm, rather than PPO.

5.6 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. As is standard, we combine four consecutive frames to make the observation ($4 \times 84 \times 84$), and update VBE ever four steps. We do 3 runs for all the agents for 12 million steps.

In VBE the target and the predictor RQFs have 3 CNN-layers, followed by 2 non-linear layers (representation network) and a final linear output-layer. We only update the final layer of the predictor RQFs, and initialize the predictors to have the same representation network as the target RQFs, so to have learnable targets. In practice we found that even 1 RQF works well for VBE, so that is what we use in our experiments. We chose this configuration because it was faster to run and seemed to



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

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

Figure 4: Comparing online performance in six Atari games, with shaded regions corresponding to standard errors. The environments in the second row are considered to be more challenging from an exploration perspective. The x-axis is the number of environment interaction steps in millions, and the y-axis is the online Undiscounted Episodic Return, for which higher is better.

be more stable than updating the whole network. We include the comparison to the variant of VBE where we update the whole network for the RQFs in Appendix D.

For BDQN we use an ensemble size of 10, and used a double-or-nothing bootstrap $p = 0.5$ (Owen & Eckles, 2012). As in the original BDQN implementation, each value function in the ensemble uses a shared representation network. The additive priors have the same network architecture as the value functions. ACB and RND agents do 128-step roll-outs and then do 4 epochs of network updates using PPO. To make sure that each agent gets the same number of interactions with the environment and to match our online setting, we run ACB and RND with one agent interacting with the environment instead of running multiple parallel agents. ACB uses an ensemble size of $k = 128$ for computing the reward bonus, and RND uses a CNN-based target and predictor. Note that VBE runs at least three times faster than BDQN, ACB and RND.

In Figure 4 we see that VBE outperforms the other algorithms in four out of six environments. This result is starkly different from the prior work, and is due to the fact that we examine early learning. RND was originally trained on around 2 billion frames, and ACB on around 20 million steps with data coming from 128 parallel agents. In Pitfall all algorithms except ACB are competitive with VBE. In Gravitar, RND and ACB perform surprisingly well from the beginning. VBE can learn to perform as well with more training steps, as can be seen in the result in Appendix Section D. Overall, these results show that VBE scales to more complex deep RL settings and results in sample efficiency improvements in early learning in several Atari environments. We also compare VBE with DDQN-based variants of ACB and RND in Appendix E.

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.

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*, 2016.

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.

Art B. Owen and Dean Eckles. Bootstrapping data arrays of arbitrary order. *The Annals of Applied Statistics*, 6(3), sep 2012. doi: 10.1214/12-aoas547. URL https://doi.org/10.1214%2F12-aoas547.

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.

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.

A 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. ■

B Experiment Details

B.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 proportional to the distance of the agent from the center of the puddle, i.e., negative reward for being close 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 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. 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 hard exploration episodic grid world environment. In each state the agent can take two actions, left or right, which moves the agent down one row with column shifting based on left or right action. Collisions to the grid edges are handled by the agent staying in the same column but moving down one row. Since the agent can never access the states on the right side of the diagonal of the grid, the total number of states are thus $\frac{N \times (N+1)}{2}$. The most rewarding state is the state on the bottom right corner of the grid. To reach this the agent to take the action to go right every time. However, there is a penalty of $\frac{0.01}{N}$ for taking the action right, except for in this high rewarding state where the agent gets a reward of 1 for taking the right action. This makes it a very challenging environment. A policy that explores uniform randomly has the probability of 2^{-N} of reaching the goal state in each episode.

B.2 Algorithm Details

In the classic environments, every agent uses the same neural architecture, containing 2 non-linear layers with 50 nodes each and ReLU activation, followed by a linear output-layer. DQN-P, BDQN and all variants of VBE use target networks which are updated periodically after every τ steps. For DQN-P and BDQN we use $\tau = 4$ for all four classic environments. VBE and its variants 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, 2, 3 show the best performing sets of ensemble size k and bonus scale c for results in Sections 5.3, 5.4, 5.5.

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

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

C Linear function approximation

In this section we test VBE and the baseline agents on the same four classic environments as in Section 5.4, with tile-coded features and a single linear-layer network. For Riverswim we use the following tile-coding parameters: $(tiles = 4, tiling = 32, features = 128)$, Puddle world:

	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 ($k = 128$)	$c = 1.0$	$c = 3.0$	$c = 1.0$	$c = 10.0$
RND	$c = 1.0$	$c = 10.0$	$c = 10.0$	$c = 1.0$

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

	River Swim	Puddle World	Mountain Car	DeepSea
VBE	$k = 20, c = 1.0$	$k = 1, c = 10.0$	$k = 2, c = 1.0$	$k = 20, c = 1.0$
VB ACB	$k = 20, c = 10.0$	$k = 1, c = 1.0$	$k = 8, c = 1.0$	$k = 20, c = 1.0$
VB RND	$c = 10.0$	$c = 1.0$	$c = 1.0$	$c = 1.0$

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

($tiles = 5, tiling = 5, features = 128$), and Mountain car: ($tiles = 4, tiling = 16, features = 512$). The results in Figure 5 are similar to their neural network counterpart results in Figure 2. In Riverswim, RND does well and even surpasses BDQN in terms of performance. ACB, however, still fails on Riverswim. In Puddle world RND is comparable towards the end of training, and ACB is much slower. DQN-P outperforms BDQN in Puddle world, and Mountain car, whereas both ACB and RND fail in Mountain car. In Deepsea we see that DQN-P and ACB fail to learn the optimal policy. RND learns relatively quickly but then fails to stick to the optimal policy and thus collects less reward per episode throughout. In Figure 6 we compare VBE with VB ACB and RND in the linear setting. In Riverswim VB ACB does better than its PPO counter part in Figure 5a. VB ACB and RND perform comparable to VBE in Puddle world and Mountain car. Both VB ACB and RND fail in Deepsea.

D ATARI

In this section we compare two variants of VBE on the same six Atari games. VBE as mentioned in Section 5.6 only updates the final output layer of predictor RQFs. We implement a version of VBE, VBE-CNN which updates the complete CNN architecture for predictor RQFs. Since we are updating the complete network the predictor and target RQFs do not need same weights for the representation network. To ensure that the magnitude of errors is not too small, we initialize the target RQFs with a scale of 1 (scale parameter in Orthogonal initialization), and initialize predictor RQFs with a scale of 0.01. Note that this choice of scale was adopted from ACB's public implementation by Ash et al. (2022). We use an ensemble size of $k = 1$ for both these agents, and a bonus scale of $c = 10$ for all environments, except for Pitfall for which we use a bonus scale of $c = 1$ for all the agents. In Figure 7 we see that both agents continue to learn and improve at around 12 Million frames. VBE is better than VBE-CNN in Breakout, Pong and Privateye. However, in Pitfall VBE-CNN converges to a policy that results in no negative reward at around 6 Million frames. This result is quite impressive as Pitfall is a very hard exploration environment and most agents take a lot more data to learn a good policy in this environment. VBE does exceptionally well on Private Eye, which is also a very hard exploration environment. In Q*bert we see that both agents are comparable uptill 6 Million frames after which VBE-CNN first decreases in performance but eventually surpasses VBE towards the end. Finally, in Gravitar we see that both agents are competitive and continue to improve. Both variants of VBE do well in all six Atari environments, and demonstrate sample efficiency by early learning. Although we do not run the agents for as long as the baselines, VBE is still able to learn much quickly and in some environments even outperforms baselines trained on much more data.

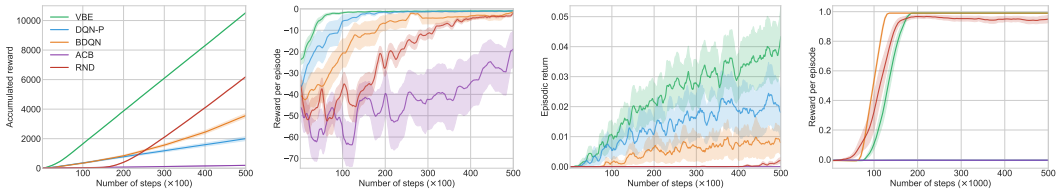
E ATARI: ALTERNATIVE CHOICES FOR THE VALUE BONUS

In this section we compare VBE with DDQN-based variants of ACB and RND, denoted as VB ACB and VB RND, on Atari to see early learning. In Figure-8 we can see that the VB ACB and VB RND are much more sample-efficient than their PPO-based counterparts. However, VBE still performs better in general. VBE performs the best in three out of six environments, i.e., in Breakout, Pong,

	VBE	VBE-CNN
Breakout	**170.17**	73.08
Pong	**18.81**	5.65
Qbert	963.73	**2026.31**
Pitfall	-18.65	**0.0**
Privateeye	**2416.05**	242.12
Gravitar	**247.76**	214.6

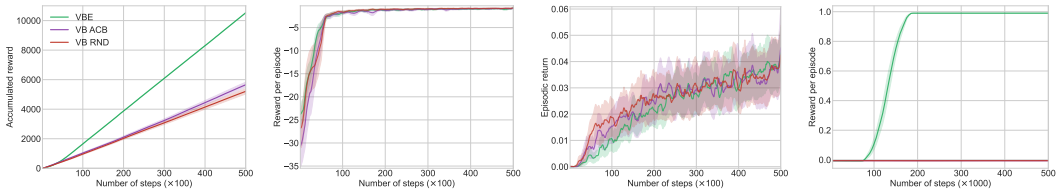
Table 4: Mean episodic return observed during the last 500 episodes of online training.

Privateeye. In Qbert, VBE is better than VB ACB, and in Pitfall VBE is better than VB RND. In Gravitar both VB ACB and VB RND perform better than VBE in the given frame budget, but we know from Figure-7 that VBE is slow early on but continues to improve beyond 5 million frames. Also, in Pitfall we note that beyond 5 million frames VBE surpasses VB ACB. We will update the paper with results on more frames in the final version.



(a) River swim (b) Puddle world (c) Mountain car (d) Deepsea

Figure 5: Comparing VBE with baseline agents in four classic control environments, with tile-coded features and a linear-layer.



(a) River swim (b) Puddle world (c) Mountain car (d) Deepsea

Figure 6: Comparing VBE with VB ACB and VB RND in the linear setting with tile-coded features.

F 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 average performance of VBE used in Section 5.4, for each environment. For Riverswim we see that the performance improves as the bonus scale and the ensemble size is increased. This makes sense as Riverswim 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 rquire 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. For the single linear-layer agent we observe a similar patter in Figure 10.

G Target policy experiments

In many cases it is straight forward to define the agent's behaviour policy depending on the problem at hand, for example, for exploration an agent can use an $\epsilon-$greedy policy, or use an upper

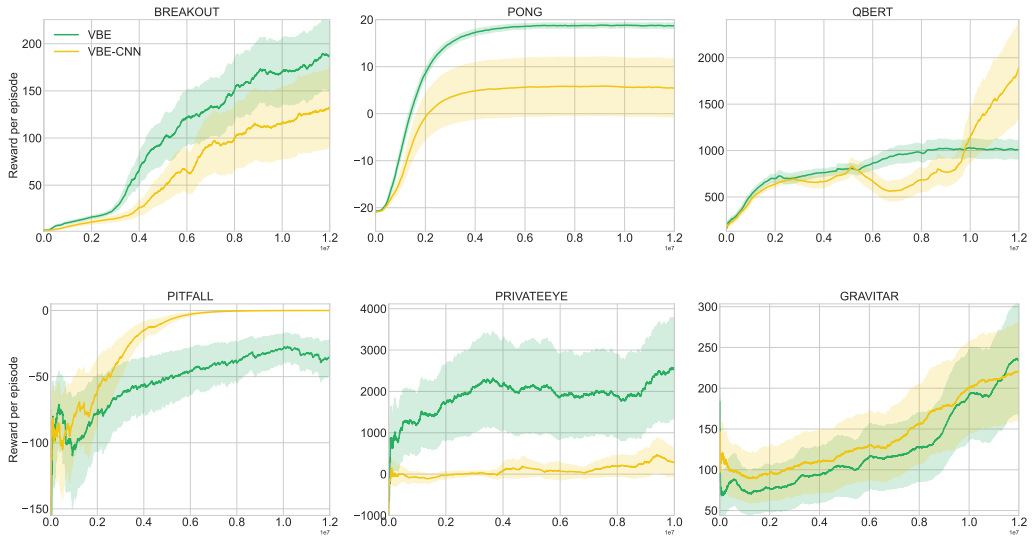


Figure 7: Comparing the online performance of VBE and VBE-CNN on six Atari environments. For Private Eye we compare the agents for 10 Million steps, and the rest for 12 Million steps.

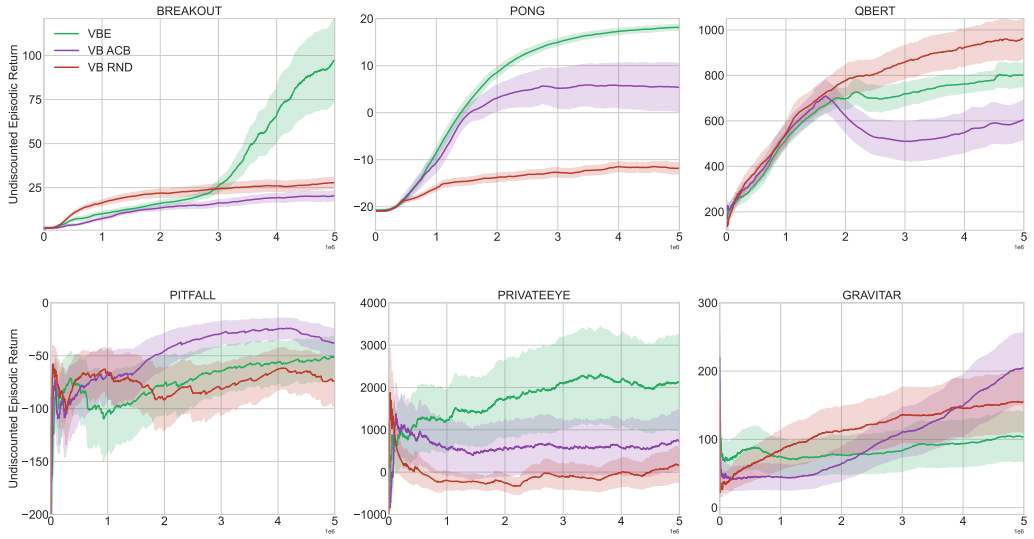
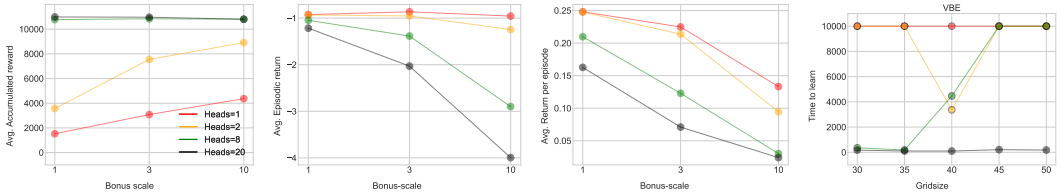


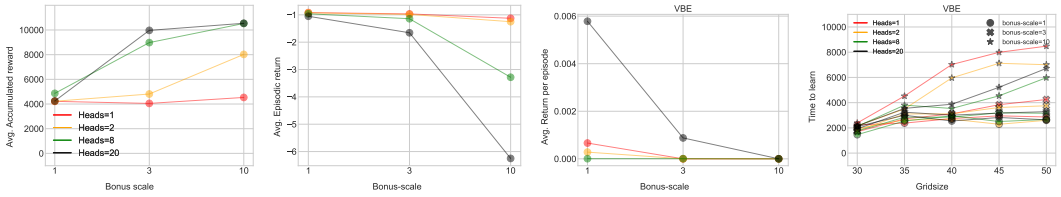
Figure 8: Comparing the online performance of VBE, VB ACB and VB RND on six Atari environments on a budget of 5 million steps to see early learning.

confidence style bonus to select actions, lets call this the optimistic policy. However, in case of the general policy iteration setting it is not clear what the target policy should be to perform updates. Should the target policy also be optimistic/on-policy? Or should it be greedy only with respect to the value estimates/off-policy? The question becomes especially challenging to answer when there are multiple value functions or an ensemble of value functions, like in our case. What should be the target policy to update the RQFs? To investigate we performed a series of experiments using different types of target policies, i.e., optimistic, and greedy. The optimistic agent uses the optimistic policy to update the value head and the RQFs. The greedy agent uses the greedy target policy to update both the value head and the RQFs. The target policy for the value heads and the RQFs is consistent, this is because we want the RQFs to correspond to the value estimates. The intuition behind using an optimistic policy is that the target action-values it gives may have high prediction errors and up-



(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 single linear-layer agent.

dating the RQFs by bootstrapping off of values with high prediction error is likely to produce errors in current RQF predictions as well. This allows uncertainty to propagates allowing the agent to do directed exploration. In case of the greedy target policy, the uncertainty still propagates however, in this case the prediction errors don't play a role in the selection of action-values, for example, an action-value with a low value but a high prediction error may not be selected using a greedy target policy.

In Figure 11 we show two different agents, optimistic and greedy, run on different grid sizes of DeepSea with different ensemble size, represented by color, and different bonus scales, represented by shapes. The agents use a single linear-layer for the 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 more exploration. The greedy agent performs well even with multiple RQFs, whereas optimistic agent fails to learn the optimal policy as the number of RQFs increase. We use the greedy target policy version of VBE in all the results of the main body.

In our experiments with VB with ACB and RND, we noticed a strange phenomenon, i.e., using an optimistic target policy allows VB ACB and VB RND to learn the optimal policy quickly on DeepsSa environments (Figure 12), and using a greedy target policy for VB ACB/RND would cause the agents fail to learn the optimal policy (Figure 6d, 3d). This is interesting, as reward bonus methods do not provide optimism for unseen action-values, VB with ACB and RND should not be able to cover the entire state space based on random initialization. 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 is non-episodic. Using the optimistic target policy and with the help of momentum the bias term consistently increases, consequently the intrinsic action values start to increase. Since the bias-term is a shared parameter, the increase in its value provides the optimism for unseen action-values as well, this allows for the agent to cover the state space and thus learn the optimal policy. In case of the greedy target policy the intrinsic values do no increase, thus the agent fails to cover the state space. In Section 5.3, we show that if we use tabular features and a linear-layer without any bias term then VB ACB/RND fail to cover the state space. In this setting the agent's behaviour and target policy is governed by only the intrinsic value functions (on-policy), however it fails on account of not providing first-visit optimism.

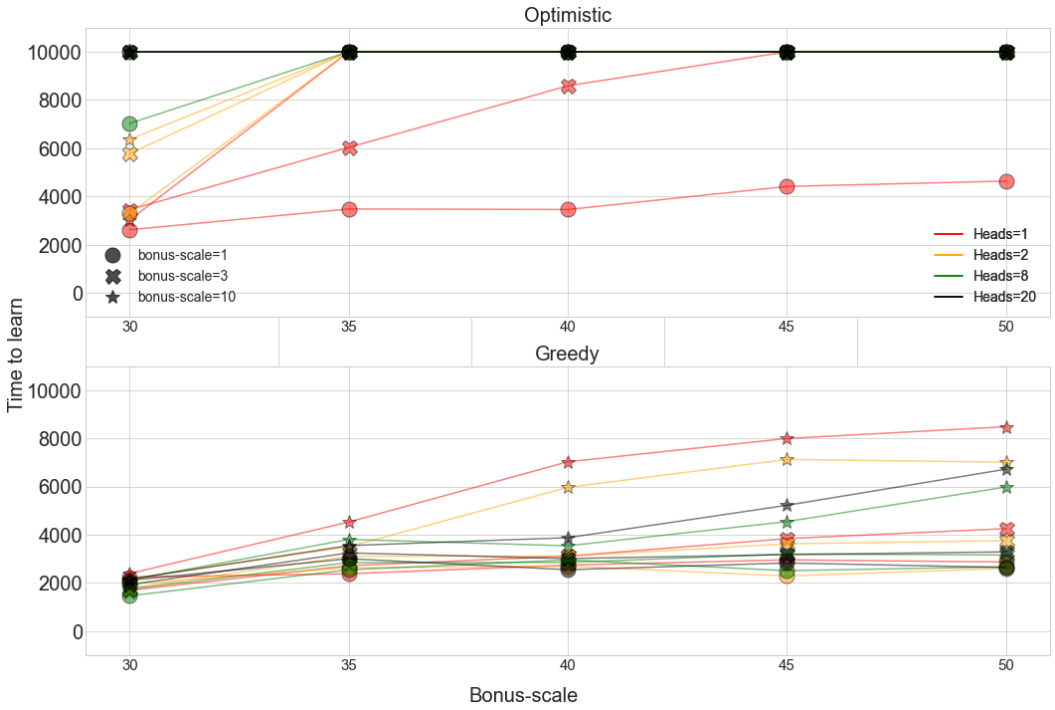
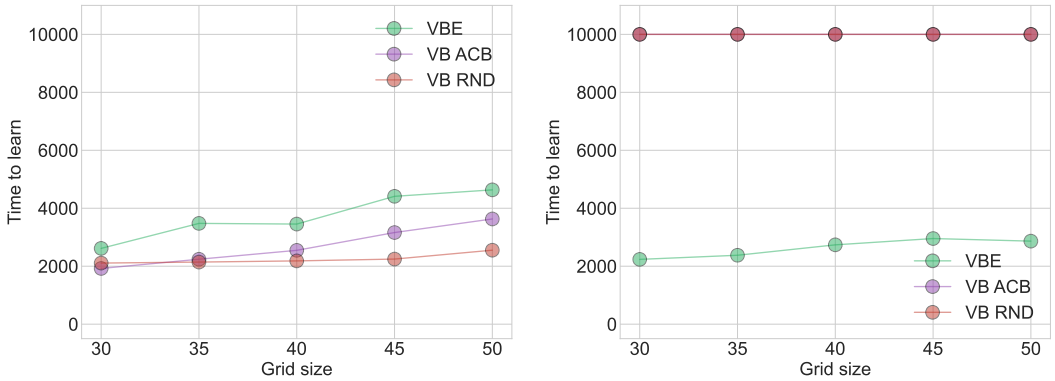


Figure 11: Comparing Optimistic target policy (On-policy) with Greedy target policy (Off-policy) on different grid sizes of Deepsea.



(a) Optimistic target policy

(b) Greedy target policy

Figure 12: In Figure 12a the agents do on-policy (optimistic) updates. In Figure 12b the agents do off-policy (greedy) updates. VB ACB/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.