Principled Exploration via Optimistic Bootstrapping and Backward Induction (Appendix)

A. UCB Bonus in OB2I

Recall that we consider the following regularized least-square problem,

$$w_t \leftarrow \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{\tau=0}^{m} \big[r_t(s_t^\tau, a_t^\tau) + \max_{a \in \mathcal{A}} Q_{t+1}(s_{t+1}^\tau, a) - w^\top \phi(s_t^\tau, a_t^\tau)\big]^2 + \lambda \|w\|^2. \tag{7}$$

In the sequel, we consider a Bayesian linear regression perspective of (7) that captures the intuition behind the UCB-bonus in OB2I. Our objective is to approximate the action-value function Q_t via fitting the parameter w, such that

$$w^\top \phi(s_t, a_t) \approx r_t(s_t, a_t) + \max_{a \in \mathcal{A}} Q_{t+1}(s_{t+1}, a),$$

where Q_{t+1} is given. We assume that we are given a Gaussian prior of the initial parameter $w \sim \mathcal{N}(0, \mathbf{I}/\lambda)$. With a slight abuse of notation, we denote by w_t the Bayesian posterior of the parameter w given the set of independent observations $\mathcal{D}_m = \{(s_t^\tau, a_t^\tau, s_{t+1}^\tau)\}_{\tau \in [0,m]}$. We further define the following noise with respect to the least-square problem in (7),

$$\epsilon = r_t(s_t, a_t) + \max_{a \in \mathcal{A}} Q_{t+1}(s_{t+1}, a) - w^\top \phi(s_t, a_t), \tag{8}$$

where (s_t, a_t, s_{t+1}) follows the distribution of trajectory. The following theorem justifies the UCB-bonus in OB2I under the Bayesian linear regression perspective.

Theorem 2 (Formal Version of Theorem 1)**.** *We assume that ϵ follows the standard Gaussian distribution $\mathcal{N}(0,1)$ given the state-action pair (s_t, a_t) and the parameter w. Let w follows the Gaussian prior $\mathcal{N}(0, \mathbf{I}/\lambda)$. We define*

$$\Lambda_t = \sum_{\tau=0}^{m} \phi(x_t^\tau, a_t^\tau)\phi(x_t^\tau, a_t^\tau)^\top + \lambda \cdot \mathbf{I}. \tag{9}$$

It then holds for the posterior of w_t given the set of independent observations $\mathcal{D}_m = \{(s_t^\tau, a_t^\tau, s_{t+1}^\tau)\}_{\tau \in [0,m]}$ that

$$\mathrm{Var}\big(\phi(s_t, a_t)^\top w_t\big) = \mathrm{Var}\big(\tilde{Q}_t(s_t, a_t)\big) = \phi(s_t, a_t)^\top \Lambda_t^{-1} \phi(s_t, a_t), \quad \forall (s_t, a_t) \in \mathcal{S} \times \mathcal{A}.$$

Here we denote by $\tilde{Q}_t = w_t^\top \phi$ the estimated action-value function.

Proof. The proof follows the standard analysis of Bayesian linear regression. See, e.g., West (1984) for a detailed analysis. We denote the target of the linear regression in (7) by

$$y_t = r_t(s_t, a_t) + \max_{a \in \mathcal{A}} Q_{t+1}(s_{t+1}, a).$$

By the assumption that ϵ follows the standard Gaussian distribution, we obtain that

$$y_t \,|\, (s_t, a_t), w \sim \mathcal{N}\big(w^\top \phi(s_t, a_t), 1\big). \tag{10}$$

Recall that we have the prior distribution $w \sim \mathcal{N}(0, \mathbf{I}/\lambda)$. Our objective is to compute the posterior density $w_t = w \,|\, \mathcal{D}_m$, where $\mathcal{D}_m = \{(s_t^\tau, a_t^\tau, s_{t+1}^\tau)\}_{\tau \in [0,m]}$ is the set of observations. It holds from Bayes rule that

$$\log p(w \,|\, \mathcal{D}_m) = \log p(w) + \log p(\mathcal{D}_m \,|\, w) + Const., \tag{11}$$

where $p(\cdot)$ denote the probability density function of the respective distributions. Plugging (10) and the probability density function of Gaussian distribution into (11) yields

$$\begin{aligned}\log p(w \,|\, \mathcal{D}_m) &= -\|w\|^2/2 - \sum_{\tau=1}^{m} \|w^\top \phi(s_t^\tau, a_t^\tau) - y_t^\tau\|^2/2 + Const. \\ &= -(w-\mu_t)^\top \Lambda_t^{-1}(w-\mu_t)/2 + Const.,\end{aligned} \tag{12}$$

where we define

$$\mu_t = \Lambda_t^{-1}\sum_{\tau=1}^{m} \phi(s_t^\tau, a_t^\tau) y_t^\tau, \qquad \Lambda_t = \sum_{\tau=0}^{m} \phi(x_t^\tau, a_t^\tau)\phi(x_t^\tau, a_t^\tau)^\top + \lambda \cdot \mathbf{I}.$$

Thus, by (12), we obtain that $w_t = w \,|\, \mathcal{D}_m \sim \mathcal{N}(\mu_t, \Lambda_t^{-1})$. It then holds for all $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$ that

$$\text{Var}\big(\phi(s_t, a_t)^\top w_t\big) = \text{Var}\big(\tilde{Q}_t(s_t, a_t)\big) = \phi(s_t, a_t)^\top \Lambda_t^{-1} \phi(s_t, a_t),$$

which concludes the proof of Theorem 2. □

Remark 1 (Extension to Neural Network Parameterization)**.** We remark that our proof can be extended to explain deep neural network parametrization under the overparameterized network regime (Arora et al., 2019). Under such a setting, a two-layer neural network $f(\cdot; W)$ with parameter W and ReLU activation function can be approximated by

$$f(x; W) \approx f(x; W_0) + \phi_{W_0}(x)^\top (W - W_0) = \phi_{W_0}(x)^\top W, \quad \forall x \in \mathcal{X},$$

where the approximation holds if the neural network is sufficiently wide (Arora et al., 2019). Here W_0 is the Gaussian distributed initial parameter and $\phi_{W_0} = ([\phi_{W_0}]_1, \ldots, [\phi_{W_0}]_m)^\top$ is the feature embedding defined as follows,

$$[\phi_{W_0}(x)]_r = \frac{1}{\sqrt{m}}\sigma\big(x^\top [W_0]_r\big), \quad \forall x \in \mathcal{X},\ r \in [m].$$

Hence, if we consider a Bayesian perspective of training neural network, where the parameter W is obtained by solving a Bayesian linear regression with the feature ϕ_{W_0}, then the proof of Theorem 2 can be applied to the setting upon conditioning on the random initialization W_0. Thus, Theorem 2 applies to the neural network parameterization under such an overparameterized neural network regime.

B. Algorithmic Description

Algorithm 2 OB2I in DRL

```
1: Initialize: replay buffer $\mathcal{D}$, bootstrapped $Q$-network $Q(\cdot;\theta)$ and target network $Q(\cdot;\theta^-)$
2: Initialize: total training frames $H = 20$M, current frame $h = 0$
3: while $h < H$ do
4:   Pick a bootstrapped $Q$-function to act by sampling $k \sim \mathrm{Unif}\{1, \ldots, \mathrm{K}\}$
5:   Reset the environment and receive the initial state $s_0$
6:   for step $i = 0$ to Terminal do
7:     With $\epsilon$-greedy choose $a_i = \mathrm{argmax}_a\, Q^k(s_i, a)$
8:     Take action and observe $r_i$ and $s_{i+1}$, then save the transition in buffer $\mathcal{D}$
9:     if $h$ % training frequency $= 0$ then
10:      Sample an episodic experience $E = \{\mathbf{S}, \mathbf{A}, \mathbf{R}, \mathbf{S}'\}$ with length $T$ from $\mathcal{D}$
11:      Initialize a $Q$-table $\tilde{\mathbf{Q}} = Q(\mathbf{S}', \mathcal{A}; \theta^-) \in \mathbb{R}^{K\times|\mathcal{A}|\times T}$ by the target $Q$-network
12:      Compute the UCB-bonus for immediate reward for all steps to construct $\mathbf{B} \in \mathbb{R}^T$
13:      Compute the action matrix $\tilde{\mathbf{A}} = \mathrm{argmax}_a\, \tilde{\mathbf{Q}}[\cdot, a, \cdot] \in \mathbb{R}^{K\times T}$ to gather all $a'$ of next-$Q$
14:      Compute the UCB-bonus for next-$Q$ for all heads and all steps to construct $\tilde{\mathbf{B}} \in \mathbb{R}^{K\times T}$
15:      Compute the mask matrix $\mathbf{M} \in \mathbb{R}^{K\times T}$ where $\mathbf{M}[k, t] = \mathbb{1}_{\tilde{\mathbf{A}}[k,t] \neq \mathbf{A}_{t+1}}$
16:      Initialize target table $\mathbf{y} \in \mathbb{R}^{K\times T}$ to zeros, and set $\mathbf{y}[\cdot, T-1] = \mathbf{R}_{T-1} + \alpha_1 \mathbf{B}_{T-1}$
17:      for $t = T-2$ to $0$ do
18:        $\tilde{\mathbf{Q}}[\cdot, a_{t+1}, t] \leftarrow \beta \mathbf{y}[\cdot, t+1] + (1-\beta)\tilde{\mathbf{Q}}[\cdot, a_{t+1}, t]$
19:        $\mathbf{y}[\cdot, t] \leftarrow \big(\mathbf{R}_t + \alpha_1 \mathbf{B}_t\big) + \gamma\big(\tilde{\mathbf{Q}}[\cdot, a', t] + \alpha_2 \mathbf{M}[\cdot, t] \circ \tilde{\mathbf{B}}[\cdot, t]\big)$ where $a' = \tilde{\mathbf{A}}[\cdot, t]$
20:      end for
21:      Compute the $Q$-value of $(\mathbf{S}, \mathbf{A})$ for all heads as $\mathbf{Q} = Q(\mathbf{S}, \mathbf{A}; \theta) \in \mathbb{R}^{K\times T}$
22:      Perform a gradient descent step on $(\mathbf{y} - \mathbf{Q})^2$ with respect to $\theta$
23:    end if
24:    Every $C$ steps reset $\theta^- \leftarrow \theta$
25:    $h \leftarrow h + 1$
26:  end for
27: end while
```

Remark 2 (Remark on ϵ-Greedy)**.** We adopt the ϵ-greedy technique based on the empirical concerns. Empirically, ϵ-greedy is helpful at the early stage of training, since the bootstrapped Q-heads typically lack diversity at the early stage of training. As shown in Figure 3, the bonus for OB2I is small at the begining of training. A similar observation also arises in Randomized Prior Function (Osband et al., 2018), where each head is initialized together with a random but fixed prior function to improves the diversity between Q-heads at the initialization. In OB2I, we use ϵ-greedy as an empirical technique to improve the diversity of Q-heads at the beginning of training while diminishing ϵ-term to zero as the training evolves. For a fair comparison, in our experiments, we preform ϵ-greedy for all BEBU-based baselines (BEBU, BEBU-UCB, and BEBU-IDS) with the same values of ϵ. We remark that the ϵ-greedy technique is also widely used in implementations of methods based on Bootstrapped DQN, including Bootstrapped DQN implementation at `https://github.com/johannah/bootstrap_dqn`, `https://github.com/rrmenon10/Bootstrapped-DQN`, Sunrise (Chen et al., 2017; Lee et al., 2020) implementation at `https://github.com/pokaxpoka/sunrise`, and the official IDS (Nikolov et al., 2019) implementation at `https://github.com/nikonikolov/rltf`. NoisyNet (Fortunato et al., 2018) implementation also applies this technique at `https://github.com/Kaixhin/Rainbow`.

In addition, from a theoretical perspective, adopting ϵ-greedy policies in place of greedy policies will hinder the performance difference term $\langle \pi^k, Q^* - Q^k \rangle$ in the analysis of LSVI-UCB (Jin et al., 2020), which is upper bounded by zero if π^k is the greedy policy corresponding to Q^k. In contrast, if π^k is the ϵ-greedy policy, adding and subtracting the greedy policy yields an $\epsilon Q_{\max}$ upper bound, which propagates to an additional $O(\epsilon T)$ term in the regret. Therefore, if ϵ is sufficiently small, the algorithm attains the optimal $\sqrt{T}$-regret. In OB2I, we diminish ϵ-term to zero as the training evolves, which does not incur a large bias to the regret.

Algorithm 3 BEBU & BEBU-UCB & BEBU-IDS

```
1: Input: Algorithm Type (BEBU, BEBU-UCB, or BEBU-IDS)
2: Initialize: replay buffer $\mathcal{D}$, bootstrapped $Q$-network $Q(\cdot;\theta)$ and target network $Q(\cdot;\theta^-)$
3: Initialize: total training frames $H = 20\text{M}$, current frame $h = 0$
4: while $h < H$ do
5:   Pick a bootstrapped $Q$-function to act by sampling $k \sim \text{Unif}\{1,\dots,\text{K}\}$
6:   Reset the environment and receive the initial state $s_0$
7:   for step $i = 0$ to Terminal do
8:     if Algorithm type is BEBU then
9:       With $\epsilon$-greedy choose $a_i = \text{argmax}_a\, Q^k(s_i, a)$
10:    else if Algorithm type is BEBU-UCB then
11:      With $\epsilon$-greedy choose $a_i = \text{argmax}_a[\bar{Q}(s_i,a) + \alpha\cdot\sigma(Q(s_i,a))]$, where $\bar{Q}(s_i,a_i) = \frac{1}{K}\sum_{k=1}^{K} Q^k(s_i,a_i)$ and $\sigma(Q(s_i,a_i)) = \sqrt{\frac{1}{K}\sum_{k=1}^{K}(Q^k(s_i,a_i) - \bar{Q}(s_i,a_i))^2}$ are the mean and standard deviation of the bootstrapped Q-estimates
12:    else if Algorithm type is BEBU-IDS then
13:      With $\epsilon$-greedy choose $a_i = \text{argmin}_a \frac{\hat{\Delta}_i(s_i,a)^2}{I_i(s_i,a)}$ by following the regret-information ratio, where $\hat{\Delta}_i(s_i,a_i) = \max_{a'\in\mathcal{A}} u_i(s_i,a') - l_i(s_i,a_i)$ is the expected regret, and $[l_i(s_i,a_i), u_i(s_i,a_i)]$ is the confidence interval. In particular, $u_i(s_i,a_i) = \bar{Q}(s_i,a_i) + \lambda_{\text{ids}}\cdot\sigma(Q(s_i,a_i))$ and $l_i(s_i,a_i) = \bar{Q}(s_i,a_i) - \lambda_{\text{ids}}\cdot\sigma(Q(s_i,a_i))$. $I(s_i,a_i) = \log(1 + \sigma(Q(s_i,a_i))^2/\rho^2) + \epsilon_{\text{ids}}$ measures the uncertainty, where $\rho$ and $\epsilon_{\text{ids}}$ are constants.
14:    else
15:      Algorithm type error.
16:    end if
17:    Take action and observe $r_i$ and $s_{i+1}$, then save the transition in buffer $\mathcal{D}$
18:    if $h$ % training frequency $= 0$ then
19:      Sample an episodic experience $E = \{\mathbf{S}, \mathbf{A}, \mathbf{R}, \mathbf{S}'\}$ with length $T$ from $\mathcal{D}$
20:      Initialize a $Q$-table $\tilde{\mathbf{Q}} = Q(\mathbf{S}', \mathcal{A}; \theta^-) \in \mathbb{R}^{K\times|\mathcal{A}|\times T}$ by the target $Q$-network
21:      Compute the action matrix $\tilde{\mathbf{A}} = \text{argmax}_a\, \tilde{\mathbf{Q}}[\cdot, a, \cdot] \in \mathbb{R}^{K\times T}$ to gather all $a'$ of next-$Q$
22:      Initialize target table $\mathbf{y} \in \mathbb{R}^{K\times T}$ to zeros, and set $\mathbf{y}[\cdot, T-1] = \mathbf{R}_{T-1} + \alpha_1 \mathbf{B}_{T-1}$
23:      for $t = T-2$ to $0$ do
24:        $\tilde{\mathbf{Q}}[\cdot, a_{t+1}, t] \leftarrow \beta\mathbf{y}[\cdot, t+1] + (1-\beta)\tilde{\mathbf{Q}}[\cdot, a_{t+1}, t]$
25:        $\mathbf{y}[\cdot, t] \leftarrow \mathbf{R}_t + \gamma\tilde{\mathbf{Q}}[\cdot, a', t]$ where $a' = \tilde{\mathbf{A}}[\cdot, t]$
26:      end for
27:      Compute the $Q$-value of $(\mathbf{S}, \mathbf{A})$ for all heads as $\mathbf{Q} = Q(\mathbf{S}, \mathbf{A}; \theta) \in \mathbb{R}^{K\times T}$
28:      Perform a gradient descent step on $(\mathbf{y} - \mathbf{Q})^2$ with respect to $\theta$
29:    end if
30:    Every $C$ steps reset $\theta^- \leftarrow \theta$
31:    $h \leftarrow h + 1$
32:  end for
33: end while
```

Remark 3 (Remark on Computational Efficiency)**.** We remark that OB2I requires much less training time than BEBU-UCB and BEBU-IDS, since both BEBU-UCB and BEBU-IDS requires computing the corresponding confidence bounds in each time step of interaction. In contrast, OB2I only requires estimating the confidence bound for batch training. Meanwhile, the number of interaction steps L_1 with the environment are typically set to be much larger than the number of training steps L_2 (e.g., in DQN, $L_1 \approx 4L_2$). Hence, OB2I is more computational efficient under such a conventional setting.

C. Additional Experiment: MNIST Maze

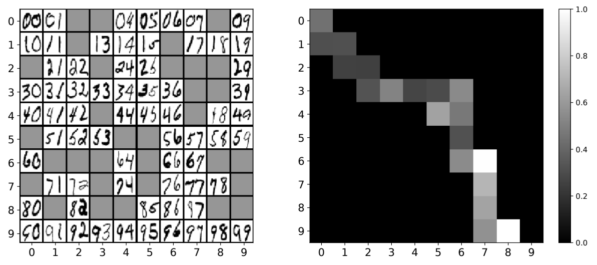
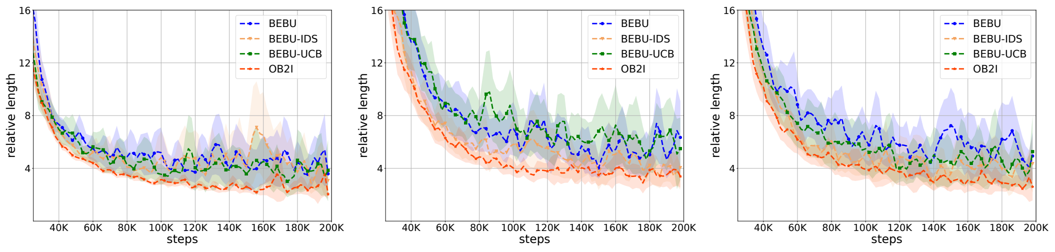


Figure 4. An example MNIST maze (left) and the UCB-bonuses in the agent's path (right).



(a) Wall density of 30% (b) Wall density of 40% (c) Wall density of 50%

Figure 5. Results of 200K steps training of MNIST maze with different wall-density setup.

We use 10×10 MNIST maze with randomly placed walls to evaluate our method. The agent starts from the initial position $(0, 0)$ in the upper-left of the maze and aims to reach the goal position $(9, 9)$ in the bottom-right. The state of position (i, j) is represented by stacking two randomly sampled images with label i and j from the MNIST dataset. When the agent steps to a new position, the state representation is reconstructed by sampling images. Hence the agent gets different states even stepping to the same location twice, which minimizes the correlation among locations. Meanwhile, we introduce additional stochasticity in the transition probability. Specifically, the agent has a probability of 10% to arrive in the adjacent locations when taking an action. For example, when taking action 'left', the agent has a 10% chance of transiting to 'up', and a 10% chance of transiting to 'down'. The agent gets a reward of -1 when bumping into a wall, and gets 1000 when reaching the goal.

We use the different setup of wall-density in the experiment. Here the wall-density means the proportion of walls among all the locations. Figure 4 (left) shows a generated maze with wall-density of 50%, where the gray positions represent walls. We train all methods with wall-density of 30%, 40%, and 50%. For each setup, we train 50 independent agents for 50 randomly generated mazes. We use the relative length defined by $l_{\text{agent}}/l_{\text{best}}$ to evaluate the performance of algorithms, where l_{agent} is the length of the agent's travel to reach the goal in an episode (maximum steps are 1000), and l_{best} is the length of the shortest path to reach the goal. The performance comparison is shown in Figure 5. We observe that OB2I performs the best among all the methods. In addition, BEBU-IDS also performs well. To further illustrate the performance of OB2I, We use a trained OB2I agent to take action in the maze presented in Figure 4 (left). We present the corresponding UCB-bonuses of state-action pairs along the agent's visitation trajectory in Figure 4 (right).

We observe that OB2I assigns high UCB-bonus to positions that are critical to exploration. For example, the state-action pairs in location $(3, 3)$ and $(6, 7)$ are assigned high UCB-bonus as they are the bottleneck positions in the maze illustrated in Figure 4 (left), where the agent must visit $(3, 3)$ and $(6, 7)$ to reach the goal at $(9, 9)$. The UCB-bonus encourages the agent to walk through these bottleneck positions correctly. We refer to Appendix E for additional examples.

D. Implementation Detail

D.1. MNIST Maze

Hyper-parameters of BEBU. BEBU is the basic algorithm of BEBU-UCB and BEBU-IDS. BEBU uses the same network-architecture as Bootstrapped DQN (Osband et al., 2016). The diffusion factor and other training parameters are set by following EBU paper (Lee et al., 2019). Details are summarized in Table 3.

Table 3. Hyper-parameters of BEBU for MNIST-Maze

Hyperparameters	Value	Description
state space	$28 \times 28 \times 2$	Stacking two images sampled from MNIST dataset with labels according to the agent's current location.
action space	4	Including left, right, up, and down.
K	10	Number of bootstrapped heads.
network-architecture	conv(64,4,4) conv(64,3,1) dense$\{512, 4\}_{k=1}^{K}$	Using convolution (channels, kernel size, stride) layers first, then fully connected into K bootstrapped heads. Each head has 512 ReLUs and 4 linear units.
gradient norm	10	The gradient is clipped by 10. The gradient of each head is normalize by $1/K$ according to bootstrapped DQN.
learning starts	10000	The agent takes actions according to the initial policy before learning starts.
replay buffer size	170	A simple replay buffer is used to store episodic experience.
training frequency	50	Number of action-selection step between successive gradient descent steps.
H	200,000	Total timesteps to train a single maze.
target network update frequency	2000	The target-network is updated every 2000 steps.
optimizer	Adam	Adam optimizer is used for training. Detailed parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon_{\rm ADAM} = 10^{-7}$.
learning rate	0.001	Learning rate for Adam optimizer.
ϵ	$\frac{(h-H)^2}{H^2}$	Exploration factor. H is the total timesteps for training, and h is the current timestep. ϵ starts from 1 and is annealed to 0 in a quadratic manner.
γ	0.9	Discount factor.
β	1.0	Diffusion factor of backward update.
wall density	30%, 40%, and 50%	Proportion of walls in all locations of the maze.
reward	-1 or 1000	Reward is -1 when bumping into a wall, and 1000 when reaching the goal.
stochasticity	10%	Has a probability of 10% to arrive in the adjacent locations when taking an action.
evaluation metric	$l_{\rm rel} = l_{\rm agent}/l_{\rm best}$	Ratio between length of the agent's path and the best length.

Hyper-parameters of BEBU-UCB. BEBU-UCB uses the upper-bound of Q-values to select actions. In particular, $a = \arg\max_{a\in\mathcal{A}} [\mu(s,a) + \lambda_{\rm ucb}\sigma(s,a)]$, where $\mu(s,a)$ and $\sigma(s,a)$ are the mean and standard deviation of bootstrapped Q-values $\{Q^k(s,a)\}_{k=1}^{K}$. We use $\lambda_{\rm ucb} = 0.1$ in our experiment.

Hyper-parameters of BEBU-IDS. The action-selection in IDS (Nikolov et al., 2019) follows the regret-information ratio as $a_t = {\rm argmin}_{a\in\mathcal{A}} \frac{\hat{\Delta}_t(s,a)^2}{I_t(s,a)}$, which balances the regret and exploration. $\hat{\Delta}_t(s,a)$ is the expected regret that indicates the loss of reward when choosing a suboptimal action a. IDS uses a conservative estimate of regret, namely, $\hat{\Delta}_t(s,a) = \max_{a'\in\mathcal{A}} u_t(s,a') - l_t(s,a)$, where $[l_t(s,a), u_t(s,a)]$ is the confidence interval of action-value function. In particular, $u_t(s,a) = \mu(s,a) + \lambda_{\rm ids}\sigma(s,a)$ and $l_t(s,a) = \mu(s,a) - \lambda_{\rm ids}\sigma(s,a)$, where $\mu(s,a)$ and $\sigma(s,a)$ are the mean and standard deviation of bootstrapped Q-values $\{Q^k(s,a)\}_{k=1}^{K}$. The information gain $I_t(a)$ measures the uncertainty of action-values by $I(s,a) = \log(1 + \frac{\sigma(s,a)^2}{\rho(s,a)^2}) + \epsilon_{\rm ids}$, where $\rho(s,a)$ is the variance of the return distribution, which can be measured by C51 (Bellemare et al., 2017) in distributional RL and is a constant in ordinary Q-learning. We set $\lambda_{\rm ids} = 0.1$, $\rho(s,a) = 1.0$, and $\epsilon_{\rm ids} = 10^{-5}$ for our experiment.

Hyper-parameters of OB2I. We set α_1 and α_2 to be 0.01 for our experiments. We find that adding a normalizer to UCB-bonus $\tilde{\mathbf{B}}$ of the next-Q value enables more stable performance. A similar technique was used in Burda et al. (2019a). Specifically, we divide $\tilde{\mathbf{B}}$ by a running estimate of its standard deviation. Since the UCB-bonuses for next-Q are typically different among the Q-heads, such a normalization allows Q-networks to have a smooth and stable update.

D.2. Atari games

Hyper-parameters of BEBU. We adopt the same basic setting of the Atari environment as (Mnih et al., 2015) and (Lee et al., 2019). We summarize the details to Table 4.

Table 4. Hyper-parameters of BEBU for Atari games

Hyperparameters	Value	Description
state space	$84 \times 84 \times 4$	Stacking 4 recent frames as the input to network.
action repeat	4	Repeating each action 4 times.
K	10	The number of bootstrapped heads.
network-architecture	conv(32,8,4) conv(64,4,2) conv(64,3,1) dense$\{512, \lvert\mathcal{A}\rvert\}_{k=1}^{K}$	Using convolution(channels, kernel size, stride) layers first, then fully connected into K bootstrapped heads. Each head has 512 ReLUs and $\lvert\mathcal{A}\rvert$ linear units.
gradient norm	10	The gradient is clipped by 10, and also be normalize by $1/K$ for each head by following bootstrapped DQN.
learning starts	50000	The agent takes random actions before learning starts.
replay buffer size	1M	The number of recent transitions stored in the replay buffer.
training frequency	4	The number of action-selection step between successive gradient steps.
H	20M	Total frames to train an environment.
target network update frequency	10000	The target-network is updated every 10000 steps.
optimizer	Adam	Detailed Adam parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon_{\text{ADAM}} = 10^{-7}$.
mini-batch size	32	The number of training cases for gradient decent each time.
learning rate	0.00025	Learning rate for Adam optimizer.
initial exploration	1.0	Initial value of ϵ in ϵ-greedy exploration.
final exploration	0.1	Final value of ϵ in ϵ-greedy exploration.
final exploration frames	1M	The number of frames that the initial value of ϵ linearly annealed to the final value.
γ	0.99	Discount factor.
β	0.5	Diffusion factor of backward update.
ϵ_{eval}	0.05	Exploration factor in ϵ-greedy for evaluation.
evaluation policy	ensemble vote	The same evaluation method as in Bootstrapped DQN (Osband et al., 2016).
evaluation length	108000	The policy is evaluated for 108000 steps.
evaluation frequency	100K	The policy is evaluated every 100K steps.
max no-ops	30	Maximum number no-op actions before an episode starts.

Hyper-parameters of BEBU-UCB. BEBU-UCB selects actions by $a = \arg\max_{a\in\mathcal{A}} [\mu(s,a) + \lambda_{\text{ucb}}\sigma(s,a)]$. The detail is given in Appendix D.1. We use $\lambda_{\text{ucb}} = 0.1$ in our experiment by searching coarsely.

Hyper-parameters of BEBU-IDS. The action-selection follows the regret-information ratio as $a_t = \text{argmin}_{a\in\mathcal{A}} \frac{\hat{\Delta}_t(s,a)^2}{I_t(s,a)}$. See detail in Appendix D.1. We use $\lambda_{\text{ids}} = 0.1$, $\rho(s,a) = 1.0$ and $\epsilon_{\text{ids}} = 10^{-5}$ in our experiment by searching coarsely.

Hyper-parameters of OB2I. We set α_1 and α_2 to the same value of 0.5×10^{-4}. The UCB-bonus $\tilde{\mathbf{B}}$ for the next-Q value is normalized by dividing a running estimate of its standard deviation to have a stable performance.

Implementation of Bayesian-DQN. Since Bayesian-DQN is not evaluated in the whole Atari suite, we adopt the official release code in `https://github.com/kazizzad/BDQN-MxNet-Gluon` and make two modification for a fair comparison. (1) We add the 30 no-op evaluation mechanism, which we use to evaluate OB2I and other baselines in our work. (2) We set the frame-skip to 4 to be consistent with our baselines. We remark that inconsistency still exists since the original implementation of Bayesian-DQN is based on MX-Net Library, while OB2I and other baselines are implemented with Tensorflow. We release the modified code in `https://github.com/review-anon/Bayesian-DQN`.

Results of DQN, UBE, BootDQN, Noisy-Net, and BootDQN-IDS. These methods have been evaluated by the whole Atari suite. We directly adopt the scores reported in the corresponding articles (Mnih et al., 2015; O'Donoghue et al., 2018; Osband et al., 2016; Fortunato et al., 2018; Nikolov et al., 2019). However, we remark that inconsistency in the comparison exists since (1) UBE, BootDQN, and BootDQN-IDS use double Q-learning, and (2) Noisy-Net uses both the double Q-learning and dueling networks, in their original implementations. (3) In contrast, DQN, OB2I and BEBU-based baselines all use the standard Q-learning without advanced techniques.

E. Visualizing OB2I

OB2I uses the UCB-bonus that indicates the disagreement of bootstrapped Q-estimates to measure the uncertainty of Q-functions. The state-action pairs with high UCB-bonuses signify the bottleneck positions or meaningful events. We provide visualization in several tasks to illustrate the effect of UCB-bonuses. Specifically, we choose *Mnist-maze* and two Atari games *RoadRunner* and *Mspacman* to analyze.

E.1. MNIST-maze

Figure 6 illustrates the UCB-bonus in four randomly generated mazes. The mazes in Figure 6(a) and 6(b) have a wall-density of 40%. The mazes in Figure 6(c) and 6(d) have a wall-density of 50%. The left of each figure shows the map of maze, where the black blocks represent the walls. We omit the MNIST representation of states in the illustrations for simplification. A trained OB2I agent starts at the upper-left, then takes actions to achieve the goal at bottom-right. The UCB-bonuses of state-action pairs along the agent's visitation trajectory are computed and illustrated on the right of each figure. The value is normalized to $0 \sim 1$ for visualization. We show the maximal value if the agent appears several times in the same location.

The positions with UCB-bonuses that higher than 0 draw the path of the agent. The path is usually winding and includes positions beyond the shortest path because the state transition has stochasticity. The state-action pairs with high UCB-bonuses are typically the bottleneck positions in the path. In maze 6(a), the agent slips from the right path in position $(4,7)$ to $(4,9)$. The state-action in position $(4,8)$ produces high bonus to guide the agent back to the right path. In maze 6(b), the bottleneck state in $(3,2)$ has high bonus to avoid the agent from entering into the wrong side of the fork. The other two mazes also have bottleneck positions, like $(3,3)$ in maze 6(c) and $(7,6)$ in maze 6(d). Visiting these important locations is crucial to reaching the goal. We remark that the UCB-bonus of OB2I encourages the agent to walk through these bottleneck positions correctly.

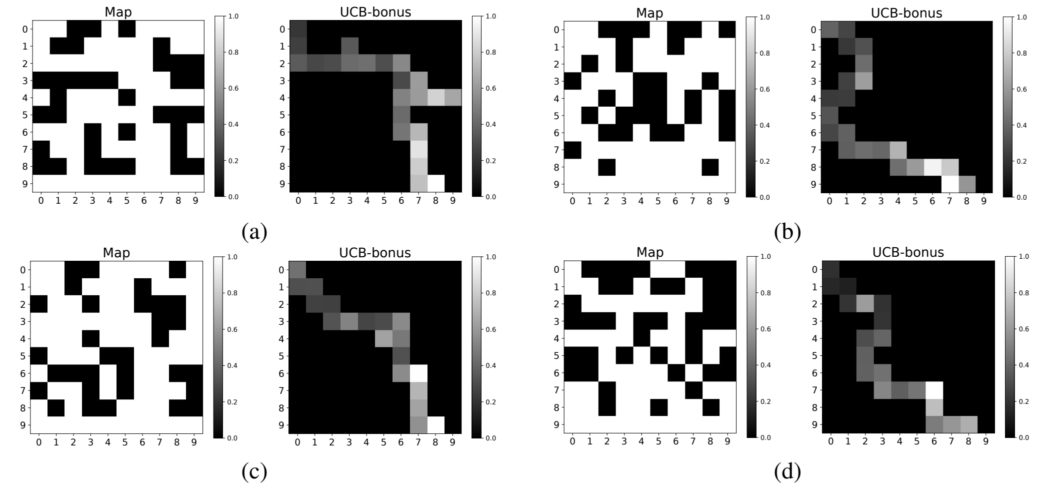


Figure 6. Visualization of UCB-bonus in Mnist-maze

E.2. RoadRunner

In RoadRunner, the agent is chased by Wile E. Coyote and run endlessly to the left to escape. Picking the bird seeds on the street takes 100 points. Inducing Wile E. Coyote to be run over by a car takes 1000 points and also make the agent get rid of the danger of being chased up. In this task, the performance of OB2I is 90% higher than that of BEBU. To illustrate how OB2I works, we use an OB2I agent to play this game for an episode and records the UCB bonus in all 1152 steps. Figure 7 shows the UCB bonus and the corresponding frames in 16 chosen spikes.

We find almost all spikes of UCB-bonus correspond to avoiding trucks and using trucks to get rid of Wile E. Coyote's chase (spike 2-14). The uncertainty is high with the emergence of truck because such a scenario rarely occurs. More importantly, utilizing the truck to get rid of Wile E. Coyote's chase has more uncertainty because the agent may get hit by the truck and lose its life. The UCB-bonus encourages the agent to learn skills that use the truck to gain advantages over the chaser and, hence, obtaining high scores. In addition, the agent eats bird seeds in spike 1. In spikes 15 and 16, the agent comes to a novel round.

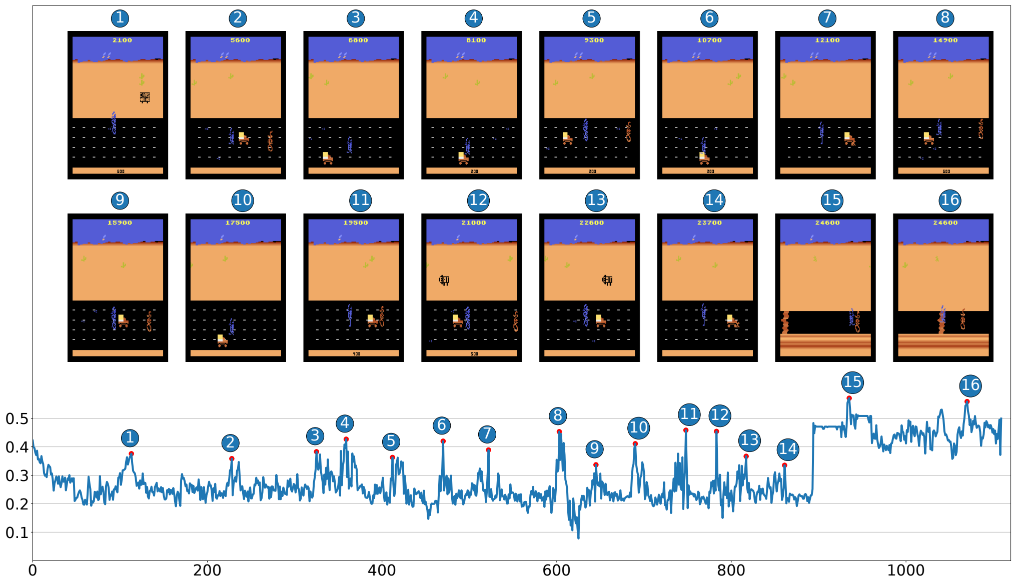


Figure 7. Visualization of the UCB-bonus in RoadRunner. We further record frames after each spike, and the video is available at https://www.dropbox.com/sh/6ffgl9v53kkldau/AABzADhD9TW-9gjMYiJI-4jYa?dl=0

E.3. MsPacman

In MsPacman, the agent earns points by avoiding monsters and eating pellets. Eating an energizer causes the monsters to turn blue, allowing them to be eaten for extra points. We use a trained OB2I agent to interact with the environment for an episode. Figure 8 shows the UCB bonus in all 708 steps. We choose 16 spikes to visualize the frames. The spikes of exploration bonuses correspond to meaningful events for the agent to get rewards: starting a new scenario (1,2,9,10), changing direction (3,4,13,14,16), eating energizer (5,11), eating monsters (7,8,12), and entering the corner (6,15). These state-action pairs with high UCB-bonuses make the agent explore the environment efficiently.

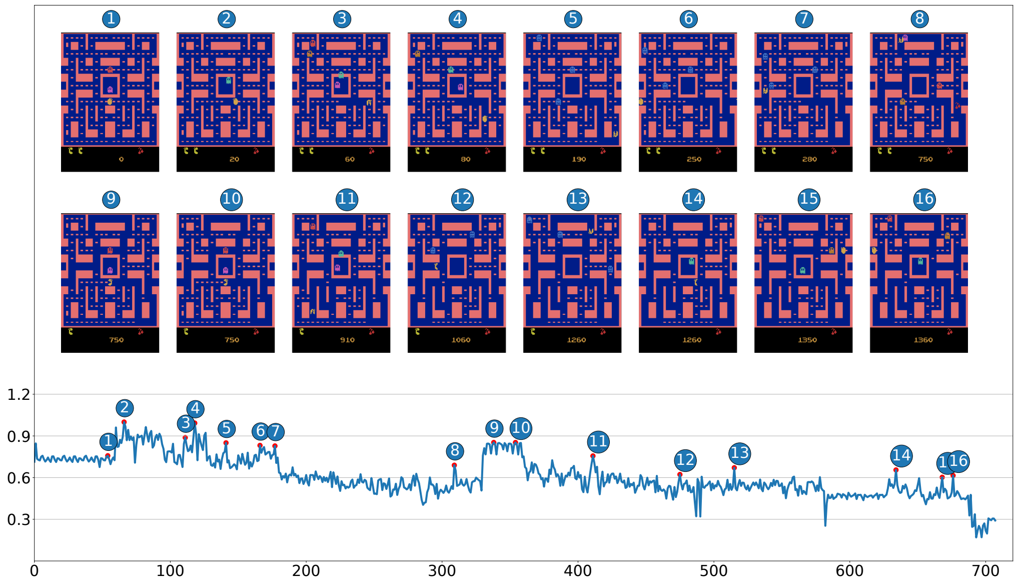


Figure 8. Visualization of the UCB-bonus in MsPacman. We further record frames after each spike, and the video is available at https://www.dropbox.com/sh/6ffgl9v53kkldau/AABzADhD9TW-9gjMYiJI-4jYa?dl=0

F. Raw Scores of all 49 Atari Games

Table 5. Raw scores for Atari games. Bold scores signify the best score out of all methods.

	Random	Human	BEBU	BEBU-UCB	BEBU-IDS	OB2I
Alien	227.8	6,875.0	**1,118.0**	811.1	857.9	916.9
Amidar	5.8	1676.0	81.7	**166.4**	148.1	94.0
Assault	222.4	1,496.0	1,377.0	**3,574.5**	2,441.8	2,996.2
Asterix	210.0	8,503.0	2,315.0	2,709.3	2,433.9	**2,719.0**
Asteroids	719.1	13,157.0	962.8	**1,025.0**	868.8	959.9
Atlantis	12,850.0	29,028.0	3,020,500.0	3,191,600.0	3,144,440.0	**3,146,300.0**
Bank Heist	14.2	734.4	331.8	277.0	361.6	**378.6**
Battle Zone	2,360.0	37,800.0	5,446.4	**16,348.8**	10,520.0	13,454.5
BeamRider	363.9	5,775.0	2,930.0	3,208.3	3,391.0	**3,736.7**
Bowling	23.1	154.8	29.9	30.7	**40.2**	30.0
Boxing	0.1	4.3	72.4	68.3	69.8	**75.1**
Breakout	1.7	31.8	**473.2**	382.3	412.7	423.1
Centipede	2,090.9	11,963.0	2,547.2	2,377.9	**3,328.4**	2,661.8
Chopper Command	811.0	9,882.0	930.6	1,013.4	1,100.0	**1,100.3**
Crazy Climber	10,780.5	35,411.0	49,735.7	39,187.5	42,242.9	**53,346.7**
Demon Attack	152.1	3,401.0	6,506.3	6,840.4	**7,080.0**	6,794.6
Double Dunk	-18.6	-15.5	-18.9	**-16.5**	-17.0	-18.2
Enduro	0.0	309.6	504.1	697.8	513.6	**719.0**
Fishing Derby	-91.7	5.5	-56.7	-83.8	**-53.3**	-60.1
Freeway	0.0	29.6	21.5	21.6	21.3	**32.1**
Frostbite	65.2	4,335.0	393.4	470.4	466.2	**1,277.3**
Gopher	257.6	2,321.0	4,842.6	**7,211.8**	7,171.5	6,359.5
Gravitar	173.0	2,672.0	256.1	321.0	283.3	**393.6**
H.E.R.O	1,027.0	25,763.0	2,951.4	2,905.0	3,059.4	**3,302.5**
Ice Hockey	-11.2	0.9	-5.4	-6.5	-4.6	**-4.2**
Jamesbond	29.0	406.7	**650.0**	360.3	302.1	434.3
Kangaroo	52.0	3,035.0	3624.2	2,711.1	**4,448.0**	2,387.0
Krull	1,598.0	2,395.0	15,716.7	11,499.0	10,818.0	**45,388.8**
Kung-Fu Master	258.5	22,736.0	56.0	20,738.9	**26,909.7**	16,272.2
Montezuma's Revenge	0.0	4,376.0	0.0	0.0	0.0	**0.0**
Ms. Pacman	307.3	15,693.0	1,723.8	1,706.8	1,615.5	**1,794.9**
Name This Game	2,292.3	4,076.0	8,275.3	6,573.9	**8,925.0**	8,576.8
Pong	-20.7	9.3	18.1	18.5	17.2	**18.7**
Private Eye	24.9	69,571.0	1,185.8	1,925.2	1,897.1	1,174.1
Q*Bert	163.9	13,455.0	3,588.4	3,783.2	3,696.0	**4,275.0**
River Raid	1,338.5	13,513.0	3,127.5	**3,617.7**	3,169.1	2,926.5
Road Runner	11.5	7,845.0	11,483.0	20,990.7	17,281.4	**21,831.4**
Robotank	2.2	11.9	10.3	13.3	10.7	**13.5**
Seaquest	68.4	20,182.0	447.0	**492.3**	332.4	332.1
Space Invaders	148.0	1,652.0	814.4	782.2	794.7	**904.9**
Star Gunner	664.0	10,250.0	1,467.2	1,201.5	1,158.9	**1,290.2**
Tennis	-23.8	-8.9	-1.0	-2.0	-1.0	**-1.0**
Time Pilot	3,568.0	5,925.0	2,622.1	3,321.2	1,950.6	**3,404.5**
Tutankham	11.4	167.6	167.0	151.0	80.5	**297.0**
Up and Down	533.4	9,082.0	**5,954.8**	4,530.2	4,619.7	5,100.8
Venture	0.0	1,188.0	42.9	3.4	**150.0**	16.1
Video Pinball	16,256.9	17,298.0	26,829.6	48,959.1	58,398.3	**80,607.0**
Wizard of Wor	563.5	4,757.0	810.8	**1,316.7**	578.2	480.7
Zaxxon	32.5	9,173.0	1,587.5	2,104.8	1,594.2	**2,842.0**

G. Performance Comparison

We use the relative scores as

$$\frac{\mathrm{Score}_{\mathrm{Agent}} - \mathrm{Score}_{\mathrm{Baseline}}}{\max\{\mathrm{Score}_{\mathrm{human}}, \mathrm{Score}_{\mathrm{baseline}}\} - \mathrm{Score}_{\mathrm{random}}}$$

to compare OB2I with baselines. The results of OB2I comparing with BEBU, BEBU-UCB, and BEBU-IDS is shown in Figure 9, Figure 10, and Figure 11, respectively.

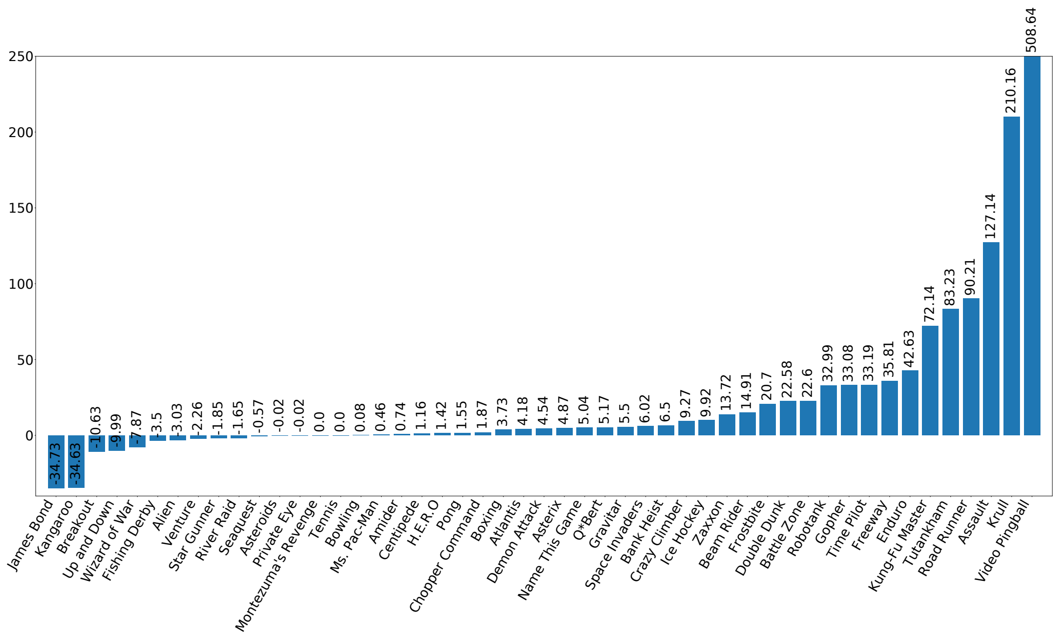


Figure 9. Relative score of OB2I compared to BEBU in percents (%).

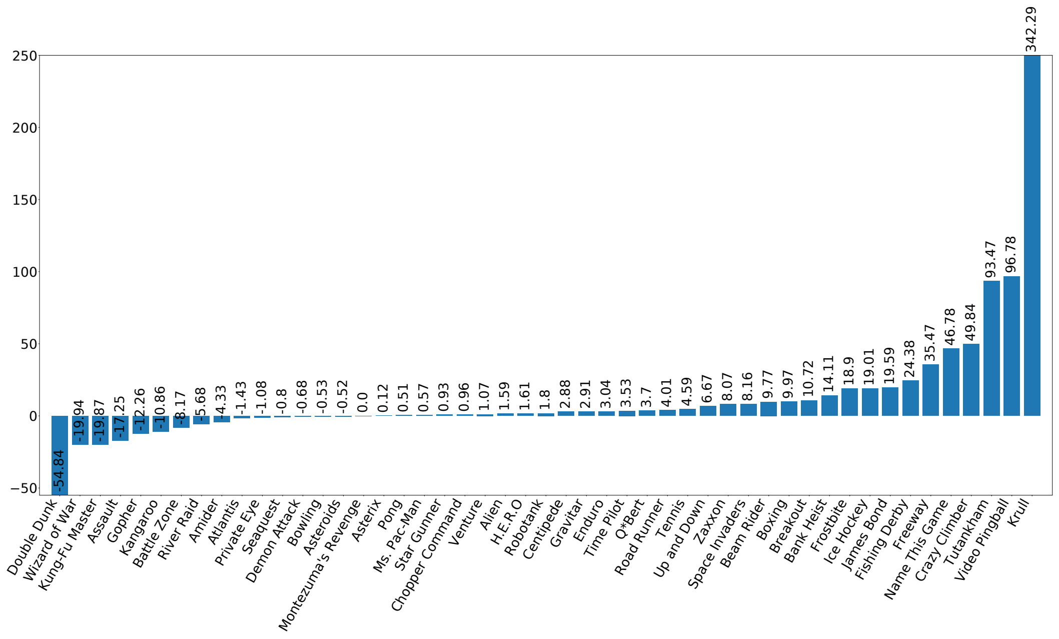


Figure 10. Relative score of OB2I compared to BEBU-UCB in percents (%).

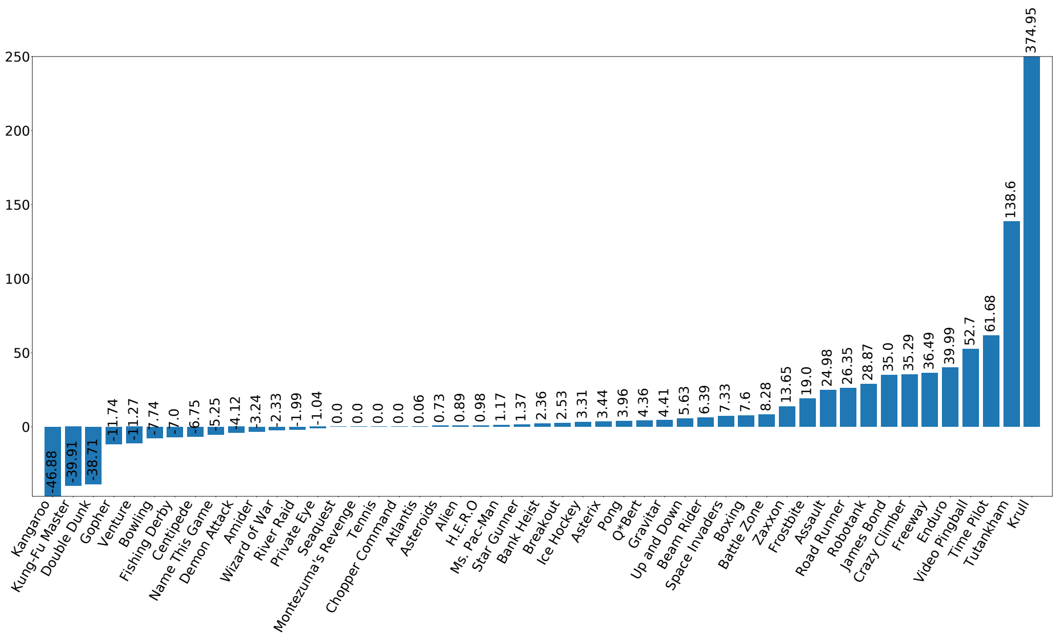


Figure 11. Relative score of OB2I compared to BEBU-IDS in percents (%).

H. Failure Analysis

Our method does not have a good performance on Montezuma's Revenge (see Table 6) because the epistemic uncertainty-based methods are not particularly tweaked for this domain. Meanwhile, IDS, NoisyNet and BEBU-based methods also fail on Montezuma's Revenge and score zero. Bootstrapped DQN achieves 100 points, which is also low and does not indicate successful learning in Montezuma's revenge. In contrast, the bonus-based methods achieve significantly higher scores on Montezuma's Revenge (e.g., RND achieves 8152 points). Nevertheless, according to Taiga et al. (2020) and Table 1, NoisyNet and IDS significantly outperform several strong bonus-based methods evaluated by the mean and median scores of 49 Atari games.

Table 6. Comparison of scores in *Montezuma's Revenge*.

Frames	200M				20M			
	DQN	BootDQN	NoisyNet	BootDQN-IDS	BEBU	BEBU-UCB	BEBU-IDS	OB2I
Scores	0	100	3	0	0	0	0	0

Moreover, we find that the length of episode (or horizon) matters since OB2I propagates uncertainty within an episode. We

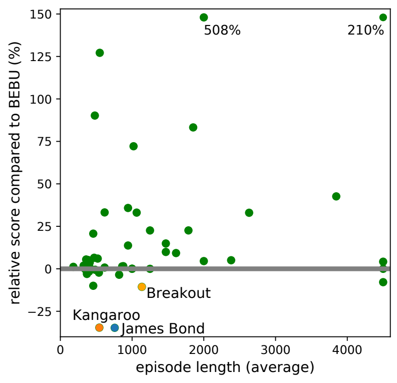


Figure 12. The relationship between Episode Length and Relative Scores

visualize the connection between horizon and performance in Fig. 12, where each point represents a game. We find that the games where OB2I is suboptimal typically have short horizons. In such games, propagating uncertainty does not bring much advantage, since it may be unnecessary.

Theoretically, BEBU (or BootDQN) instantiates Thompson sampling (with uninformative prior). As long as the prior is correctly specified, Thompson sampling attains the optimal Bayesian (average-case) regret. In contrast, OB2I instantiates optimism in the face of uncertainty (via UCB), which attains the optimal frequentist (worst-case) regret. In a few cases, OB2I may be overly conservative (with overly large UCB), since it aims to minimize the worst-case regret.

I. Algorithmic Comparison

Table 7. Algorithmic comparison of the closely related works

	Bonus or Posterior Variance	Update Method	Uncertainty Characterization
EBU (Lee et al., 2019)	-	backward update	-
Bootstrapped DQN (Osband et al., 2016)	bootstrapped	on-trajectory update	bootstrapped distribution
UBE (O'Donoghue et al., 2018)	closed form	on-trajectory update	posterior sampling
Bayesian-DQN (Azizzadenesheli et al., 2018)	closed form	on-trajectory update	posterior sampling
LSVI-UCB (Jin et al., 2020)	closed form	backward update	optimism
BEBU (base of our work)	bootstrapped	backward update	bootstrapped distribution
OB2I (ours)	bootstrapped	backward update	optimism