Abstract—Algorithms that tackle deep exploration – an important challenge in reinforcement learning – have relied on epistemic uncertainty representation through ensembles or other hypermodels, exploration bonuses, or visitation count distributions. An open question is whether deep exploration can be achieved by an incremental reinforcement learning algorithm that tracks a single point estimate, without additional complexity required to account for epistemic uncertainty. We answer this question in the affirmative. In particular, we develop Langevin DQN, a variation of DQN that differs only in perturbing parameter updates with Gaussian noise and demonstrate through a computational study that the presented algorithm achieves deep exploration.

I. INTRODUCTION

In reinforcement learning (RL), intelligent exploration relies on decisions that are driven not only by expectations but also epistemic uncertainty. Thompson Sampling (TS), for example, is a popular exploration scheme that makes decisions based on a posterior distribution over models [1], [2]. In its basic form, to generate decision, TS samples a model from the posterior and then selects an action that optimizes the sampled model.

Generating exact posterior samples is computationally feasible only for very simple environments, like tabular MDPs with Dirichlet priors over transition probability vectors [3]. Scaling TS to complex domains calls for approximations [2]. To serve this need, [4] developed randomized least-squares value iteration (RLSVI), which aims to approximate sample from the posterior over the optimal value function without explicitly representing the distribution. This algorithm randomly perturbs a prior and an accumulated dataset and fits, to this perturbed prior and data, a point estimate of the value function. The randomness induced by perturbations induces an intelligent form of exploration in actions subsequently selected to maximize the resulting value estimates.

Though RLSVI avoids maintaining an explicit posterior distribution, for each episode of operation, the algorithm produces a new point estimate based on an independently perturbed data set and prior. This requires intensive computations which do not leverage previously computed point estimates. Ensemble sampling [4], [5], [6], [7] can approximate the performance of RLSVI via maintaining a set of point estimates, each updated incrementally as data accumulates. However, maintaining an ensemble of complex models is itself computationally burdensome. Moreover, for a good approximation of the posterior distribution, the ensemble size needs to grow drastically with the complexity of the posterior distribution [8].

As an alternative to maintaining an ensemble of models, one can instead learn a hypermodel, which can be used to generate approximate posterior samples, as discussed in [8], [9]. This is a promising approach, but it requires a representation that can be much more complex than a point estimate of the value function.

An important feature of RLSVI and incremental variants that leverage ensembles or more general hypermodels is that they exhibit intelligence in the form of deep exploration. In particular, actions are taken to resolve epistemic uncertainty not only based on immediate consequences but also on what will be observed over the subsequent time periods. In this paper, we seek to understand whether a reinforcement learning algorithm that evolves a single point estimate of the optimal value function through incremental training can achieve deep exploration. We propose an algorithm – Langevin DQN – which we believe to serve this purpose. This algorithm synthesizes DQN [10] and stochastic gradient Langevin iterations [11].

On the surface, Langevin DQN may appear to share motivation and spirit with the “noisy networks” version of DQN [12], which randomly perturbs neural network weights both when computing state-action and target values for training and in selecting actions. But there are critical differences in where and how the noise is being added, as is evidenced by the fact that noisy networks do not achieve deep exploration [13].

Since the introduction of stochastic gradient Langevin dynamics (SGLD) [11], it has been widely studied both theoretically and empirically. However, its application in sequential decision problems is at a nascent stage. [14] has studied the use of SGLD in a bandit setting with independent arms and presented optimal regret guarantees with only a constant computation cost per iteration. In contrast, our work, Langevin DQN can be applied to reinforcement learning problems and has the ability to perform deep exploration using a fixed number of update steps per episode. Through our work we demonstrate that it is possible to achieve deep exploration with a single point estimate, which is updated incrementally, without additional complexity to account for epistemic uncertainty. Moreover, Langevin DQN differs from DQN only by a Gaussian noise term in the update step.

Since the initial version of this paper was submitted online, [20] has built upon this work, further underscoring its significance. The main distinction between Langevin DQN and the algorithm proposed in [20] is the inclusion of a temperature term, which controls the level of noise introduced at each

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University

This work was done at Stanford University
gradient step.

We provide some background on Langevin Monte Carlo and episodic reinforcement learning problem in Section II and III, respectively. We introduce our proposed Langevin DQN algorithm in Section IV and show its efficacy to perform deep exploration through computational results on an environment known as deep sea in V. Finally, we conclude in Section VI.

II. LANGEVIN SGD

Langevin Monte Carlo is a Markov chain Monte-Carlo (MCMC) algorithm that can generate samples approximately from a desired probability distribution. The algorithm represents a discrete-time approximation to the Langevin diffusion process, which takes the form

$$d\theta_t = \nabla_{\theta} \log p(\theta) dt + \sqrt{2}\sigma dW_t,$$

where $\theta_t$ takes values in $\mathbb{R}^d$, $p$ is the probability density of interest, and $W_t$ is a standard Brownian motion in $\mathbb{R}^d$. Under suitable technical conditions, it can be shown that $p$ is the unique invariant distribution of the Langevin diffusion process (see e.g., Proposition 6.1 in [15]).

A common Langevin Monte Carlo algorithm arises from Euler-Maruyama discretization:

$$\Delta \theta_k = \epsilon_k \nabla_{\theta} \log p(\theta) + \sqrt{2\epsilon_k} z_k,$$

where $\Delta \theta_k = \theta_k - \theta_{k-1}$, $\epsilon_k \in \mathbb{R}^+$ is a step size, $z_k$ is independently sampled from the $d$-dimensional standard normal distribution, and $\theta_t$ represents an approximation of $\theta_t$ with $t = \sum_{j=1}^k \epsilon_j$. The hope is that, under technical regularity conditions and with a suitable step size sequence, for sufficiently large $k$, the marginal distribution of $\theta_k$ is close to $p$ (for example, non-asymptotic convergence rates with respect to the $1$-Wasserstein distance are provided in [16]).

Before we develop Langevin DQN, it is helpful to first discuss a simpler but closely related application of Langevin Monte Carlo, which we will refer to as Langevin SGD. In particular, consider a supervised learning problem, where given a set $D = \{(x_i, y_i)\}_{i=1}^T$ of data pairs, each in $\mathcal{X} \times \mathcal{Y}$, the goal is to fit a function $f_\theta : \mathcal{X} \to \mathbb{R}$, which is parameterized by $\theta \in \mathbb{R}^d$. If we assume that $y_i = f_\theta(x_i) + \epsilon_i$ for some $\theta^* \in \mathbb{R}^d$ and iid $\epsilon_i \sim N(0, \sigma_\epsilon^2)$, then the negative log-likelihood function is given by

$$\mathcal{L}(\theta, D) = \frac{1}{2T} \sum_{i=1}^T (f_\theta(x_i) - y_i)^2.$$

Further, letting $\psi$ denote the negative log prior density of $\theta^*$, the posterior log-density of $\theta^*$ is given by

$$\log p(\theta|D) = -\mathcal{L}(\theta, D) - \psi(\theta).$$

Specialized to our supervised learning context, the Langevin Monte Carlo algorithm (1) becomes

$$\Delta \theta_k = -\epsilon_k \nabla_{\theta} \log p(\theta_{k-1}) \mathcal{L}(\theta, D) + \psi(\theta) + \sqrt{2\epsilon_k} z_k.$$

This iteration is like gradient descent, but with an extra term that injects noise into the parameter update. Let us refer to this as Langevin gradient descent. While gradient descent aims to converge on a MAP estimate of $\theta^*$, Langevin gradient descent generates a stochastic sequence of parameter vectors $\theta_0, \theta_1, \theta_2, \ldots$ such that, under suitable technical conditions and for sufficiently large $k$, the marginal distribution of $\theta_k$ offers a close approximation to the posterior distribution of $\theta^*$.

When the dataset $D$ is large, processing the entire dataset in each iteration may be computationally onerous. We can replace each Langevin gradient descent step (2) with one computed from a minibatch:

$$\Delta \theta_k = -\epsilon_k \nabla_{\theta} \log p(\theta_{k-1}) \left(\frac{\mathcal{L}(\theta, \tilde{D})}{|\tilde{D}|} + \psi(\theta)\right) + \sqrt{2\epsilon_k} z_k,$$

where $\tilde{D} \subseteq D$ is a fixed size minibatch, for which data points are sampled uniformly at random from $D$ with replacement.

We will refer to this algorithm as Langevin SGD. The authors of [11] argue that, for an appropriately chosen step size sequence, the marginal distribution of $\theta_k$ converges to the posterior distribution in a sense that can be made precise.

To introduce notation that simplifies development of Langevin DQN, let $\alpha_k = \epsilon_k |D|$ so that we can rewrite Langevin SGD (3) as

$$\Delta \theta_k = -\alpha_k \nabla_{\theta} \log p(\theta_{k-1}) \left(\frac{\mathcal{L}(\theta, \tilde{D})}{|\tilde{D}|} + \psi(\theta)\right) + \sqrt{2\alpha_k} z_k.$$

III. REINFORCEMENT LEARNING

Langevin DQN synthesizes Langevin SGD and DQN to address RL. In this paper, we restrict attention to an episodic RL setting in which an agent interacts with an unknown environment over episodes, aiming to maximize accumulated rewards. We model the environment as a Markov Decision Process (MDP) characterized by a quintuple $M = (S, A, R, \mathcal{P}, \rho)$. Here, $S$ is a finite state space, $A$ is a finite action space, $R$ is a reward model, $\mathcal{P}$ is a transition model, and $\rho$ is an initial state distribution. Given a state $s \in S$, action $a \in A$, and next state $s' \in S$, the distribution of the possible rewards the agent can experience when transitioning from $s$ to $s'$ upon taking action $a$. Similarly, $\mathcal{P}_{s,a}(s')$ is the conditional probability that the state transitions to $s'$ from state $s$ upon taking action $a$, while $1 - \sum_{s' \in S} \mathcal{P}_{s,a}(s')$ is the probability that the episode terminates. We denote the sequence of observations made by the agent in episode $l$ by $(s_0^l, a_0^l, r_0^l, s_1^l, \ldots, s_{\tau_l-1}^l, a_{\tau_l-1}^l, r_{\tau_l-1}^l, s_{\tau_l}^l)$, where $s_0^l$ is the state of the environment observed by the agent at time $h$ and episode $l$, $r_h^l$ is the reward observed by the agent on taking an action $a_h^l$, and $s_{\tau_l}^l$ denotes the time at which episode terminates at state $s_{\tau_l}^l$.

A (stationary stochastic) policy $\pi : S \times A \to [0, 1]$ assigns a probability mass function over actions to each state. In particular, we denote by $\pi(a|s)$ as the probability that the policy $\pi$ selects action $a$ when in state $s$. Let $\Pi$ denote the set of all policies. Denote by $P_{\pi}$ the transition probability matrix
under policy \( \pi \); i.e., \( P\pi,s,s' = \sum_{a \in A} \pi(a|s)P_{s,a}(s') \). We make the following assumption which ensures that episodes terminate.

**Assumption 1. (finite episodes)**
For all \( \pi \in \Pi \), \( \lim_{h \to \infty} P^h_\pi = 0 \).

Under Assumption 1, the state-action value function \( Q^\pi : S \times A \mapsto \mathbb{R} \) of each policy \( \pi \) is finite, where \( Q^\pi \) is defined by

\[
Q^\pi(s, a) = \mathbb{E}_{M, \pi} \left[ \sum_{h=0}^{\tau-1} \gamma^h r_h | s_0 = s, a_0 = a \right],
\]

where the subscripts of the expectation indicate that state transitions and rewards are generated by MDP \( M \) with actions sampled by policy \( \pi \). We denote the optimal value function by \( Q^*(s, a) = \max_{\pi} Q^\pi(s, a) \).

Similar to DQN, Langevin DQN, is a value learning algorithm. In value learning, an agent updates the parameter vector \( \theta \) of a parameterized value function \( Q_\theta \) as data accumulates, with an aim of estimating the optimal value function \( Q^* \). At each time, an action is selected based on the current value function estimate.

A template for value learning with greedy actions is presented in Algorithm 1. The algorithm begins with an empty data buffer and then iterates over episodes. In each episode, the agent updates the parameter vector based on buffered data and then applies \( \epsilon \)-greedy actions through termination. The parameter update could be carried out, for example, by the DQN or Langevin DQN algorithm. These algorithms make use of a target model with parameter vector \( \theta^\text{target} \) in their updates to maintain stability [19]. \( \theta^\text{target} \) is a snapshot of \( \theta \) and updated periodically (say, every \( M \) update steps). Observed actions, state transitions, and rewards are added to the buffer. In particular, each item in the buffer is a tuple \((s, a, r, s')\), where \( s \) is a state, \( r, s' \) are the reward and the next state observed by the agent on taking an action \( a \). The most basic version of a buffer would simply accumulate all observations. But in practical implementations, one might design a discipline that limits the buffer size by ejecting old or randomly selected data samples.

### IV. LANGEVIN DQN

In this section we present Langevin DQN, an algorithm inspired from Langevin SGD for sequential decision problems, particularly for reinforcement learning problems.

The DQN algorithm updates the parameter vector \( \theta \) by taking SGD-like steps that aim to reduce TD loss possibly summed with a regularization penalty. These SGD-like steps are similar to Algorithm 2, but without the Gaussian noise term. The Langevin DQN update, presented in Algorithm 2 instead uses Langevin-SGD-like steps.

**Algorithm 2 LangevinDQN_update**

**Input:**
- \( L(\theta = \cdot, \theta^\text{target} = \cdot, D = \cdot) \) TD loss
- \( \psi(\theta = \cdot) \) regularizer
- \( \alpha \) learning rate
- \( D \) data buffer
- \( B \) mini-batch size
- \( \theta^- \) target parameters
- \( \theta^+ \) new parameters

**Return:**
- \( \Delta \theta \) update

1. \( \hat{D} \leftarrow \text{sample minibatch}(D, B) \)
2. \( z \leftarrow \text{sample normal}(0, I_{\dim(\theta^-)}) \)
3. compute increment

\[
\Delta \theta \leftarrow -\alpha \nabla_{\theta = \theta^-} \left( \frac{1}{|D|} L(\theta, \theta^\text{target}, \hat{D}) + \frac{1}{|D|} \psi(\theta) \right) + \sqrt{\frac{2\alpha}{|D|}} z
\]

4. return \( \theta^- + \Delta \theta \)

For a buffer \( D \) with elements of form \((s, a, r, s')\), the TD loss used in algorithm 2 can be written as

\[
L(\theta, \theta^\text{target}, (s, a, r, s')) = \sum_{(s, a, r, s') \in D} L(\theta, \theta^\text{target}, (s, a, r, s')), \]

where

\[
L(\theta, \theta^\text{target}, (s, a, r, s')) = \frac{1}{2\sigma^2} (r + \max_{s'' \in A} Q^\text{target}(s'', a') - Q_\theta(s, a))^2, \]

with an understanding that \( Q_\theta(s', a') = 0 \) if the transition from \( s \) terminates the episode i.e., \( s' \) is a terminal state. The term that is squared in this expression is the so-called temporal difference. Note that bandits are special case with each episode ends after a single action selection, i.e., \( s' \) is terminal state for all transitions.

Langevin DQN is very similar to DQN, requiring very minor changes. The primary difference lies in the fact that, while Langevin DQN updates parameters based on an expression very similar to DQN, the expression used in Langevin DQN includes an additional random perturbation term. The only other difference is that, while \( \epsilon \)-greedy exploration is
typically used with DQN, Langevin DQN needs no additional exploration scheme and thus it suffices to apply greedy actions, i.e., $\epsilon = 0$. We would like to note that it is possible to design reinforcement learning algorithms that operate in continuing rather than episodic environments using an update rule similar to Langevin DQN, though we do not develop that idea here.

V. COMPUTATIONAL EXPERIMENTS

To assess whether Langevin DQN achieves deep exploration, we apply it to the deep sea environment \[4\], \[17\]. The deep sea environment with depth $N$ can be seen as an $N \times N$ two-dimensional grid in which the agent starts in the upper-left corner and should ideally reach a treasure chest at the lower-right corner.

At each time, the agent applies an action from $A = \{0, 1\}$ to move to the left or to the right cell in the next row. However, the mapping between $\{0, 1\}$ to (left, right) is unknown to the agent and varies across states. For simplicity, we assume that there is a treasure in the chest which is revealed to the agent when it takes the right action on reaching the chest, resulting in a reward of 1. In addition, there is a small penalty of $0.01/N$ whenever the agent takes the right action.

We compare the performance of Langevin DQN against traditional DQN \[10\], which uses $\epsilon$-greedy exploration. Both share Algorithm 1 as a template. Recall that Langevin DQN agent acts greedily according to its value function estimates which is obtained by applying iterations of Algorithm 2 in the update step in Algorithm 1.

In our experiments, we use 2-hidden layer neural networks to approximate value functions. Langevin DQN and DQN each use a single neural network mapping state-action pairs to value. We use common hyperparameters, including neural network architecture, buffer discipline, buffer size, batch size, number of updates performed between episodes, and target network update frequency, for both algorithms to ensure that update routines impose similar computational requirements. The exploration intensity $\epsilon$ was set to zero for Langevin DQN but tuned to optimized performance for DQN.

We used the Adam optimizer \[18\] for updating parameters of DQN. Since Langevin DQN has an additional Gaussian noise term in the update equation, Adam could not be used in a straightforward manner. Hence, we implemented preconditioned Langevin SGD, using the Adam optimizer as the backbone. More details about the optimizer are provided in Appendix A.

In order to assess and compare performance, we define a notion of learning time. In particular, we take learning time to be the first episode at which cumulative regret is less than 0.8 times the number of episodes. In other words,

$$\text{Learning time} = \min\{l : \text{Regret}(l) < 0.8 \cdot l\}.$$  

The cumulative regret until the episode $l$ is defined as the difference between the maximal reward that the agent could have accumulated and the reward that the agent has accumulated over the $l$ episodes. In the case of the deep sea environment, the maximum possible reward an agent can obtain in an episode is 0.99, and the agent observes a positive reward only when it takes the optimal actions throughout the episode and reaches the treasure chest. In all other cases, the agent observes a non-positive reward and experiences regret of at least 0.99. The learning time can be thought of as the first episode by which the agent has behaved optimally for at least two tenths of past episodes, and we say that the agent has ‘learned’ or ‘solved’ the environment at that episode. The median learning time for an $\epsilon$-greedy strategy is lower bounded by $2^N$ \[4\].

We carried out experiments with deep sea environments with depths varying from 10 to 20 in steps of 2. We experimented with different values of $\epsilon$ for DQN and found that starting with a value of 0.5 and decaying it optimizes the performance. For the Langevin DQN agent, we scale $\sigma^2_{\text{wh}}$ with deep sea depths. For each set of hyperparameters, we ran 5 trials, each with a different random seed. In each trial, an agent operates over 5,000 episodes. Note that as the depth of the deep sea increases, more data accumulates per episode. Since we used a fixed mini-batch size, we scaled the number of update steps by the depth of the deep sea environment in both Langevin DQN and DQN algorithms.

Figure 1 shows the performance of DQN and Langevin DQN for deep sea environments of depths 10 to 20. The horizontal axis shows the depth of the deep sea environment. The vertical axis shows the fraction of seeds in which an agent has been able to solve a deep sea environment within 5000 episodes for a specific number of update steps performed per an environment step.

Figure 1. Performance of DQN and Langevin DQN for deep sea environments of depths 10 to 20. The horizontal axis shows the depth of the deep sea environment. The vertical axis shows the fraction of seeds in which an agent has been able to solve a deep sea environment within 5000 episodes for a specific number of update steps performed per an environment step. One update step per environment step is equal to $N$ number of update steps per an episode for a deep sea environment of depth $N$. Note that same hyperparameter settings are used across all the random seeds. From the figure, we can see that DQN is incapable of performing deep exploration irrespective of the number of update steps, and outperformed significantly by Langevin DQN. We can also see that as we increase the number of update steps, the performance of Langevin DQN increases. To understand this better, we plot the median learning time of Langevin DQN agent on a deep sea environment of size 10 vs the number of update steps per environment step in Figure 2. From the figure, the learning time (and the number of samples required to learn) for the Langevin DQN algorithm decreases drastically with an increase in the number of
update steps and goes to as low as 90 episodes.

Fig. 2. Median learning time of Langevin DQN on a deep sea environment of depth 10 for varying number of update steps.

To better understand how Langevin DQN facilitates deep exploration, we examine the uncertainty in Q-values generated by Langevin DQN across different states in the deep sea with a depth of 10, in Figure 3. At the start of each episode, the agent performs 250 update steps per environment step. The uncertainty at each state is calculated as sum of the standard deviation of Q-values across all actions at that state, where the standard deviation of a Q-values at a state-action pair is calculated empirically based on the Q-values of the past 10 episodes. We have clipped the standard deviation values at 50 for better visualization. Note that a state is uncertain if the Q-values of any action at that state are uncertain, and in a deep sea environment, the agent cannot reach the states in the upper right triangle of the grid.

From Figure 3, we can see that at episode 10, the agent is uncertain about almost the entire state space, except for a single state. By episode 40, the agent has observed few states at the bottom left corner, which reduces uncertainty in Q-values at those states; however, since some trajectories from the top left states lead to bottom right states which are unexplored, the Q-values at the top left states are still uncertain even though they have been visited several times. As the agent interacts with the environment, it explores, resolves its uncertainty, and finally learns an optimal policy.

In addition to the experiments on deep sea environment, we have also conducted experiments on all the bsuite environments [17]. In these experiments, we used a fixed hyperparameter settings for all the bsuite environments. Figure 4 shows the performance of Langevin DQN and DQN across different categories of bsuite environments. We can see that Langevin DQN performs as well as DQN, if not better, on all categories of bsuite environments except for scale environments. In the scale environments, the maximum reward of an environment is scaled with a multiplier ranging from $10^{-3}$ to $10^{3}$. This test for the robustness of an algorithm against reward scaling, using a single hyperparameter settings. This leads to lower performance in Langevin DQN, as we are using the same value of $\sigma^2_w$ for all reward scales.

Fig. 3. Uncertainty of Langevin DQN at different states of a deep sea environment of depth 10 when run for 250 update steps per environment step. We define uncertainty of a state as the sum of standard deviations of Q-values across all actions at that state.

From these experiments we demonstrated that the Langevin DQN algorithm is capable of deep exploration, while tracking a single point estimate which is updated incrementally. Note that the main advantage of the Langevin DQN is its simplicity. In spite of being very similar to DQN, the Langevin DQN algorithm is capable of deep exploration.

VI. CONCLUSION

In this paper, we introduced Langevin DQN, a slight variation of DQN that achieves deep exploration using a single point estimate of the value function. We showed this through computational experiments and provided insights into how Langevin DQN accomplishes this. Langevin DQN is the first algorithm capable of deep exploration with a single point estimate with no additional complexity to account for epistemic uncertainty.
We thank Vivek Borkar, Xiuyuan Lu, Morteza Ibrahimi and Ian Osband for helpful discussions and pointing out errors and ambiguities in earlier drafts. This research was supported by the Army Research Office (ARO) under award W911NF2010055.

REFERENCES

[1] Thompson, William R. "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples." Biometrika 25, no. 3-4 (1933): 285-294.
[2] Russo, Daniel J., Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. "A tutorial on thompson sampling." Foundations and Trends® in Machine Learning 11, no. 1 (2018): 1-96.
[3] Osband, Ian, Daniel Russo, and Benjamin Van Roy. "(More) efficient reinforcement learning via posterior sampling." Advances in Neural Information Processing Systems 26 (2013).
[4] Osband, Ian, Benjamin Van Roy, Daniel J. Russo, and Zheng Wen. "Deep exploration via randomized value functions." Journal of Machine Learning Research 20, no. 124 (2019): 1-62.
[5] Osband, Ian, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. "Deep exploration via bootstrapped DQN." Advances in neural information processing systems 29 (2016).
[6] Lu, Xiuyuan, and Benjamin Van Roy. "Ensemble sampling." Advances in neural information processing systems 30 (2017).
[7] Dwarkar, Vikranth, Zheng Wen, Ian Osband, Xiuyuan Lu, Seyed Mohammad Asghari, and Benjamin Van Roy. "Ensembles for uncertainty estimation: Benefits of prior functions and bootstrapping." arXiv preprint arXiv:2206.03653 (2022).
[8] Dwarkar, Vikranth, Xiuyuan Lu, Morteza Ibrahimi, Ian Osband, Zheng Wen, and Benjamin Van Roy. "Hypermodels for exploration." arXiv preprint arXiv:2006.07464 (2020).
[9] Dwarkar, Vikranth R., Benjamin Van Roy, and Morteza Ibrahimi. "Posterior sampling networks." In Reinforcement Learning and Decision Making Conference, pp. 366-370. 2019.
[10] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).
[11] Welling, Max, and Yee W. Teh. "Bayesian learning via stochastic gradient Langevin dynamics." In Proceedings of the 28th international conference on machine learning (ICML-11), pp. 681-688. 2011.
[12] Meire Fortunato and Mohammad Gheshlaghi Azar and Bilal Piot and Jacob Menick and Ian Osband and Doina Precup, Anima Anandkumar, and Kamyar Azizzadenesheli. "Provable ability exceeds another in view of the evidence of two samples." arXiv preprint arXiv:2015:529-533.
[13] Thompson, William R. "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples." Biometrika 25, no. 3-4 (1933): 285-294.

A. Experimental Setup

Code for Langevin DQN can be found at https://github.com/vik0/LangevinDQN. We implement most of our code in TensorFlow using the source code of Bsuite [17] at https://github.com/deepmind/bsuite.

B. Optimizer Details

We use a preconditioned version of Langevin SGD (5), using Adam optimizer as the backbone. The pseudo-code for updating a parameter θ using the Langevin SGD optimizer is provided in Algorithm 3.  \( \hat{g}^2 \) used in the algorithm is the element wise square of the stochastic gradient \( \hat{g} \).

Algorithm 3 Langevin_SGD_optimizer

Input: \( \alpha \) learning rate \( \beta_2 \) decay rate of 1st grad moments \( \beta_2 \) decay rate of 2nd grad moments \( \epsilon \) small constant for numerical stability \( k \) stochastic gradient of the loss \( \theta_{k-1} \) parameters at \( k-1 \)th iteration \( m_{k-1} \) 1st raw moment est at iter \( k-1 \) \( v_{k-1} \) 2nd raw moment est at iter \( k-1 \)

Return: \( \theta_k \) new parameters \( m_k, v_k \) new 1st, 2nd raw moment estimates

1: Sample a Gaussian random vector with same dimension as \( \theta; z \sim N(0, I) \)
2: Update:

\[
\alpha_k = \alpha \left( \frac{\sqrt{1 - \beta_2^k}}{1 - \beta_1^k} \right)
\]

\[
m_k = \beta_1 m_{k-1} + (1 - \beta_1) \hat{g}
\]

\[
v_k = \beta_2 v_{k-1} + (1 - \beta_2) \hat{g}^2
\]

\[
\theta_k = \theta_{k-1} - \alpha_k \frac{m_k}{\sqrt{v_k + \epsilon}} + \sqrt{\frac{2\alpha_k}{\sqrt{v_k + \epsilon}}} z
\]

3: return \( \theta_k, m_k, v_k \), and \( v_k \)

The update rule used by Langevin DQN in our experiments is

\[
\Delta \theta \leftarrow -\alpha \nabla_{\theta} \mathbb{E}_{\theta \sim \mathcal{D}} \left( \sum_{(s,a,r,s') \in \mathcal{D}} \left( r + \max_{a' \in \mathcal{A}} Q_{\text{target}}(s', a') - Q_{\theta}(s, a) \right)^2 \right) + \frac{\sigma^2}{|\mathcal{D}|} \mathbb{E}_{\theta \sim \mathcal{D}} \left( \psi(\theta) \right) + \sqrt{\frac{2\alpha \sigma^2}{|\mathcal{D}|}} \frac{z}{\sqrt{\epsilon^2 + \epsilon}}.
\]

It is worth noting that we tried both intra-episodic and inter-episodic updates and did not observe a significant difference in the performance. We present the results from inter-episodic updates in this paper.

We use \( \psi(\theta) = \lambda ||\theta||^2 \) in our experiments.