# Bayesian Ensembles for Exploration in Deep Q-Learning

Pascal R. van der Vaart
Delft University of Technology
Delft, Netherlands
p.r.vandervaart-1@tudelft.nl

Neil Yorke-Smith
Delft University of Technology
Delft, Netherlands
n.yorke-smith@tudelft.nl

Matthijs T. J. Spaan
Delft University of Technology
Delft, Netherlands
m.t.j.spaan@tudelft.nl

## ABSTRACT

Exploration in reinforcement learning remains a difficult challenge. In order to drive exploration, ensembles with randomized prior functions have recently been popularized to quantify uncertainty in the value model. However these ensembles have no theoretical motivation why they should resemble the actual posterior. In this work, we view training ensembles from the perspective of Sequential Monte Carlo, a Monte Carlo method that approximates a sequence of distributions with a set of particles, and propose an algorithm that exploits both the practical flexibility of ensembles and theory of the Bayesian paradigm. We incorporate this method into a standard DQN agent and experimentally show qualitatively good uncertainty quantification and improved exploration capabilities over a regular ensemble.

## KEYWORDS

Reinforcement Learning, Exploration, Bayesian, Uncertainty

## 1 INTRODUCTION

Reinforcement learning (RL) algorithms are still notoriously sample inefficient. One pressing reason is the difficulty of exploring an environment efficiently while assuming little prior knowledge [34]. A promising approach that is currently studied is to quantify uncertainty in the value models learned by the agent, and then either provide intrinsic reward, be optimistic, or use Thompson sampling to explore [1–4, 8, 15, 16, 22, 28, 29, 31]. However, quantifying uncertainty for deep neural networks is in itself a difficult task [19, 25].

Ensembles of neural networks have been shown to provide better predictive accuracy over a single model in supervised learning tasks [12, 21], as well as suitable methods for uncertainty quantification for exploration in reinforcement learning [14, 28, 29]. While ensembles with independent models of identical architecture tend to collapse to the same predictive model [18], there are several techniques developed to prevent this, such as adversarial learning [21], bootstrapping the data [29], and adding additive priors [28]. Further, some techniques such as Stein Variational Gradient Descent [9, 23] alleviate this issue by interpreting the ensemble as an approximation to the Bayesian posterior and training it as such. The method that we propose falls into this last category and aims to be closer to the posterior for more accurate uncertainty quantification, while retaining the flexibility of ensembles.

Bayesian neural networks can have desirable properties if the posterior can be inferred accurately. They have in theory optimal predictive accuracy given the correct likelihood and prior and also provide accurate uncertainty quantification. Unfortunately, exactly inferring the posterior is intractable already for some simple statistical models, and accurately approximating this posterior is very difficult for neural networks. Typically, posterior approximation methods fall into one of two categories: Markov Chain Monte Carlo (MCMC), and Variational Inference.

Recently both types of methods have been altered specifically for application to neural networks. Variational Inference can be scaled to large networks by picking simple model classes to tractably optimize within the class and has been applied to RL in the context of uncertainty quantification and exploration [15, 16, 32]. Due to the complex nature of Neural Networks however, it is unclear how the model class biases uncertainty quantification. On the other hand, MCMC is in theory unbiased and also shows strong results in large networks in practice [7, 17, 33]. However, for complex multimodal distributions, MCMC methods can struggle to find every mode [11]. This is an important drawback in deep learning, where the posterior distribution is likely very ill behaved, and especially in RL where under-approximation of the posterior complexity might lead to underestimating the uncertainty and therefore failure of exploration. Sequential Monte Carlo (SMC), which uses a set of particles to approximate the posterior, can be a remedy to these issues in non-deep learning applications [11].

In this work, we forego Variational Inference to avoid a decision in model class, and instead alleviate the issues in MCMC by using SMC. Noting the success of ensembles in deep learning, we unify ensembles and MCMC methods by using SMC algorithms to train an ensemble in a Bayesian manner, to benefit from both the practical effectiveness of ensembles and theoretical foundations of MCMC. Specifically, our contributions are as follows:

(1) We adapt existing SMC algorithms to a minibatch setting, and show empirically that they are feasible methods to train ensembles so that they serve as proper approximations to the Bayes posterior.
(2) As our main contribution, we introduce Sequential Monte Carlo DQN (SMC-DQN)[1], an RL algorithm which uses SMC to track a posterior over the Q-values in a theoretically sound manner, by sequentially updating its models with incoming data and correctly conditioning on target parameters. The agent uses Thompson sampling from the posterior distribution to drive exploration.
(3) We experimentally test our agent's exploration capabilities on several environments, observing significantly stronger performance over regular ensembles and results that are competitive with a strong baseline.

*Proc. of the Adaptive and Learning Agents Workshop (ALA 2024), Avalos, Milec, Müller, Wang, Yates (eds.), May 6-7, 2024, Auckland, New Zealand, https://ala2024.github.io/.* 2024.

[1] Code is available at https://github.com/Pascal314/BayesianEnsemblesAAMAS

## 2 BACKGROUND

In this section we introduce the necessary background for our contribution.

### 2.1 Markov Decision Processes

A Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ of a state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and discount factor $0 \leq \gamma < 1$. At each time step $t$, an agent observes the current state $s_t$, chooses an action $a_t \sim \pi(s_t)$ according to its policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, and receives reward $r_t = R(s_t, a_t)$. The goal of reinforcement learning is to find a policy $\pi$ that maximizes the discounted cumulative reward $\mathbb{E}_{T,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$. Of central importance is the Q-function

$$Q^\pi(s,a) = R(s,a) + \mathbb{E}_{T,\pi}\left[\sum_{t=1}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right],$$

denoting the expected discounted future reward if the agent executes action $a$ in state $s$ and then follows the policy $\pi$.

Since the transition function $T$ and reward function $R$ are assumed to be unknown to the agent, computing a strong policy requires exploration of the environment to learn which actions result in optimal return.

### 2.2 Bootstrapped DQN

A common strategy to find an optimal policy is Deep Q-learning (DQN) [26], where a parameterized neural network $Q_\theta$ is trained to minimize the temporal difference error

$$\operatorname*{argmin}_\theta \left[Q_\theta(s,a) - r - \gamma \max_{a'} Q_{\theta'}(s',a')\right]^2, \tag{1}$$

with $(s, a, r, s')$ sampled from the environment or a replay buffer, and $\theta' \leftarrow \theta$ is periodically updated for stability. The corresponding policy picks $\operatorname{argmax}_a Q_\theta(s,a)$ in every state. The agent can pick a random action instead with probability $\epsilon$, to perform exploratory actions. To improve exploration, the BootDQN algorithm [29] leverages the uncertainty estimation abilities of ensembles. By learning an ensemble of $Q$-networks

$$Q_{\theta_1}(s,a), \ldots, Q_{\theta_n}(s,a)$$

the agent achieves deep exploration through Thompson sampling, sampling uniformly $i \in \{1, \ldots, n\}$ and acting greedily with respect to the network $Q_{\theta_i}$ for a full episode by picking $\operatorname{argmax}_a Q_{\theta_i}(s,a)$ in every state $s$. To update its predictions, each network $Q_{\theta_i}$ is equipped with its own target network $Q_{\theta'_i}$, and gets updated with its own targets:

$$\theta_i \leftarrow \theta_i - \nabla_{\theta_i}\left[Q_{\theta_i}(s,a) - r - \max_{a'} Q_{\theta'_i}(s',a')\right]^2, \tag{2}$$

where $(s, a, r, s')$ are transitions sampled uniformly from a replay buffer.

Crucially, the ensemble $Q_{\theta_1}(s,a), \ldots, Q_{\theta_n}(s,a)$ has to stay diverse in under-explored states in order to keep exploration going. This can be achieved by bootstrapping the data for each ensemble member, or by using randomized prior functions. Randomized prior functions have been shown to be effective at keeping ensemble diversity [28].

A randomized prior function is a fixed function $Q_{\vartheta_i}(s,a)$ that is sampled independently for each ensemble member $Q_{\theta_i}(s,a)$, at the start of training. It remains unmodified during training and is added to the model outputs:

$$(Q_{\theta_i} + Q_{\vartheta_i})(s,a) = Q_{\theta_i}(s,a) + Q_{\vartheta_i}(s,a).$$

During action selection, the $Q$-values of ensemble member $i$ are given by $(Q_{\theta_i} + Q_{\vartheta_i})(s,a)$, and the BootDQN update (2) is modified to the following:

$$\theta_i \leftarrow \theta_i - \nabla_{\theta_i}\left[(Q_{\theta_i} + Q_{\vartheta_i})(s,a) - r - \max_{a'}(Q_{\theta'_i} + Q_{\vartheta_i})(s',a')\right]^2.$$

The fact that each ensemble member has a unique prior function causes unique generalization behaviour on unobserved data. This keeps the ensemble outputs diverse in under-explored states.

However, randomized prior functions lack theoretical motivation when considered as Bayesian priors for neural networks. In problems with well-defined likelihoods and priors, the Bayesian posterior can therefore be expected to outperform methods that rely on randomized prior functions.

### 2.3 Bayesian Neural networks

A Bayesian Neural Network (BNN) is any neural network $f_\theta$ parameterized by $\theta \in \Theta$, with some prior distribution $p(\theta)$ over $\Theta$. Given training data $(x_1, \ldots, x_n)$ and labels $(y_1, \ldots, y_n)$ i.i.d. from some likelihood $\mathcal{L}(y|f_\theta(x))$, the goal is to compute or sample from the posterior distribution over the parameter $\theta$:

$$\begin{aligned} p(\theta|\mathcal{D}) &= \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta)d\theta} \\ &= \frac{\prod_{i=1}^n \mathcal{L}(y_i|f_\theta(x_i))p(\theta)}{\int \prod_{i=1}^n \mathcal{L}(y_i|f_\theta(x_i))p(\theta)d\theta}. \end{aligned} \tag{3}$$

Unfortunately, especially in the case of large neural networks, the posterior is intractable to compute or sample from exactly. Therefore it is necessary to resort to approximation methods such as variational inference or MCMC.

### 2.4 Sequential Monte Carlo

Sequential Monte Carlo algorithms model a sequence of distributions $p_0(\theta), \ldots, p_m(\theta)$. An initial state of particles is drawn from $p_0$, and by repeatedly applying importance sampling, an MCMC algorithm and resampling steps, the initial samples are transformed to be a sample of $p_m(\theta)$. A basic outline is given in Algorithm 1. Under certain conditions, notably that sample distributions have to be invariant under the MCMC steps, this Monte Carlo scheme converges to the correct target [11].

Leveraging this fact, we can set

$$p_m(\theta) = p(\theta|\mathcal{D}) \propto p(\theta)p(\mathcal{D}|\theta), \tag{4}$$

pick a sequence of temperatures

$$0 = \lambda_0 < \lambda_1 < \cdots < \lambda_m = 1,$$

and use SMC to sample from $p_0(\theta), \ldots, p_m(\theta)$, where the distributions are given by

$$p_t(\theta) \propto p(\mathcal{D}|\theta)^{\lambda_t} p(\theta). \tag{5}$$

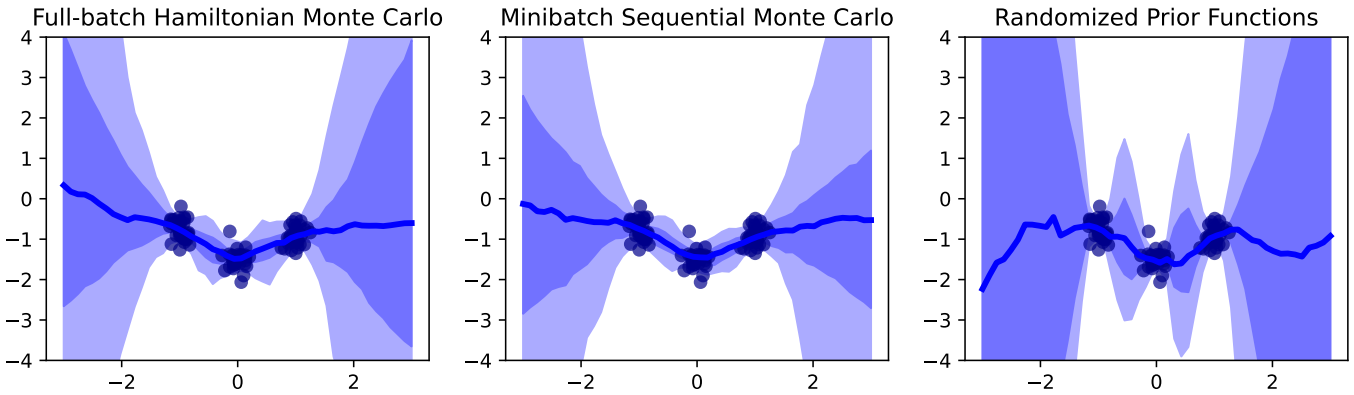


**Figure 1: The 25th-75th and 5th-95th quantiles of the predictive posterior as predicted by SMC and an ensemble with randomized prior functions in a supervised learning setting, compared to the gold standard Hamiltonian Monte Carlo without noise. Sequential Monte Carlo more closely resembles the posterior distribution as approximated by Hamiltonian MC.**

**Algorithm 1:** Base Sequential Monte Carlo

**Input:** sequence of target distributions $p_0(\theta), \ldots, p_m(\theta)$ and MCMC kernels $P_1, \ldots P_m$
**Result:** a sample $\theta_1, \ldots, \theta_n \sim p_m(\theta)$
$\theta_1, \ldots, \theta_n \sim p_0$
$w_1, \ldots, w_n \leftarrow 1$
**for** $t = 1, \ldots, m$ **do**
  $\theta_1, \ldots, \theta_n \sim \texttt{resample}(\theta_1, \ldots, \theta_n | w_1, \ldots, w_n)$
  **for** $j = 1, \ldots n$ **do**
    $w_j \leftarrow \frac{p_t(\theta_j)}{p_{t-1}(\theta_j)}$
  **end**
  **for** $j = 1, \ldots n$ **do**
    $\theta_j \leftarrow P_t(\theta_j)$
  **end**
**end**

Equation 5 effectively interpolates between the prior and the posterior. Setting the temperature sequence correctly is important, because a too coarse interpolation can cause the importance sampling weights to be unstable and a too fine interpolation wastes computation. Fortunately, automatic on-the-fly tuning methods exist that choose the next temperature based on the effective sample size of the current sample [6, 10].

## 3 SEQUENTIAL MONTE CARLO FOR BNNS

Having introduced the necessary background, to prepare for our main contribution, we next analyze SMC in the context of Bayesian Neural Networks.

Applying SMC to model the posterior of a Bayesian Neural Network is in practice similar to an ensemble. The particles $\theta_1, \ldots, \theta_n$ are individual models, and equipped with importance sampling weights $w_1, \ldots, w_n$ model a theoretically unbiased approximation of the predictive posterior distribution.

The model is initialized by sampling initial parameters $\theta_1, \ldots, \theta_n$ from the prior $p(\theta)$, and trained by running SMC with target distributions $\left(p(\mathcal{D}|\theta)^{\lambda_t} p(\theta)\right)_{t \geq 0}$. Unfortunately, typically in deep learning the data set $\mathcal{D}$ is so large that mini-batches are required to tractably compute the likelihood and gradients. This poses two problems:

(1) The reweighting step is now noisy, which lowers the quality of importance sampling.
(2) The MCMC step, which typically is gradient based, is now noisy. This means we may have to rely on MCMC methods that only approximately leave the target distribution invariant.

However, these problems can largely be alleviated, as the theoretical results by Llorente et al. [24] show that noisy importance sampling, which is central to the SMC algorithm, has higher variance but remains unbiased. Furthermore, Wenzel et al. [33] and Garriga-Alonso and Fortuin [17] show that accurate noisy MCMC kernels that leave the target distribution (approximately) invariant do exist, so we can use these kernels specifically crafted for mini-batch noise in our SMC algoritm.

Specifically, in our experiments we estimate the reweighting steps by sampling a single batch independently for every particle every iteration. As MCMC kernel we use the Symplectic Euler Langevin scheme with hyper-parameters as suggested by Wenzel et al. [33]. At each iteration $t$, the temperature in the next step $\lambda_{t+1} = \lambda_t + \delta$ is picked by keeping the effective sample size (ESS) at a desired level $d$:

$$\max_{\delta} \delta, \text{ such that: } \delta < 1 - \lambda_t, \text{ and}$$
$$\text{ESS} = \frac{\left(\sum_{j=1}^n w_j\right)^2}{\sum_{j=1}^n w_j^2} = \frac{\left(\sum_{j=1}^n p(\mathcal{D}|\theta_j)^\delta\right)^2}{\sum_{j=1}^n p(\mathcal{D}|\theta_j)^{2\delta}} > d. \tag{6}$$

Our initial experimental findings in a supervised learning setting, shown in Figure 1, show a comparison of the predictive posteriors approximated by noise-free Hamiltonian Monte Carlo, Mini-batch Sequential Monte Carlo, and an ensemble with randomized prior functions. Even with mini-batches, SMC with noisy likelihoods

**Algorithm 2:** Sequential Monte Carlo for BNNs

**Input:** Prior $p_0(\theta)$ and target $p_0(\theta)p(\mathcal{D}|\theta)$, MCMC algorithm
**Result:** A sample $\theta_1, \dots, \theta_n \sim p_0(\theta)p(\theta|\mathcal{D})$
$\theta_1, \dots, \theta_n \sim p_0$
$w_1, \dots, w_n \leftarrow 1$
$t \leftarrow 0$
$\lambda_0 \leftarrow 0$
**while** $\lambda_t < 1$ **do**
  Pick $\lambda_t > \lambda_{t-1}$ (eq. 6)
  $\log p_t(\theta) \leftarrow \log p_0(\theta) + \lambda_t \log p(\mathcal{D}|\theta)$
  $\theta_1, \dots, \theta_n \sim \texttt{resample}(\theta_1, \dots, \theta_n | w_1, \dots, w_n)$
  **for** $j = 1, \dots n$ **do**
    $\log w_j \leftarrow (\lambda_t - \lambda_{t-1}) \log p(\theta_j|\mathcal{D})$
  **end**
  $w_1, \dots, w_n = \texttt{normalize}(w_1, \dots, w_n)$
  **for** $j = 1, \dots n$ **do**
    $\theta_j \leftarrow \texttt{MCMC}(\theta_j, \log p_t(\theta))$
  **end**
  $t \leftarrow t + 1$
**end**

and gradients results in performance on par with the gold standard noise-free Hamiltonian Monte Carlo [27]. However, small mini-batches may require a finer grained interpolation, since the temperature schedule depends on maintaining a high enough ESS, which is known to be lower when using noisy reweighting [24].

## 4 SEQUENTIAL MONTE CARLO DQN (SMC-DQN)

With the goal of improving exploration, we construct an agent that can accurately quantify uncertainty in its $Q$-values by approximating the posterior distribution over its parameters given the transitions that have previously been observed. Specifically, we extend a standard DQN agent by replacing its point-wise estimator $Q_\theta(s, a)$ with an ensemble of neural networks $Q_{\theta_1}(s, a), \dots, Q_{\theta_n}(s, a)$ and weights $w_1, \dots w_n$ to maintain an approximation of the posterior $p(\theta|\mathcal{D}, \boldsymbol{\theta'})$, conditioned on the current replay buffer

$$\mathcal{D} = ((s_t, a_t, r_t, s_{t+1}))_{t=1\dots N}$$

and current target parameters

$$\boldsymbol{\theta'} = (\theta'_1, \dots, \theta'_n).$$

In line with the work by Schmitt et al. [32], a normal distribution

$$Q_\theta(s, a) - r(s, a) - \gamma \max_{a'} Q_{\theta'}(s', a') \sim \mathcal{N}(0, \sigma)$$

is used as a probabilistic interpretation of the squared temporal difference error, and to represent the uncertainty in the targets we define the likelihood to be a mixture distribution

$$\log \mathcal{L}(s, a, r, s'|\theta, \boldsymbol{\theta'}) = \log \sum_{i=1}^{n} \frac{1}{n} \exp\left( \frac{1}{2\sigma^2} [Q_\theta(s, a) - r(s, a) - \gamma \max_{a'} Q_{\theta'_i}(s', a')]^2 \right), \tag{7}$$

**Algorithm 3:** SMC-DQN

**Input:** MDP, batch size $B$, ensemble size $n$
**Result:** Posterior over $Q_\theta(s, a)$
$\theta_1, \dots, \theta_n \sim p(\theta)$
$w_1, \dots, w_n \leftarrow 1$
**while** *training* **do**
  $i \sim \texttt{uniform}(1, \dots, n)$
  $s_0 \sim \texttt{MDP}$
  $t \leftarrow 0$
  **while** *episode not done* **do**
    $a_t = \text{argmax}_a Q_{\theta_i}(s_t)$
    $r_t, s_{t+1} \sim \texttt{MDP}(s_t, a_t)$
    $\mathcal{B} = \mathcal{B} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
    **if** $|\mathcal{B}| = B$ **then**
      $\theta_1, \dots, \theta_n, w_1, \dots, w_n \leftarrow$ $\texttt{SMC}(p(\theta|\boldsymbol{\theta'}, \mathcal{D}), p(\theta|\boldsymbol{\theta'}, \mathcal{D} \cup \mathcal{B}))$ (eq. 11)
      $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{B}$
      $\mathcal{B} \leftarrow \emptyset$
    **end**
    **if** *time for target update* **then**
      $\theta'_{i,\text{new}} \leftarrow \theta_i$
      $\theta_1, \dots, \theta_n, w_1, \dots, w_n \leftarrow$ $\texttt{SMC}(p(\theta|\boldsymbol{\theta'}, \mathcal{D}), p(\theta|\boldsymbol{\theta'}_{\text{new}}, \mathcal{D}))$ (eq. 12)
      $\theta'_i \leftarrow \theta'_{i,\text{new}}$
    **end**
  **end**
**end**

contrasting BootDQN which shares no target values between ensemble members. The log posterior distribution is defined as

$$\log p(\theta|\boldsymbol{\theta'}, \mathcal{D}) \propto \log p(\theta) + \log \mathcal{L}(\mathcal{D}|\theta, \boldsymbol{\theta'}), \tag{8}$$

where

$$\log \mathcal{L}(\mathcal{D}|\theta, \boldsymbol{\theta'}) = \sum_{(s,a,r,s') \in \mathcal{D}} \log \mathcal{L}(s, a, r, s'|\theta, \boldsymbol{\theta'}). \tag{9}$$

After collecting a new batch of trajectories

$$\mathcal{B} = ((s_t, a_t, r_t, s_{t+1}))_{t=1,\dots,B}$$

by acting in the environment, the posterior distribution can be updated efficiently by noting that

$$\begin{aligned} \log p(\theta|\boldsymbol{\theta'}, \mathcal{D} \cup \mathcal{B}) &= \log p(\theta) + \log \mathcal{L}(\mathcal{D} \cup \mathcal{B}|\theta, \boldsymbol{\theta'}) \\ &= \log p(\theta) + \log \mathcal{L}(\mathcal{D}|\theta, \boldsymbol{\theta'}) + \log \mathcal{L}(\mathcal{B}|\theta, \boldsymbol{\theta'}) \\ &= \log p(\theta|\boldsymbol{\theta'}, \mathcal{D}) + \log \mathcal{L}(\mathcal{B}|\theta, \boldsymbol{\theta'}). \end{aligned} \tag{10}$$

Therefore, since our agent is currently holding a sample of $p(\theta|\boldsymbol{\theta'}, \mathcal{D})$, the posterior can be updated by running SMC on the sequence

$$p(\theta|\boldsymbol{\theta'}, \mathcal{D}),\ p(\theta|\boldsymbol{\theta'}, \mathcal{D})\mathcal{L}(\mathcal{B}|\theta, \boldsymbol{\theta'})^{\lambda_1},\ \dots,\ p(\theta|\boldsymbol{\theta'}, \mathcal{D})\mathcal{L}(\mathcal{B}|\theta, \boldsymbol{\theta'})^{\lambda_k},\ p(\theta|\boldsymbol{\theta'}, \mathcal{D})\mathcal{L}(\mathcal{B}|\theta, \boldsymbol{\theta'}), \tag{11}$$

which interpolates between the posterior given what was already known, and the new posterior including the new batch. This is

equivalent to Equation 5 when taking $p(\theta|\boldsymbol{\theta}', \mathcal{D})$ as prior and $p(\mathcal{B}|\theta, \boldsymbol{\theta}')$ as likelihood.

To evaluate Equation 11, the likelihood $\mathcal{L}(\mathcal{B}|\theta, \boldsymbol{\theta}')$ only requires the latest batch, and can easily be computed exactly. However, we choose to approximate $p(\theta|\boldsymbol{\theta}', \mathcal{D})$ by sampling mini-batches uniformly from the replay buffer, since computing it exactly would require summing over the entire replay buffer.

Updating the target networks $\boldsymbol{\theta}'$ changes the target distribution, meaning that the sample

$$\theta_1, \ldots, \theta_n, \quad w_1, \ldots, w_n$$

is no longer a sample of the posterior with respect to the updated targets, i.e., $p(\theta|\boldsymbol{\theta}'_{\text{new}}, \mathcal{D})$. Therefore, the typical target update $\theta'_i \leftarrow \theta_i$ is now accompanied by another SMC step, which smoothly interpolates between the distribution conditioned on the old targets and the distribution conditioned on the new targets via the sequence

$$p(\theta)\mathcal{L}(\mathcal{D}|\theta, \boldsymbol{\theta}'_{\text{old}})^{(1-\lambda_t)} p(\theta)\mathcal{L}(\mathcal{D}|\theta, \boldsymbol{\theta}'_{\text{new}})^{\lambda_t} \rightarrow p(\theta)\mathcal{L}(\mathcal{D}|\theta, \boldsymbol{\theta}'_{\text{new}}), \quad (12)$$

as $\lambda_t$ increases from 0 to 1. This transforms a sample of the posterior with respect to the previous targets to a sample with respect to the new targets. Intuitively speaking, this trains the main networks $\theta_1, \ldots, \theta_n$ to match the new targets, before acting in the environment again.

Similarly to BootDQN, actions are selected by Thompson sampling one model $\theta_i$, and committing to this model for a full episode. Algorithm 3 shows the general structure with both updates, where `SMC` refers to Algorithm 2. The Symplectic Euler Langevin scheme [33] is used as MCMC step. We opt not to resample in every SMC step, in order to maintain more diversity in the ensemble. Further, to maximize the exploration behaviour of the agent, we sample uniformly from the particles as opposed to utilizing the weights.

While the structure is similar to a typical DQN implementation, a major difference is that the SMC steps take significantly more time than typical Q-value updates. For each batch from the environment, the agent adapts its model to match the posterior given the new replay buffer, and also after each target update the agent trains its networks to match the posterior given the new targets. The speed at which the agent can condition its model on the new data causes a trade off between computation complexity and sample complexity. In this work we are mostly concerned with sample complexity in the exploration setting. Furthermore, there is a very clear split between samples that are already in the data set $\mathcal{D}$ and samples that have yet to be incorporated $\mathcal{B}$. It is assumed that the replay buffer $\mathcal{D}$ is large enough to store every experienced transition.

## 5 EXPERIMENTAL STUDY

So far we have introduced SMC-DQN, an agent that is architecturally similar to DQN with an ensemble, but employs SMC to maintain a posterior over $Q$-value functions. To test our agent's ability to explore in difficult sparse reward environments, we evaluate on the exploration tasks in Behaviour Suite (BSuite) [30]. We also test on BSuite's Mountain Car environment, which requires further exploration after reaching the goal state to recover an optimal policy, to test whether our agent still explores effectively after finding reward.

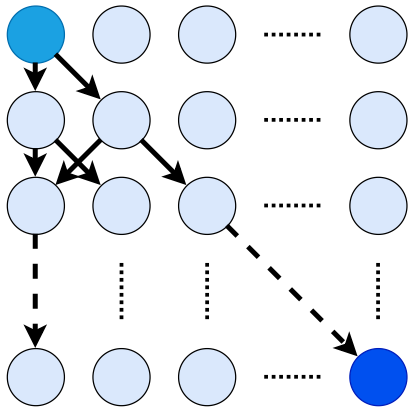

**Figure 2: Graphical representation of Deep Sea. The agent starts in the top left state, and transitions downwards to the left or right at every time step depending on the action taken. The only positive reward is granted in the bottom right state**

### 5.1 Environments

Our agents run for 10000 episodes on both Deep Sea environments, 1000 episodes on Cartpole, and 300 on Mountain Car, which, except for in Mountain Car, is the number of episodes prescribed by BSuite. We only run our agents for 300 episodes as opposed to BSuite's 1000 episodes on Mountain Car because performance plateaus already after 100 episodes.

*Deep Sea.* Deep Sea is a one hot encoded chain-like environment, where the observations are a matrix of size $n \times n$ and there are two discrete actions. The agent starts at the top left state, moves down one row at every time step, and also moves left or right depending on the action taken. A graphical view is shown in Figure 2. Only the bottom right state results in positive reward, meaning that the agent has to execute the correct action $n$ times in a row to observe any positive reward in an episode; hence this environment is impossible to solve with an $\epsilon$-greedy approach. Further, the agent receives a small negative reward when moving to the right. In our experiments, we use an environment of size $30 \times 30$ and report the episodic return excluding the small negative reward, so an optimal policy receives reward 1 in every episode. This is a difficult exploration task where the agent can not rely on generalization. In both deep sea variants, which action goes left or right is randomized for each state to avoid trivial solutions.

*Deep Sea Stochastic.* Stochastic Deep Sea adds a $(1 - \frac{1}{n})$ probability that the 'right' action fails and goes to the left instead. Furthermore, on the bottom left state the agent receives a stochastic reward with mean 0. We use an environment of size $20 \times 20$ and in our plots we normalize the reward an agent received by the theoretical optimal mean return of $(1 - \frac{1}{n})^n$ so that an optimal policy will receive an average episodic return of 1. This environment requires the agent's exploration method to be able to deal with stochasticity.

*Cartpole Swingup.* Cartpole Swingup is a sparse reward control task with continuous states and discrete actions, similar to the classic Cartpole environment, except that the pole starts in a downward

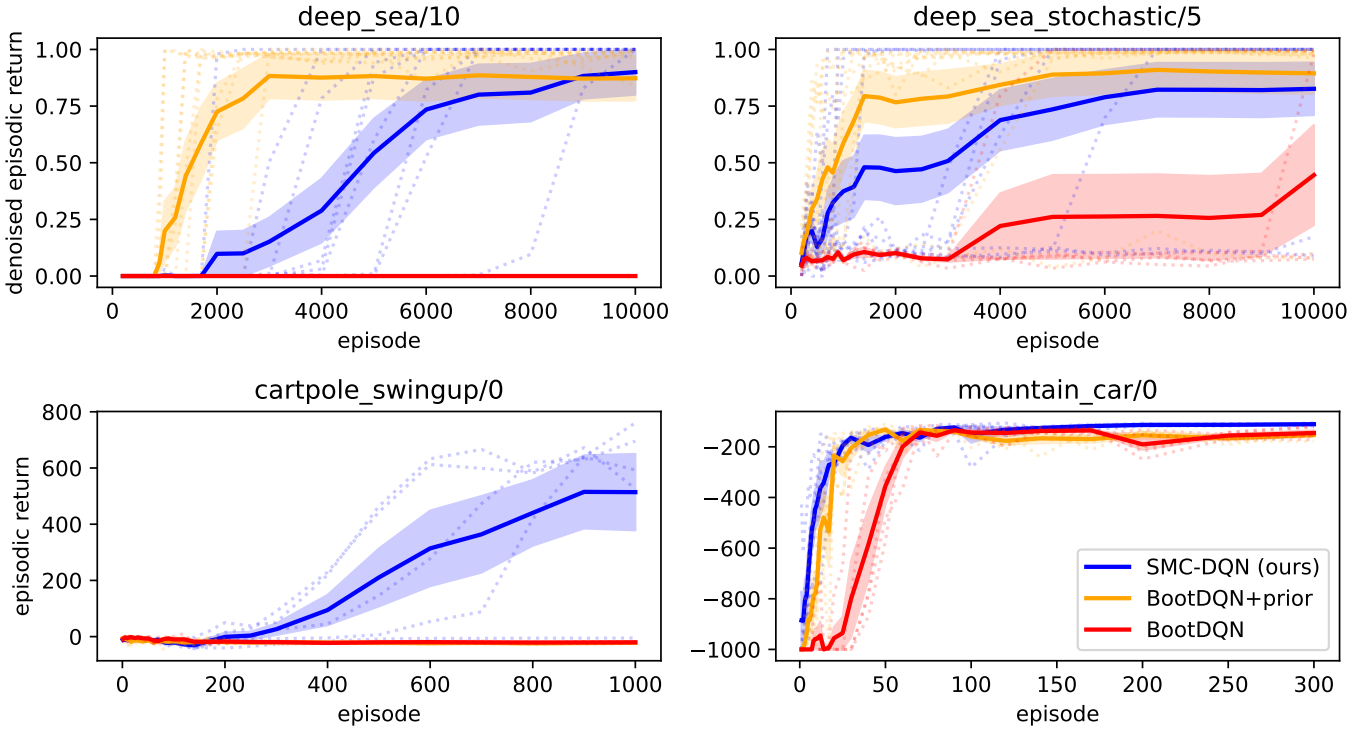


**Figure 3: Learning curves over the BSuite environments. The solid line is the mean of 10 seeds for the Deep Sea environments, and 5 seeds for Cartpole Swingup and Mountain Car. The shaded area denotes the standard error of the mean. Dashed lines show individual seeds.**

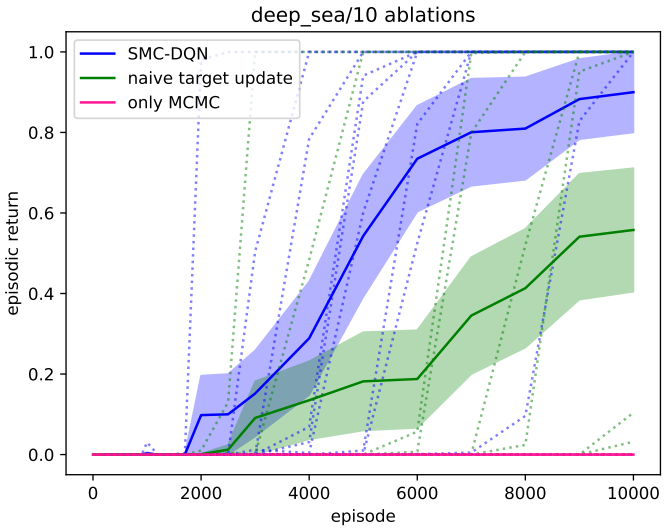


**Figure 4: Learning curves on $30 \times 30$ deep sea for 2 ablations compared to SMC-DQN. The solid line is the mean of 10 seeds for 'naive target update' and 5 seeds for 'only MCMC'. Shaded areas denote the standard error of the mean and dashed lines show individual seeds.**

position and the agent has to explore to find that reward is received when the pole is upright. The agent also receives negative reward for moving the cart, disincentivizing exploration. This is a difficult exploration task in a continuous state space.

*Mountain Car.* Mountain Car is a control task with continues states and discrete actions, where the agent has to drive an underpowered car up a hill by building up momentum in a valley. In BSuite's implementation, an episode is 1000 steps and cancels early when the agent reaches the goal. The agent receives $-1$ reward at every time step, and has to learn to end the episode as fast as possible to maximize reward. Exploring to find the goal state is not necessarily difficult in this environment, but finding an optimal policy that reaches the goal state quickly can be difficult.

## 5.2 Baselines and Hyper-parameters

*Baselines.* We compare against the baseline Bootstrapped DQN agents in BSuite with and without randomized prior functions, since our algorithm can be considered an extension to the Bootstrapped DQN agent without priors, and the bootstrapped DQN agent with priors is also similar to our method and known to be a strong baseline in the exploration tasks that we consider.

*Hyper-parameters.* We test all our agents with ensembles of size 10. For SMC-DQN, we use the same hyper-parameters across each experiment, except for Cartpole Swingup, where we set the standard deviation of the likelihood to $\sigma = 1$ instead of $\sigma = 0.1$ due to the

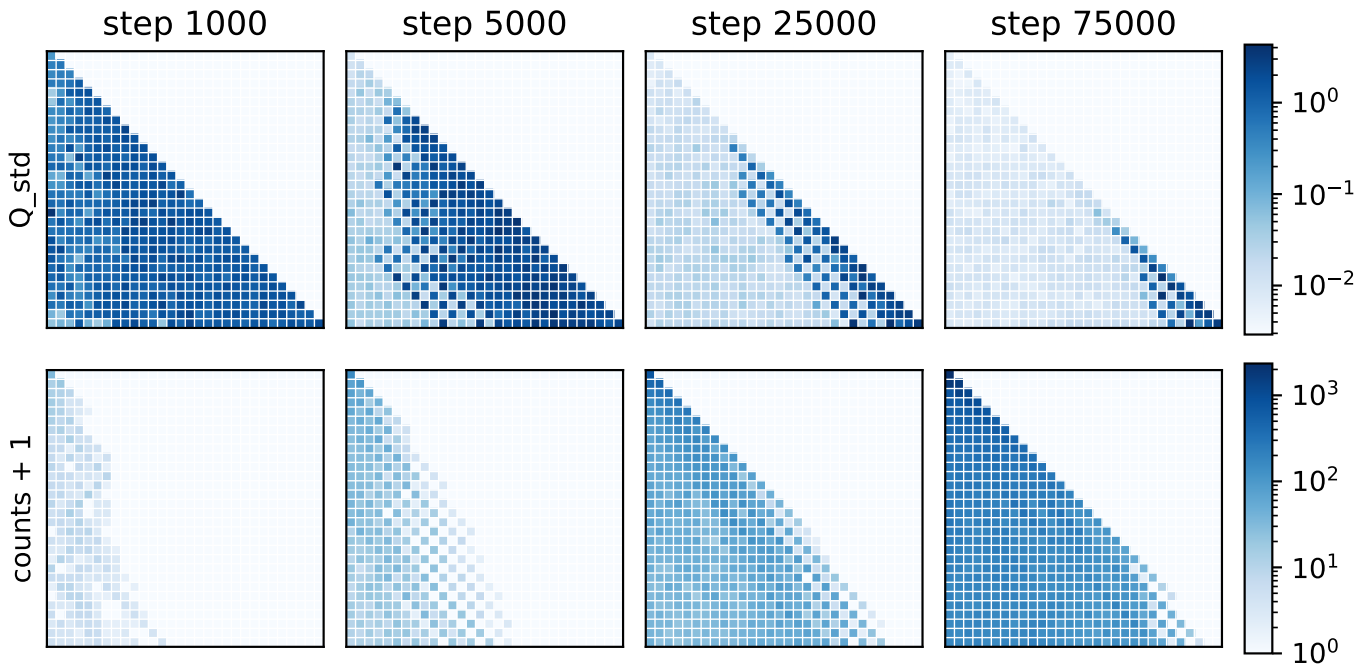


**Figure 5: Graphical view of the standard deviation and visitation counts of the values during training on deep_sea/10. Each pixel shows standard deviation in the maximum $Q$-value (top row) of that state as predicted by the ensemble. The bottom row shows the logarithmic visitation counts. In each figure, the top left state is the initial state, and the bottom right state is the goal state. The diagonal is the only sequence of states that results in reward. States in the upper triangle are unreachable.**

much larger scale of cumulative reward in Cartpole Swingup. For our baselines, we use the hyper-parameters provided by BSuite. For the exact hyper-parameters we refer to the Supplementary Material.

### 5.3 Experimental Results

Figure 3 shows the performance of the agents on each task. It can be seen that SMC-DQN outperforms BootDQN without priors on all our benchmarks. On Deep Sea it achieves comparable performance to BootDQN with priors, and significantly outperforms BootDQN with priors on Cartpole Swingup, where BootDQN at this ensemble size fails to learn a meaningful policy even with prior functions. When increasing the prior scale on BootDQN with priors we still did not observe effective exploration with an ensemble of size 10 on Cartpole Swingup. We refer to the supplementary material for these results. Further, on Mountain Car SMC-DQN learns at the same speed as BootDQN with priors in the beginning, but converges to a slightly better policy.

To confirm that SMC-DQN's uncertainty quantification is sensible, Figure 5 shows the standard deviation in Q-values as proxy to uncertainty, and visitation counts of SMC-DQN on $30 \times 30$ Deep Sea. Each square represents a state in Deep Sea, with the starting state in the top left and the bottom right state being the only state that results in positive reward. It is clearly visible how the uncertainty in values stays high for unvisited states, meaning that the approximated posterior distribution can correctly lead the agent towards under-explored areas.

### 5.4 Ablation study

Finally, in Figure 4 we study the effect of individual components of SMC-DQN by running two ablations.

*Only MCMC.* First, to test whether SMC aids in approximating the posterior, we run an ablated agent which only uses the MCMC kernel, the Symplectic Euler Langevin scheme, to approximate the posterior.

The weights $w_1, \ldots, w_n$ are dropped, and the models are now treated as an ensemble of independent markov chains. The SMC step is replaced by 100 steps of the MCMC kernel, which is the number of steps that SMC would run this kernel for on the final target distribution. The split between the new batch and old buffer is removed, meaning that all transitions are immediately added to the replay buffer. The posterior density is approximated with samples from the replay buffer:

$$\log p(\theta_i|\boldsymbol{\theta}', \mathcal{D}) \approx \log p(\theta) + \sum_{i=1}^{B} -\frac{1}{2\sigma^2}\left(Q_\theta(s_i, a_i) - r_i - {}' \max_{a'} \gamma Q_{\theta_i'}(s', a')\right)^2,$$

for each $i = 1, \ldots, n$. All other hyperparameters are left unchanged.

This agent fails to observe any reward, suggesting that the MCMC kernel by itself can not keep a diverse set of models to explore with.

*Naive target update.* Further, we check whether the new target update, which is theoretically required to maintain an approximation of the posterior, also improves reward in RL experiments. We

run an ablated agent that naively updates its targets, omitting the SMC step.

The only change made to the algorithm is to remove the SMC step after updating the targets. This means that when the target networks are updated, we simply set $\theta'_i \leftarrow \theta_i$ for every $i = 1, \ldots, n$, and leave the weights and parameters $\theta_1, \ldots, \theta_n$ unchanged until the main update when the next batch of transitions is collected.

It can be observed that this agent can still receive reward, but on average later and less reliably.

### 5.5 Discussion

Our results show a clear gap between Deep Sea and the continuous environments in the performance relative to the baseline. We hypothesize that this is due to the fact that the likelihood does not explain the one-hot encoded environment Deep Sea very well. On the continuous environments, agents can exploit the generalization capabilities of neural networks, allowing the posterior to model sensible generalization behaviours. However, since Deep Sea is under the hood a tabular environment, a perfect uncertainty mechanism on Deep Sea would model every state as independent unless they are connected, while the Bayesian posterior attempts to generalize over all states through the dependency on $\theta$. This means that the posterior distribution does not necessarily provide more accurate uncertainty quantification than an ensemble with randomized priors of large scale.

## 6 RELATED WORK

Approximating the Bayesian posterior over Q-values to quantify uncertainty and drive exploration in Reinforcement Learning has been previously studied by several prior works. Monte Carlo Dropout [16] is a Variational Inference method that uses dropout layers, which probabilistically disables neural network connections to create stochasticity in the outputs. The network together with the dropout probabilities are then trained so that the outputs match the predictive posterior. Furthermore, NoisyNets [15] adds stochasticity by modelling the posterior as independent normal distributions for each weight. Azizzadenesheli et al. [1] replaces the last layer of a neural network with Bayesian Linear Regression, inferring the posterior distribution only over the parameters in the last layer. Epistemic Value Estimation [32] increases the size of the model and uses a Laplace approximation to approximate the posterior distribution over the full set of parameters. Due to the complex and non-linear nature of neural networks, it is not clear in these variational inference methods how the choice of model class for the posterior distribution affects the approximation accuracy. Our algorithm, on the other hand, uses MCMC methods to approximate the posterior, which is not restricted to a model class that has to be picked in advance. This leads to unbiased approximations in theory, although due to practical considerations using an MCMC method to approximate a complex posterior distribution is not a simple task.

Langevin DQN [13] is an MCMC-based algorithm that approximates the posterior using Langevin Dynamics, which essentially perturbs the gradients of a DQN agent with normally distributed noise. Similarly, LMCDQN [20] uses a more intricate MCMC kernel with a preconditioner to improve computational performance. These methods are comparable to the inner MCMC kernels inside the SMC algorithm in our agent, and could be used as drop-in replacements to the Symplectic Euler Langevin algorithm that we used for its empirically established accuracy [33]. While Langevin DQN and LMCDQN can also be used to train an ensemble of Q-networks, SMC-DQN differs in that SMC theoretically and practically ties together the ensemble members as opposed to running separate Markov Chains. Furthermore, we use a mixture distribution as likelihood so that the ensemble consistently models one posterior. Further, we update our target parameters in a theoretically sound manner that takes into account that the posterior distribution is conditioned on all target parameters jointly.

## 7 CONCLUSION

This paper introduced the novel idea of using sequential Monte Carlo to train an ensemble in order to approximate the Bayesian posterior distribution. Specifically, we modified the BootDQN algorithm to use Sequential Monte Carlo, thus keeping track of a posterior over the Q-values in a theoretically sound manner.

We found that such an approach is able to maintain a diverse set of models that can drive exploration in difficult-to-explore environments such as Deep Sea and Cartpole Swingup. Especially in continuous state environments, the uncertainty quantification provided by the posterior distribution leads to better exploration compared to our baselines.

In the future, we intend to investigate the influence of the choice of prior and likelihood, derive methods to synthesize meaningful priors and likelihoods, as well as extend our method to more intricate reinforcement-learning architectures.

## ACKNOWLEDGMENTS

This work has received funding from the European Union's Horizon 2020 research and innovation programme, under grant agreements 964505 (E-pi) and 952215 (TAILOR).

# REFERENCES


[1] Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. 2018. Efficient exploration through Bayesian Deep Q-Networks. In *2018 Information Theory and Applications Workshop (ITA)*. IEEE.

[2] Chenjia Bai, Lingxiao Wang, Lei Han, Jianye Hao, Animesh Garg, Peng Liu, and Zhaoran Wang. 2021. Principled Exploration via Optimistic Bootstrapping and Backward Induction. In *International Conference on Machine Learning*.

[3] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. 2016. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, Vol. 29.

[4] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2018. Exploration by random Network Distillation. In *International Conference on Learning Representations*.

[5] Alberto Cabezas, Adrien Corenflos, Junpeng Lao, and Rémi Louf. 2024. BlackJAX: Composable Bayesian inference in JAX. arXiv:2402.10797 [cs.MS]

[6] Michael Cai, Marco Del Negro, Edward Herbst, Ethan Matlin, Reca Sarfati, and Frank Schorfheide. 2021. Online Estimation of DSGE Models. *The Econometrics Journal* 24, 1 (2021), C33–C58.

[7] Tianqi Chen, Emily Fox, and Carlos Guestrin. 2014. Stochastic Gradient Hamiltonian Monte Carlo. In *International Conference on Machine Learning*.

[8] Kamil Ciosek, Quan Vuong, Robert Loftin, and Katja Hofmann. 2019. Better exploration with optimistic actor critic. In *Advances in Neural Information Processing Systems*, Vol. 32.

[9] Francesco D'Angelo and Vincent Fortuin. 2021. Repulsive Deep Ensembles are Bayesian. In *Advances in Neural Information Processing Systems*, Vol. 34.

[10] Hai-Dang Dau and Nicolas Chopin. 2022. Waste-free Sequential Monte Carlo. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 84, 1 (2022), 114–148.

[11] Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. 2006. Sequential Monte Carlo Samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68, 3 (2006), 411–436.

[12] Thomas G. Dietterich. 2000. Ensemble Methods in Machine Learning. In *Multiple Classifier Systems*. Springer Berlin Heidelberg, 1–15.

[13] Vikranth Dwaracherla and Benjamin Van Roy. 2021. Langevin DQN. arXiv:2002.07282 [cs.LG]

[14] Matthew Fellows, Kristian Hartikainen, and Shimon Whiteson. 2021. Bayesian Bellman Operators. In *Advances in Neural Information Processing Systems*, Vol. 34.

[15] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. 2019. Noisy Networks for Exploration. arXiv:1706.10295 [cs.LG]

[16] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian Approximation: Representing Model Uncertainty in deep Learning. In *International Conference on Machine Learning*.

[17] Adrià Garriga-Alonso and Vincent Fortuin. 2021. Exact Langevin Dynamics with Stochastic Gradients. *arXiv:2102.01691* (2021).

[18] Mario Geiger, Arthur Jacot, Stefano Spigler, Franck Gabriel, Levent Sagun, Stéphane d'Ascoli, Giulio Biroli, Clément Hongler, and Matthieu Wyart. 2020. Scaling description of generalization with number of parameters in deep learning. *Journal of Statistical Mechanics: Theory and Experiment* 2020, 2 (2020).

[19] Eyke Hüllermeier and Willem Waegeman. 2021. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning* 110, 3 (2021), 457–506.

[20] Haque Ishfaq, Qingfeng Lan, Pan Xu, A. Rupam Mahmood, Doina Precup, Anima Anandkumar, and Kamyar Azizzadenesheli. 2023. Provable and Practical: Efficient Exploration in Reinforcement Learning via Langevin Monte Carlo. arXiv:2305.18246 [cs.LG]

[21] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in Neural Information Processing Systems*, Vol. 30.

[22] Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. 2021. Sunrise: A Simple Unified Framework for Ensemble Learning in Deep Reinforcement Learning. In *International Conference on Machine Learning*.

[23] Qiang Liu and Dilin Wang. 2016. Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. In *Advances in Neural Information Processing Systems*, Vol. 29.

[24] Fernando Llorente, Luca Martino, Jesse Read, and David Delgado-Gómez. 2022. Optimality in Noisy Importance Sampling. *Signal Processing* 194 (2022), 108455.

[25] Owen Lockwood and Mei Si. 2022. A Review of Uncertainty for Deep Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533.

[27] Radford M. Neal et al. 2011. MCMC using Hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC.

[28] Ian Osband, John Aslanides, and Albin Cassirer. 2018. Randomized Prior Functions for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 31.

[29] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. Deep Exploration via Bootstrapped DQN. In *Advances in Neural Information Processing Systems*, Vol. 29.

[30] Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvari, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado Van Hasselt. 2020. Behaviour Suite for Reinforcement Learning. In *International Conference on Learning Representations*.

[31] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. 2017. Count-Based Exploration with Neural Density Models. In *International Conference on Machine Learning*.

[32] Simon Schmitt, John Shawe-Taylor, and Hado van Hasselt. 2023. Exploration via Epistemic Value Estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37.

[33] Florian Wenzel, Kevin Roth, Bastiaan Veeling, Jakub Swiatkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. 2020. How Good is the Bayes Posterior in Deep Neural Networks Really?. In *International Conference on Machine Learning*.

[34] Tianpei Yang, Hongyao Tang, Chenjia Bai, Jinyi Liu, Jianye Hao, Zhaopeng Meng, Peng Liu, and Zhen Wang. 2021. Exploration in Deep Reinforcement Learning: From Single-Agent to Multiagent Domain. *IEEE Transactions on Neural Networks and Learning Systems* (2021).