Bayesian Deep Q-Learning via Sequential Monte Carlo

Anonymous Author(s)
Affiliation
Address
email

Abstract

Exploration in reinforcement learning remains a difficult challenge. Recently, ensembles with randomized prior functions have been popularized to quantify uncertainty in the value model, in order to drive exploration with success. However these ensembles have no theoretical guarantee to resemble the actual posterior. In this work, we view training ensembles from the perspective of sequential Monte Carlo, 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 improved exploration capabilities over a regular ensemble.

1 Introduction

Reinforcement learning algorithms are still notoriously sample inefficient. One pressing reason is the difficulty of exploring an environment efficiently while assuming little prior knowledge. 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 or use Thompson sampling to explore. However, quantifying uncertainty for deep neural networks is in itself a difficult task [Hüllermeier and Waegeman, 2021, Lockwood and Si, 2022].

Ensembles of neural networks have been shown to provide better predictive accuracy over a single model in supervised learning tasks [Lakshminarayanan et al., 2017], as well as suitable methods for uncertainty quantification for exploration in reinforcement learning [Osband et al., 2016, 2018, Fellows et al., 2021]. While ensembles with independent models of identical architecture tend to collapse to the same predictive model [Geiger et al., 2020], there are several techniques developed to prevent this, such as regularization, adversarial learning [Lakshminarayanan et al., 2017], bootstrapping the data [Osband et al., 2016], and adding additive priors [Osband et al., 2018]. Furthermore, some techniques such as Stein Variational Gradient Descent [Liu and Wang, 2016, D'Angelo and Fortuin, 2021] 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, while retaining the practicality 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) methods, which are theoretically unbiased but have high variance, and Variational Inference methods, which have low variance but are typically biased. Recently both types of methods have been altered specifically for application to neural networks, with MCMC showing strong results in large networks [Chen et al., 2014, Wenzel et al., 2020, Garriga-Alonso and Fortuin, 2021]. However, for complex multimodal distributions, MCMC methods can struggle to find every mode [Del Moral et al., 2006]. This is an important drawback in deep learning, where the posterior distribution is likely very ill behaved, and especially in reinforcement learning where

Submitted to 16th European Workshop on Reinforcement Learning (EWRL 2023). Do not distribute.

under-approximation of the posterior complexity might lead underestimating the uncertainty and therefore failure of exploration. Sequential Monte Carlo can be a remedy to these issues in non-deep learning applications [Del Moral et al., 2006], and is also promising for deep learning noting the success of ensembles.

In this work, we unify ensembles and MCMC methods by using sequential Monte Carlo algorithms to train an ensemble in a Bayesian manner, to enjoy both the practical effectiveness of ensembles and theoretical foundations of MCMC.

Our contributions are two-fold:

1. We introduce existing sequential Monte Carlo algorithms as feasible methods to train ensembles so that they serve as proper approximations to the Bayes posterior.
2. We modify the BootDQN algorithm [Osband et al., 2016] to use sequential Monte Carlo instead, which keeps track of a posterior over the Q-values in a theoretically sound manner.

2 Background

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

$$Q^\pi(s,a) = R(s,a) + \gamma \mathbb{E}_{T,\pi} \sum_{t=0}^{T} \left[\gamma^t r_t \mid s_0 = s, a_0 = a\right],$$

which denotes the expected discounted future reward if the agent executes action a in state s and follows the policy π afterwards.

Since the transition function T and reward function R are assumed to be unknown by the agent, solving the Markov Decision Process 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), where a parameterized neural network Q_θ is learned and the corresponding policy picks $\operatorname{argmax}_a Q_\theta(s,a)$ in every state [Mnih et al., 2015]. To improve exploration, the BootDQN algorithm [Osband et al., 2016] 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_{θ_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_{θ_i} is equipped with their own target network $Q_{\theta_i'}$, and gets updated with their own targets:

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

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 underexplored 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 [Osband et al., 2018].

A randomized prior function is a fixed prior 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 rule is modified:

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

Each ensemble member having a unique prior function causes unique generalization behaviour on unobserved data, which keeps the ensemble outputs diverse in underexplored states. Note that randomized prior functions are not unique to reinforcement learning and can be used in supervised learning settings as well.

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_θ parameterized by $\theta \in \Theta$, with some prior distribution $\pi(\theta)$ over Θ.

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 θ.

$$\pi(\theta|\mathcal{D}) = \frac{\pi(\mathcal{D}|\theta)\pi(\theta)}{\int \pi(\mathcal{D}|\theta)\pi(\theta)d\theta} = \frac{\prod_{i=1}^n \mathcal{L}(y_i|f_\theta(x_i)\pi(\theta)}{\int \prod_{i=1}^n \mathcal{L}(y_i|f_\theta(x_i)\pi(\theta)d\theta}$$

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 and Markov Chain Monte Carlo.

2.4 Sequential Monte Carlo

Sequential Monte Carlo algorithms model a sequence of distributions $\pi_0(\theta), \ldots, \pi_n(\theta)$. An initial state of particles is drawn from π_0, and by repeatedly applying importance sampling, a Markov chain Monte Carlo (MCMC) algorithm and resampling steps, the initial samples are transformed to be a sample of $\pi_n(\theta)$. A basic outline is given in Algorithnm 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 [Del Moral et al., 2006].

Algorithm 1: Base Sequential Monte Carlo

Input: sequence of target distributions $\pi_0(\theta), \ldots, \pi_m(\theta)$ and MCMC kernels $P_1, \ldots P_m$
Result: a sample $\theta_1, \ldots, \theta_n \sim \pi_m(\theta)$
$\theta_1, \ldots, \theta_n \sim \pi_0$; `/* Draw initial particles */`
$w_1, \ldots, w_n \leftarrow 1$;
for $t = 1, \ldots, m$ **do**
  for $j = 1, \ldots n$ **do**
    $w_j \leftarrow \frac{\pi_t(\theta_j)}{\pi_{t-1}(\theta_j)}$; `/* Reweighting */`
  end
  for $j = 1, \ldots n$ **do**
    $\theta_j \leftarrow P_t(\theta_j)$; `/* Apply Markov chain kernel */`
  end
  $\theta_1, \ldots, \theta_n \sim$ `resample`$(\theta_1, \ldots, \theta_n | w_1, \ldots, w_n)$; `/* Resample with replacement */`
end

Leveraging this fact, we can set $\pi_n(\theta) = \pi(\theta|\mathcal{D}) \propto \pi(\theta)\pi(\mathcal{D}|\theta)$ and pick a sequence of temperatures $0 = \gamma_0 < \gamma_1 < \cdots < \gamma_n = 1$ and use SMC to sample from the sequence $\pi_0(\theta), \ldots, \pi_n(\theta)$, where distribution is given by

$$\pi_t(\theta) \propto \pi(\mathcal{D}|\theta)^{\gamma_t}\pi(\theta).$$

This effectively interpolates between the prior and the posterior. Setting the temperature sequence correctly is important, because a too course interpolation can cause the importance sampling weights

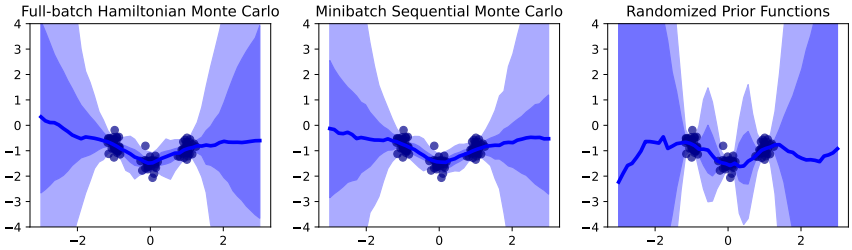

Figure 1: Sequential Monte Carlo and randomized prior functions in a supervised learning setting, compared against the gold standard Hamiltonian Monte Carlo without noise. Sequential Monte Carlo more closely resembles the posterior distribution as approximated by Hamiltonian Monte Carlo.

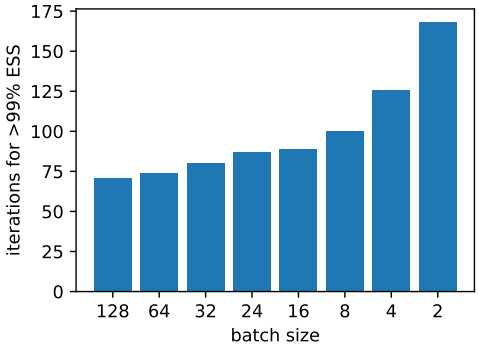

Figure 2: Number of interpolation iterations required in Sequential Monte Carlo to achieve 99% effective sample size for decreasing batch size, in the same experimental setup as Figure 1.

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 [Cai et al., 2021, Dau and Chopin, 2022].

3 Sequential Monte Carlo for 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 an unbiased approximation of the predictive posterior distribution.

The model is initialized by sampling initial parameters $\theta_1, \ldots, \theta_n$ from the prior $\pi(\theta)$, and trained by running Adaptive SMC with target distributions $\pi(\mathcal{D}|\theta)^{\gamma_t}\pi(\theta)$. 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 potential problems:

1. The reweighting step is now noisy.
2. The MCMC step, which typically is gradient based, is now noisy.

A full theoretical analysis of these problems we leave to future work. However, the theoretical results of [Llorente et al., 2022] show that noisy importance sampling has higher variance but remains unbiased. Furthermore, [Wenzel et al., 2020] and [Garriga-Alonso and Fortuin, 2021] show that accurate noisy MCMC schemes that leave the target distribution (approximately) invariant do exist.

Our initial experimental findings in a supervised learning setting, shown in Figures 1 and 2, show that even with very small mini batches, SMC with noisy likelihoods and gradients results in performance on par with the gold standard noise-free Hamiltonian Monte Carlo [Neal et al., 2011], albeit at the cost of requiring a finer grained temperature schedule for lower batch sizes. This is to be expected, as the temperature schedule depends on maintaining a high enough effective sample size, which is known to be lower when using noisy reweighting [Llorente et al., 2022].

Algorithm 2: Sequential Monte Carlo for BNNs

Input: Prior $\pi_0(\theta)$ and target $\pi_0(\theta)\pi(\mathcal{D}|\theta)$, MCMC algorithm
Result: A sample $\theta_1, \dots, \theta_n \sim \pi_0(\theta)\pi(\theta|\mathcal{D})$

```
θ_1, ..., θ_n ~ π_0 ;                                  /* Draw initial particles */
w_1, ..., w_n ← 1;
t ← 0;
γ_0 ← 0;
while γ_t < 1 do
    Pick γ_t > γ_{t-1} based on current state;
    log π_t(θ) ← log π_0(θ) + γ_t log π(D|θ);
    for j = 1, ... n do
        log w_j ← γ_t log π(θ_j|D) ;          /* Reweighting, evaluating likelihood */
    end
    w_1, ..., w_n = normalize(w_1, ..., w_n);
    for j = 1, ... n do
        θ_j ← MCMC(θ_j, log π_t(θ)) ;                  /* Apply Markov chain kernel */
    end
    θ_1, ..., θ_n ~ resample(θ_1, ..., θ_n|w_1, ..., w_n) ;                /* Resample */
    t ← t + 1;
end
```

4 Sequential Monte Carlo DQN

The ensemble of a BootDQN agent can be replaced with an ensemble trained by sequential Monte Carlo with minor changes to the architecture. Most notably, along with the parameters of the ensemble $\theta_1, \dots, \theta_n$, the agent now also stores the importance sampling weights $w_1, \dots, w_n$.

Equipped with a likelihood and prior, the agent attempts to maintain a posterior over the parameters of a Q-network, conditioned on the current replay buffer $\mathcal{D}_N = ((s_t, a_t, r_t, s_{t+1}))_{t=1,\dots,N}$ and current target parameters $\theta'_1, \dots, \theta'_n$. This is achieved by defining a joint likelihood over (θ, θ'). In line with [Schmitt et al., 2023], a normal distribution $Q_\theta(s,a) - r(s,a) - \operatorname{argmax}_{a'} Q_{\theta'}(s',a') \sim \mathcal{N}(0,\sigma)$ is used as a probabilistic interpretation of the squared temporal difference error, resulting in the log-likelihood

$$\log \mathcal{L}(\theta, \theta', \mathcal{D}) = -\frac{1}{2\sigma^2} \sum_{t=1}^{N} \left[Q_\theta(s_t, a_t) - r_t - \operatorname*{argmax}_{a'} Q_{\theta'}(s_{t+1}, a') \right]^2 .$$

The log posterior distribution is defined then as

$$\log \pi(\theta|\theta', \mathcal{D}) \propto \log \pi(\theta) + \log \mathcal{L}(\theta, \theta', \mathcal{D}).$$

After collecting a batch of trajectories $\mathcal{B} = ((s_t, a_t, r_t, s_{t+1}))_{t=1,\dots,B}$, the posterior distribution can be updated efficiently by noting that

$$\begin{aligned}
\log \pi(\theta|\theta', \mathcal{D} \cup \mathcal{B}) &= \log \pi(\theta) + \log \mathcal{L}(\theta, \theta', \mathcal{D} \cup \mathcal{B}) \\
&= \log \pi(\theta) + \sum_{t=1}^{N+B} -\frac{1}{2\sigma^2} \left[Q_\theta(s_t, a_t) - r_t - \operatorname*{argmax}_{a'} Q_{\theta'}(s_{t+1}, a') \right]^2 \\
&= \log \pi(\theta) + \log \mathcal{L}(\theta, \theta', \mathcal{D}) + \sum_{t=N}^{N+B} -\frac{1}{2\sigma^2} \left[Q_\theta(s_t, a_t) - r_t - \operatorname*{argmax}_{a'} Q_{\theta'}(s_{t+1}, a') \right]^2 .
\end{aligned}$$

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

$$\pi(\theta|\theta', \mathcal{D}),\ \pi(\theta|\theta', \mathcal{D})\mathcal{L}(\theta, \theta', \mathcal{B})^{\lambda_1}, \dots, \pi(\theta|\theta', \mathcal{D})\mathcal{L}(\theta, \theta', \mathcal{B})^{\lambda_k},\ \pi(\theta|\theta', \mathcal{D})\mathcal{L}(\theta, \theta', \mathcal{B}),$$

interpolating between the posterior given what was already known, and the new posterior including the new batch. The temperatures $\lambda_1, \dots \lambda_k$ can be tuned on the fly, depending on how novel the new batch is.

Evaluating the likelihood $\mathcal{L}(\theta, \theta', \mathcal{B})$ only requires the latest batch, and can easily be computed exactly. However, we choose to approximate $\pi(\theta|\theta', \mathcal{D})$ 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 naively changes the target distribution, meaning that the weights $w_1, \dots w_n$ are incorrect, and the sample

$$\theta_1, \dots, \theta_n, \;\; w_1, \dots, w_n$$

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

$$\pi(\theta)\mathcal{L}(\theta, \theta'_{\texttt{old}})^{(1-\lambda)}\pi(\theta)\mathcal{L}(\theta, \theta'_{\texttt{new}})^{\lambda} \rightarrow \pi(\theta)\mathcal{L}(\theta, \theta'_{\texttt{new}}),$$

as $\lambda \rightarrow 1$. 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 $\theta_1, \dots, \theta_n$ to immediately match the new targets.

Apart from the novel parameter updates and target network updates, Sequential Monte Carlo DQN (SQL DQN) makes no further alterations to BootDQN. Algorithm 3 shows the general structure with both updates, where `SMC` refers to Algorithm 2. The symplectic euler Langevin scheme [Wenzel et al., 2020] is used as MCMC step. While the structure is similar to a typical DQN implementation, a major difference is that the SMC steps take a lot more time than typical Q-value updates. For each batch from the environment, the agent fully 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. 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. It is assumed that the replay buffer $\mathcal{D}$ is large enough to store every experienced transition.

Algorithm 3: Sequential DQN

Input: MDP, batch size B, ensemble size n
Result: Posterior over $Q_\theta(s, a)$
$\theta_1, \dots, \theta_n \sim \pi(\theta)$; `/* Initialize networks */`
$w_1, \dots, w_n \leftarrow 1$;
while *training* **do**
  $i \sim \texttt{dirichlet}(w_1, \dots w_n)$;
  $s_0 \sim \texttt{MDP}$;
  $t \leftarrow 0$;
  while *episode not done* **do**
    $a_t = \text{argmax}_a \, Q_{\theta_i}(s_t)$; `/* TS sample for full episode */`
    $r_t, s_{t+1} \sim \texttt{MDP}(s_t, a_t)$;
    $\mathcal{B} = \mathcal{B} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
    if $|\mathcal{B}| = B$ **then**
      $\theta_1, \dots, \theta_n, w_1, \dots, w_n \leftarrow \texttt{SMC}(\pi(\theta|\theta', \mathcal{D}), \pi(\theta|\theta', \mathcal{D} \cup \mathcal{B}))$; `/* main update */`
      $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{B}$;
      $\mathcal{B} \leftarrow \emptyset$
    end
    if *time for target update* **then**
      $\theta'_{i,\texttt{new}} \leftarrow \theta_i$;
      $\theta_1, \dots, \theta_n, w_1, \dots, w_n \leftarrow \texttt{SMC}(\pi(\theta|\theta', \mathcal{D}), \pi(\theta|\theta'_{\texttt{new}}, \mathcal{D}))$; `/* target update */`
      $\theta'_i \leftarrow \theta'_{i,\texttt{new}}$
    end
  end
end

5 Experiments

We evaluate our agent on the 30×30 Deep Sea environment [Osband et al., 2016, 2020]. This environment is a grid, where the agent starts in the top left corner and at each time step, the agent

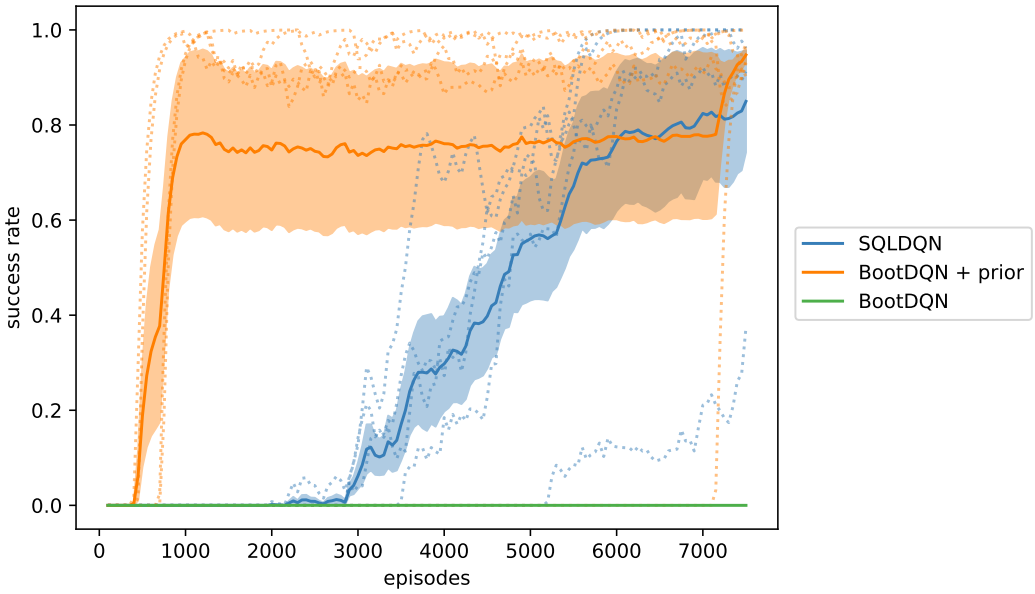

Figure 3: The fraction of runs where the agent reaches the goal state at each episode, averaged over 50 episodes. The shaded area shows the standard error of the mean over 5 seeds. Dotted lines show individual seeds. The agent with an ensemble of DQNs can not solve this environment.

drops down one row and either moves left or right. The agent receives small negative reward when moving to the right, except when reaching the bottom right state, where the agent obtains reward 1. The optimal policy is to move right along the diagonal to reach this goal state, and a single mistake in an episode will cause the agent to not receive any reward. This environment requires a dedicated exploration method, as it would take a randomly exploring agent on average 2^30 episodes to observe a positive reward.

Because updating the models and targets is more computationally expensive than in regular BootDQN, the agent uses a relatively large batch size of 1024, meaning that the models are only updated approximately every 34 episodes. Furthermore, because the target update step also updates the main models alongside the targets, the target networks can be updated after every main update without causing instability. This allows SQLDQN to adapt to new trajectories relatively quickly. To further increase the diversity of collected trajectories, the active Q-network is sampled uniformly from the particles instead of from the distribution, since during early testing the effective sample size after target updates appeared to be quite low, even though the uncertainty qualitatively looked correct. Furthermore, no resampling is performed during SMC in order to maintain diversity in the target networks.

Figure 3 shows the average success rate, i.e., the number of times the agent reaches the goal state, during training. Crucially, this is the success rate of the Thompson sampling policy which still explores up until all ensemble members agree on the correct path. It can be seen that it takes a long time for the entire ensemble to agree on the correct action sequence. Even after observing the goal, it takes several thousands more episodes to fully converge to the optimal policy. This could be improved upon by switching to a more optimistic evaluation, likelihood or prior, but not without assuming further information about the environment. For example, in this experiment the agent is not aware that the rewards and trajectories are in fact deterministic given the action sequence. In figure 4 this problem is very apparent: even after convergence at 200000 episodes, the value is still reasonably high for early but non-optimal states. This is because the aleatoric uncertainty in the likelihood is compounded from the end to the start of the trajectory, and the argmax in the targets biases the value upwards. BootDQN with prior functions does not suffer from this as much as it simply minimizes the mean squared error without taking a probabilistic point of view. It can be seen clearly in Figure 3 that a well-tuned variant of BootDQN with prior functions outperforms SQL DQN significantly in this

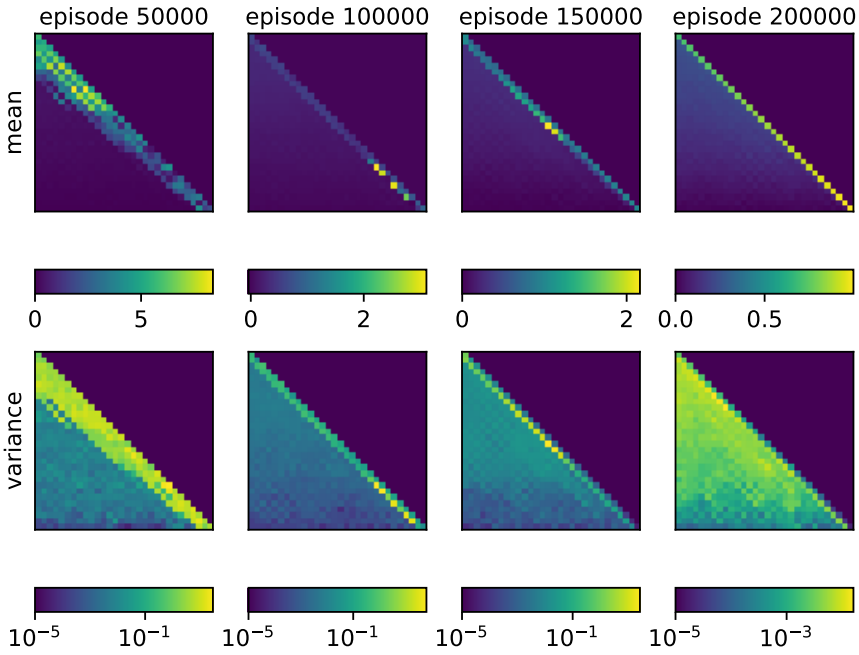

Figure 4: Graphical view the mean and variance of the values during training on Deep sea. Each pixel shows the mean (top row) or variance (bottom row) of that state as predicted by the ensemble. The top left state is the initial state, and the bottom right state is the goal state. The diagonal is the only sequence of states that results in reward. The states in the upper triangle are unreachable.

scenario. Taking this probabilistic point of view as in SQL DQN may be an advantage in stochastic scenarios, but we leave further comparisons on these environments for future work.

Qualitatively speaking, Figure 4 shows what should be expected of a posterior. The variance decreases as states are explored more often, and the variance decreases faster for states that are closer to the end of the environment. Furthermore, the values are learned bottom up, and the ensemble stays optimistic about states that have not been visited often. This allows the agent to eventually solve the environment, albeit at more steps than BootDQN with prior functions. At episode 200 000, the mean value prediction is close to γ^n for $n = 29, \ldots, 0$ along the diagonal, which is the true value, and the variance is low for every state.

6 Conclusion

This paper studied the novel idea of how sequential Monte Carlo can be used to train an ensemble to approximate the Bayesian posterior distribution. 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. While the algorithm developed here can be motivated from a theoretical standpoint, our findings are that deviating from the theoretically-sound algorithm by not performing resampling and uniformly sampling particles for action selection *improves* performance in reinforcement learning. This could be caused by the likelihood or prior not describing the environment well, especially in deterministic environments such as Deep Sea. Investigating these issues is left for future work.

References

M. Cai, M. Del Negro, E. Herbst, E. Matlin, R. Sarfati, and F. Schorfheide. Online estimation of dsge models. *The Econometrics Journal*, 24(1):C33–C58, 2021.

T. Chen, E. Fox, and C. Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691. PMLR, 2014.

F. D'Angelo and V. Fortuin. Repulsive deep ensembles are bayesian, 2021.

H.-D. Dau and N. Chopin. Waste-free sequential monte carlo. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(1):114–148, 2022.

P. Del Moral, A. Doucet, and A. Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006.

M. Fellows, K. Hartikainen, and S. Whiteson. Bayesian bellman operators. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

A. Garriga-Alonso and V. Fortuin. Exact langevin dynamics with stochastic gradients. *arXiv preprint arXiv:2102.01691*, 2021.

M. Geiger, A. Jacot, S. Spigler, F. Gabriel, L. Sagun, S. d'Ascoli, G. Biroli, C. Hongler, and M. Wyart. Scaling description of generalization with number of parameters in deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(2):023401, 2020.

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

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Q. Liu and D. Wang. Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. *arXiv e-prints*, art. arXiv:1608.04471, Aug. 2016.

F. Llorente, L. Martino, J. Read, and D. Delgado-Gómez. Optimality in noisy importance sampling. *Signal Processing*, 194:108455, 2022.

O. Lockwood and M. Si. A review of uncertainty for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, pages 155–162, 2022.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015. ISSN 00280836. URL `http://dx.doi.org/10.1038/nature14236`.

R. M. Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11): 2, 2011.

I. Osband, C. Blundell, A. Pritzel, and B. V. Roy. Deep exploration via bootstrapped dqn, 2016.

I. Osband, J. Aslanides, and A. Cassirer. Randomized prior functions for deep reinforcement learning, 2018.

I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvari, S. Singh, B. V. Roy, R. Sutton, D. Silver, and H. V. Hasselt. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020.

S. Schmitt, J. Shawe-Taylor, and H. van Hasselt. Exploration via epistemic value estimation, 2023.

F. Wenzel, K. Roth, B. Veeling, J. Swiatkowski, L. Tran, S. Mandt, J. Snoek, T. Salimans, R. Jenatton, and S. Nowozin. How good is the bayes posterior in deep neural networks really? In *International Conference on Machine Learning*, pages 10248–10259. PMLR, 2020.

Appendices

A Implementation details

A.1 Supervised learning

The neural network is a fully connected network with two hidden layers of size 100 and 10. The data is generated by sampling x i.i.d. from a uniform mixture of three normal distributions with means -1, 0, and 1 and standard deviation 0.25. The labels y are generated by randomly sampling a neural network $\theta_{\texttt{prior}}$ from the prior and setting $y_i = f_{\theta_{\texttt{prior}}}(x_i) + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, 0.25)$.

The hyperparameters for SMC are shown in Table 1, where the batch size is varied for the results shown in Figure 2. For randomized prior functions, the prior function was drawn from the same prior as used for SMC and HMC.

Sequential Monte Carlo hyperparameters	
$\pi(\theta)$ (prior)	i.i.d. $\mathcal{N}(0, 1)$
σ (likelihood std)	0.1
B (batch size)	32
MCMC steps (main)	10
MCMC steps (target)	10
Symplectic Euler Langevin Scheme hyperparameters	
ℓ (learning rate)	10^{-3}
cycle length	20
β	0.98
h (step size)	$\sqrt{\frac{\ell}{\texttt{n}_{\texttt{data}}}}$
γ	$\frac{1}{h}$

Table 1: Hyperparameters for SMC in the supervised learning experiments shown in Figures 1 and 2.

A.2 Deep Sea

The environment is the `deepsea/10` environment as implemented in BSuite [Osband et al., 2020], which is 30×30 and uses a random action mapping.

The Q-value network is the same as the baseline BootDQN agent in BSuite, which is fully connected with two hidden layers of size 50. Action selection is also unchanged and done by picking one ensemble member uniformly at random, and acting greedily with respect to that network for the rest of the episode.

The hyperparameters used are shown in Table 2.

The hyperparameters for BootDQN + prior and BootDQN are the default hyperparameters in BSuite.

Sequential Monte Carlo hyperparameters	
$\pi(\theta)$ (prior)	i.i.d. $\mathcal{N}(0,1)$
σ (likelihood std)	0.1
B (batch size)	1024
MCMC steps (main)	100
MCMC steps (target)	100
Reinforcement Learning hyperparameters	
Total episodes	7500
Buffer size	∞
Main update frequency	B
Target update frequency	B
Symplectic Euler Langevin Scheme hyperparameters	
ℓ (learning rate)	10^{-3}
cycle length	20
β	0.98
h (step size)	$\sqrt{\frac{\ell}{\mathtt{n}_{\mathtt{data}}}}$
γ	$\frac{1}{h}$

Table 2: Hyper-parameters of the Deep Sea environment experiments.