Bayesian Exploration in Deep Reinforcement Learning

Ludvig Killingberg[1], Helge Langseth[1]

[1]Norwegian University of Science and Technology, Høgskoleringen 1, 7034 Trondheim, Norway

Abstract

Posterior sampling of value functions can give efficient exploration for value-based reinforcement learning algorithms. We introduce BayesianExplore (BE), a posterior sampling-based method for reinforcement learning based on Bayesian function-space variational inference over stochastic processes. This Bayesian formalism allows us to formalize domain knowledge as a prior over the value function. Our approach, therefore, provides an alternative to reward shaping with the added benefit that the algorithm keeps seeking an optimal policy in the original environment instead of the one with altered rewards. We show that BE produces state-of-the-art efficiency in exploration with flat priors, and that it is easy to significantly improve performance by incorporating domain knowledge using simple priors.

Keywords

Bayesian deep learning, reinforcement learning

1. Introduction

A reinforcement learning environment is modelled as a Markov decision process (MDP) $M = \langle \mathcal{S}, \mathcal{A}, R, P, P_0, \gamma \rangle$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the set of available actions. At time $t = 0$ a state $\boldsymbol{s}_0$ is sampled from the distribution $P_0(\cdot)$. At each time-step an action $a_t \sim \pi(\cdot|\boldsymbol{s}_t)$ is selected and the agent transitions to a new state $\boldsymbol{s}_{t+1} \sim P(\cdot|\boldsymbol{s}_t, a_t)$. A scalar reward $r_t \sim R(\boldsymbol{s}_t, a_t)$ is observed. As the agent and environment interact in a sequence of time steps, a history of observations $\mathcal{H}_t = (\boldsymbol{s}_0, a_0, r_0, \boldsymbol{s}_1, a_1, r_1, \ldots, \boldsymbol{s}_t, a_t, r_t)$ is collected. The goal is to find a policy $\pi^\star$, such that sampling actions $a \sim \pi^*(\cdot|\boldsymbol{s})$ maximises the expected accumulated and discounted future reward, $J^\pi := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, where the expectation is taken over the policy, transition, and reward distributions. An efficient agent must be able to learn from the data it collects, but since the data is dependent on the policy, it must also prioritize exploring states and actions that the agent can learn from.

Related to J^π is the Q-function, $Q^\pi(\boldsymbol{s}_0, a_0)$, defined as the expected reward of taking action a_0 in state $\boldsymbol{s}_0$ and then following policy π thereafter: $Q^\pi(\boldsymbol{s}_0, a_0) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | \boldsymbol{S}_0 = \boldsymbol{s}_0, A_0 = a\right]$. Q-learning amounts to learning the Q-function from $\mathcal{H}_t$. The regret of a policy π is $J^{\pi^*} - J^\pi$, the loss in the expected reward obtained by following π instead of following the optimal policy π^*. Now, a learning algorithm's efficiency in exploration can be measured by its cumulative regret over time.

NAIS 2023: The 2023 symposium of the Norwegian AI Society, June 14-15, 2023, Bergen, Norway.
ludvig.killingberg@ntnu.no (L. Killingberg); helge.langseth@ntnu.no (H. Langseth)

One exploration strategy often employed in Q-learning algorithms to date is the ϵ-greedy approach, where one chooses $a^* = \arg\max_{a \in \mathcal{A}} Q^\pi(\boldsymbol{s}, a)$ with probability $1 - \epsilon$ or selects a uniformly from $\mathcal{A}$ with probability ϵ. While ϵ-greedy exploration ensures exploration of the domain, its regret bound grows linearly with time, and is therefore provably inefficient.

A very simple test-bed for exploration strategies in reinforcement learning is the multi-armed bandit problem. The state is void in this problem formulation, rendering $\mathcal{S}$, P, P_0 vacuous and r_t a function only of a_t. There are several (asymptotically) optimal algorithms for this problem, and one of the simplest ones is Thompson sampling [1]. Thompson sampling approximates a posterior distribution of the mean reward for each action. The next action a_t is decided by sampling rewards for each action from these posterior distributions and selecting the action that gave the highest sampled reward. Posterior sampling methods have also been shown to behave efficiently with respect to cumulative regret [2] on general MDPs.

Fortunato et al. [3] introduced NoisyNet as a means for balancing exploration and exploitation. They used a neural network to represent the Q-function and extended their model with stochastic weights such that each weight $w_{ij}^{(l)}$ has an added perturbation sampled from a noise distribution with standard deviation $\sigma_{ij}^{(l)}$. After initializing $\boldsymbol{w}$ and $\boldsymbol{\sigma}$ such that the network has sufficient stochasticity for exploration, both parameters are learned using standard backpropagation. Fortunato et al. [3] apply NoisyNet to three reinforcement learning algorithms: DQN, Dueling DQN, and A3C, and show improved performance on all of them. Later, NoisyNet was used in the Rainbow algorithm [4], a combination of six extensions to the DQN algorithms [3, 5, 6, 7, 8, 9], that shows state-of-the-art performance across 57 Atari games.

A limitation of NoisyNet is that the initial uncertainty in the value or policy function is crucial for exploration. If the uncertainty is too high, the algorithm will struggle to learn anything, while if the uncertainty is too low, there is nothing incentivizing exploration and the algorithm will not explore new trajectories. The learning approach in NoisyNet is similar to variational inference schemes such as Bayes by backprop [10], where the weights of a neural network model are assumed to be normally distributed with mean $\mu_{ij}^{(l)}$ and standard deviation $\sigma_{ij}^{(l)}$. However, while the objective of Bayes by backprop is to approximate $p(\boldsymbol{w}|\mathcal{D})$, the posterior distribution over the weights after seeing a fixed dataset $\mathcal{D}$, the weights in NoisyNet are not given a prior distribution, and the learning, therefore, does not result in a posterior distribution over $\boldsymbol{w}$. Consequently, NoisyNet does not have the same optimality guarantee on total regret as methods that approximate a posterior over the value functions [2], and the exploration could stop prematurely if the standard deviations $\boldsymbol{\sigma}$ decline too quickly during learning. In posterior sampling-based methods, we can fit the initial uncertainty to a prior distribution. This would mean that with an appropriate prior, network parameter initialization is not as critical to performance. Another advantage of posterior sampling is that it can provide a natural way to incorporate domain knowledge. Prior knowledge can be used to create informative prior distributions for value or policy functions.

In this paper, we will therefore introduce BayesianExplore (BE), a fully Bayesian extension of NoisyNet. The key idea is to use a Bayesian deep network to represent the posterior distribution over $Q(\boldsymbol{s}, a)$ given the history $\mathcal{H}_t$, and thereafter use Thompson sampling as a means to efficiently balance exploration and exploitation. To allow prior knowledge to be efficiently encoded, we use *function-space variational inference*, meaning that the model does not learn the posterior distribution over the parameters, but rather the posterior process over the output of the model. BE relates to Q-learning in this paper, but we note that the key idea would also apply to policy-based methods.

We summarize our contributions as follows:

- We introduce BayesianExplore, a fully Bayesian Q-learning method for reinforcement learning;
- We give initial results comparing BE to NoisyNet, showing competitive results;
- We show how simple heuristics can be efficiently encoded as functional priors;
- We show that these priors can significantly improve learning efficiency.

2. Background

Before we delve into the background, we need to define some notation. Most of the theory will be based on stochastic processes. The stochastic process we are interested in generates value functions for an MDP and exists on the associated probability space. It can be written as $\{Q(\boldsymbol{s}, \boldsymbol{a}) : (\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{S} \times \mathcal{A}\}$. For any sample $\omega \in \Omega$, $Q(\cdot, \cdot, \omega)$ is a sample function mapping $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$. To simplify notation in the following subsections we will denote $(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{S} \times \mathcal{A}$ by $\boldsymbol{x} \in \mathcal{X}$, the sample functions as $f : \mathcal{X} \to \mathbb{R}$, and the associated process as $\mathcal{F}$. We will use $\mathbf{f}_{1:N}$ for a collection of N sample-functions. Next, $\mathbf{f}(\boldsymbol{x})$ and $\{\mathbf{f}(\boldsymbol{x}), \boldsymbol{x} \in \mathbf{X}\}$ denote the process evaluated at single point $\boldsymbol{x}$ and the set of points $\{\boldsymbol{x} \in \mathbf{X}\}$, respectively. The marginal process at the set $\mathbf{X} = \{\boldsymbol{x}_i\}_{i=1}^n \in \mathcal{X}^n$ is with a slight abuse of notation denoted by $\mathcal{F}(\mathbf{X})$, and we evaluate a likelihood using the notation $P(\boldsymbol{y}|\mathcal{F}(\mathbf{X}))$. For instance, if $\mathcal{F}$ is a Gaussian process (GP) with mean $\mu(\boldsymbol{x})$ and covariance $k(\boldsymbol{x}, \boldsymbol{x}')$, $\mathbf{f}_{1:N}$ is a collection of N realizations from that Gaussian process, $\mathcal{F}(\mathbf{X})$ is the multivariate Gaussian obtained at $\mathbf{X}$ with associated parameters, $\mathbf{f}(\boldsymbol{x})$ is a univariate Gaussian with given mean and standard deviation, and $P(\boldsymbol{y}|\mathcal{F}(\mathbf{X}))$ evaluates the likelihood of $\boldsymbol{y}$ under the Gaussian jointly defined by $\mathcal{F}(\mathbf{X})$.

2.1. Noisy Networks

NoisyNet can be viewed as a stochastic process represented by a neural network with stochastic weights. We will define its sampling distribution as $f \sim \rho_{\boldsymbol{\phi}}$, where a sample function f now realizes a neural network to represent the Q-function $Q(\boldsymbol{s}, a)$. This means that NoisyNet can model stochastic policies, and Fortunato et al. [3] show through empirical analysis that the NoisyNet policy sometimes converges to a non-deterministic policy. Nevertheless, they also point out that there always exists a deterministic optimal policy for the mean squared error loss in DQN. Deep neural networks used as function approximations in Q-learning were labeled DQN by Mnih et al. [11]. Later, Mnih et al. [12] made a significant improvement to the learning stability of their original DQNs by

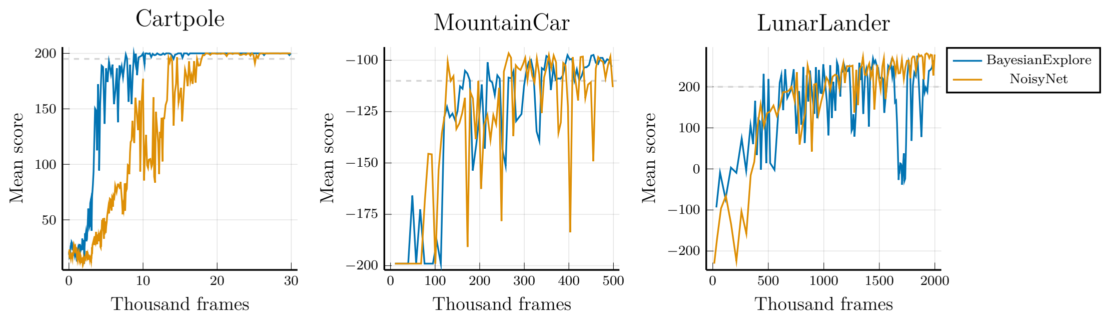

Figure 1: Comparison of learning curves of BE (with flat prior) and NoisyNet-DQN on Cartpole, MountainCar, and LunarLander with score averaged over 100 episodes. Performance is evaluated every episode for Cartpole, every 10th episode for MountainCar, and every 50th episode for LunarLander.

introducing a target network. The target network has the same structure as the regular network, and the weights are copied over from the regular network every T^{-} timesteps. The target network is used to calculate the target Q-value for the temporal difference (TD) error. Having a more stationary target is shown to improve the stability of training. This was a substantial improvement, as training the DQN was previously unstable.

2.2. Functional Variational Bayesian Neural Networks

Consider a supervised learning problem, where we desire a parameterized function $g_{\boldsymbol{w}} : \mathcal{X} \to \mathcal{Y}$ to map an input $\boldsymbol{x} \in \mathcal{X}$ to a target $y \in \mathcal{Y}$. If we train a Bayesian neural network with stochastic weights $\boldsymbol{w}$ to represent $g_{\boldsymbol{w}}$, the standard procedure is to assume that the dataset $\mathcal{D}$ with datapoints $(\boldsymbol{x}, \boldsymbol{y})$ is given, and proceed by defining a prior distribution $p(\boldsymbol{w})$ over the weights [13, 10, 14, 15]. After defining $p(\boldsymbol{w})$ we can use variational inference methods to approximate $p(\boldsymbol{w}|\mathcal{D}) = p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})/p(\mathcal{D})$ and use that to realize $g_{\boldsymbol{w}}$. The disadvantage of this approach is that prior knowledge we might have about the domain typically relates to the behaviour of the function $g_{\boldsymbol{w}}(\boldsymbol{x})$, which is very difficult to encode at the level of the individual weights in $\boldsymbol{w}$. Consequently, the prior $p(\boldsymbol{w})$ will in essence only act as a regularizer, and is not a suitable medium for incorporating informative a priori knowledge.

Functional variational Bayesian neural networks [16] is a variational inference method for neural networks that approximates the posterior distribution in function space. This means that our prior will be a distribution over functions, i.e., a stochastic process, and as part of the evaluation of the evidence lower bound (ELBO) we will need to calculate the KL-divergence from one process to another. Sun et al. [16] show that for two stochastic processes $\mathcal{P}$ and $\mathcal{Q}$, the KL-divergence from $\mathcal{P}$ to $\mathcal{Q}$ is the supremum of the marginal KL-divergences over all finite measurement sets. Let $\mathcal{P}(\mathbf{X})$ (resp. $\mathcal{Q}(\mathbf{X})$) be the marginal distribution of function values from the process $\mathcal{P}$ (resp $\mathcal{Q}$) at some set of points $\mathbf{X} \in \mathcal{X}^n$, then:

$$\mathrm{KL}[\mathcal{P}\|\mathcal{Q}] = \sup_{n\in\mathbb{N}, \mathbf{X}\in\mathcal{X}^n} \mathrm{KL}\left[\mathcal{P}(\mathbf{X})\|\mathcal{Q}(\mathbf{X})\right], \tag{1}$$

Note that as the KL-divergence between two processes is a supremum over the marginals on the right-hand side of Equation 1, it holds for any given $\mathbf{X}$ that $\mathrm{KL}[\mathcal{P}\|\mathcal{Q}] \geq \mathrm{KL}\left[\mathcal{P}(\mathbf{X})\|\mathcal{Q}(\mathbf{X})\right]$. In the following, we will nevertheless approximate the functional KL-divergence by using finite measurement sets $\mathbf{X} \in \mathcal{X}^n$, acknowledging the fact that we may underestimate the true KL-divergence between the two stochastic processes. From now on we will therefore be talking about the KL-divergence between marginal distributions of function values instead of between processes.

Now, let the stochastic processes be represented by a neural network f. A priori we will assume $f \sim p$, and use variational inference to find the posterior process $f \sim \rho_{\boldsymbol{\phi}}$ which is parameterized by $\boldsymbol{\phi}$. We will think of the generative process $\rho_{\boldsymbol{\phi}}$ as follows: We sample a vector $\boldsymbol{\xi}$ from, e.g., a standard Gaussian and populate a neural network with weights $w_i = \mu_i + \sigma_i \cdot \xi_i$, where $\boldsymbol{\phi} = (\boldsymbol{\mu}, \boldsymbol{\sigma})$ is the collection of parameters required to define the sample function.

In our notation, Sun et al. [16] showed that the gradient of the KL-divergence for functions marginalized at the measurement set $\mathbf{X}$ is

$$\begin{aligned}\nabla_{\boldsymbol{\phi}}\mathrm{KL}\left[\rho_{\boldsymbol{\phi}}(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}})\|p(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}})\right] = \\ \mathbb{E}_{\xi}\big[\nabla_{\boldsymbol{\phi}}\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}}(\nabla_{\mathbf{f}}\log\rho_{\boldsymbol{\phi}}(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}}) \\ - \nabla_{\mathbf{f}}\log p(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}}))\big].\end{aligned} \tag{2}$$

Here we have used that the expected value of the score function is zero. The difficult part in Equation (2) is to estimate $\nabla_{\mathbf{f}}\log\rho_{\boldsymbol{\phi}}(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}})$ and $\nabla_{\mathbf{f}}\log p\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}})$. The entropy derivative $\nabla_{\mathbf{f}}\log\rho_{\boldsymbol{\phi}}(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}})$ is generally intractable. Depending on how we define the prior, however, $\nabla_{\mathbf{f}}\log p(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}})$ can be easy to compute analytically. To reduce variance in the gradients, we use tractable priors in this paper.

To estimate the gradient of the log-density under $\rho_{\boldsymbol{\phi}}$, $\nabla_{\mathbf{f}}\log\rho_{\boldsymbol{\phi}}(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}})$, Sun et al. [16] use the Spectral Stein Gradient Estimator (SSGE) [17]. Shi et al. [17] show that, for a differentiable density ρ and positive definite kernel $k(\cdot,\cdot)$ in the Stein class of k, we can approximate the gradient $\nabla_{\mathbf{f}}\log\rho(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}})$. Given N samples from ρ, the Nyström approximation [18, 19] is used to calculate the first J eigenfunctions of k, $\hat{\psi}_1, \ldots, \hat{\psi}_k$. It follows that

$$\nabla_{\mathbf{f}}\log\rho(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}}) \approx \sum_{j=1}^{J}\nabla_{\mathbf{f}}\,\hat{\beta}_{ij}\hat{\psi}_j(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}}),$$

where

$$\hat{\beta}_{ij} = -\frac{1}{N}\sum_{n=1}^{N}\nabla_{\mathbf{f}}\,\hat{\psi}_j(\mathbf{f}_n).$$

In short, this is a method for estimating the gradient function of implicit distributions using approximations to eigenfunctions of a kernel-based operator. We will follow Shi et al. [17] and use the RBF kernel in all experiments. This brings three hyper-parameters to the algorithm: the number of samples N from the implicit distribution used to approximate the gradient, the number of eigenvectors J used to approximate the gradient, and η, a regularisation parameter that smooths the gradient function.

The full objective for the functional Bayesian neural network becomes

$$\mathcal{E} = \frac{1}{|\mathcal{D}_s|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}_s} \log P(y|\mathbf{f}(\boldsymbol{x})) - \lambda \cdot \mathrm{KL}\left[\rho_{\boldsymbol{\phi}}\left(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}}\right) \| p\left(\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}}\right)\right], \tag{3}$$

where we use $P(y|\mathbf{f}(\boldsymbol{x}))$ to denote the likelihood of the observation y under the stochastic process evaluated at $\boldsymbol{x}$ (e.g., $\mathbf{f}(\boldsymbol{x})$ could be a univariate Gaussian with given mean and standard deviation). Note here that we approximate $\log P(y|\mathbf{f}(\boldsymbol{x}))$ in the implementation using Monte Carlo sampling: We generate $\mathbf{f}_{1:N}(\boldsymbol{x})$ with $f \sim \rho_{\boldsymbol{\phi}}$, use these to approximate the local model $\mathbf{f}(\boldsymbol{x})$, and thereby also approximate the log-likelihood of the observation y under the generative process $\rho_{\boldsymbol{\phi}}$.

In order for $\mathcal{E}$ in Equation (3) to match the functional ELBO and be a proper lower bound for $\log p(\mathcal{D})$, λ should be set to $\lambda = \frac{1}{|\mathcal{D}|}$. Sun et al. [16] note, however, that $\mathcal{E}$ uses a lower bound for the true KL divergence between the processes, so a larger value for λ is likely necessary to maintain properly calibrated posterior uncertainty. They, therefore, use one over the batch size instead, $\lambda = \frac{1}{|\mathcal{D}_s|}$, a strategy that we will also follow.

3. Method

Osband and Van Roy [20] prove that posterior sampling of Q-values for reinforcement learning in finite horizon MDPs has at least a near-optimal regret bound. They further conjecture that their bound can be improved to show optimal regret. Additionally, a posterior sampling-based reinforcement learning algorithm can be made to utilize domain knowledge through an appropriate prior. This should improve the policy convergence rate. Combined with the computational efficiency of posterior sampling, this motivates the development of a Bayesian reinforcement learning algorithm with functional priors.

We will present a method based on functional variational Bayesian neural networks [16] that allows efficient exploration, and the inclusion of domain knowledge.

This can be achieved by modeling posterior distributions either over the policy function or a value function, then sample an action directly from that posterior (in case of policy focus) or use greedy action-selection based on samples from the value-function posterior. In this paper, we will approximate the posterior distribution of the Q-value function using DQN [12].

We use the functional variational Bayesian neural network (FVBNN) [16] framework discussed previously and compare our approach to NoisyNet. Note that when NoisyNet uses one sample from Q for each optimization step we will instead use N samples from Q. This is needed to use the FVBNN loss defined in Equation (3) rather than the temporal difference used in NoisyNet. The Bayesian formalism allows us to incorporate a priori domain knowledge through the prior p, and will encourage exploration with (close to) optimal regret [20].

Recall that the learning objective in Equation (3) requires the evaluation of the marginal $\{\mathbf{f}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbf{X}}$. Here, the measurement set $\mathbf{X}$ should contain representative samples from $\mathcal{X}$. In our setting $\boldsymbol{x}$ will be state-action pairs, and $\mathcal{X} = \mathcal{S} \times \mathcal{A}$. The data-set $\mathcal{D}^n_{\boldsymbol{\omega}}$ consists of n examples of state-action pairs combined with the predicted Q-value,

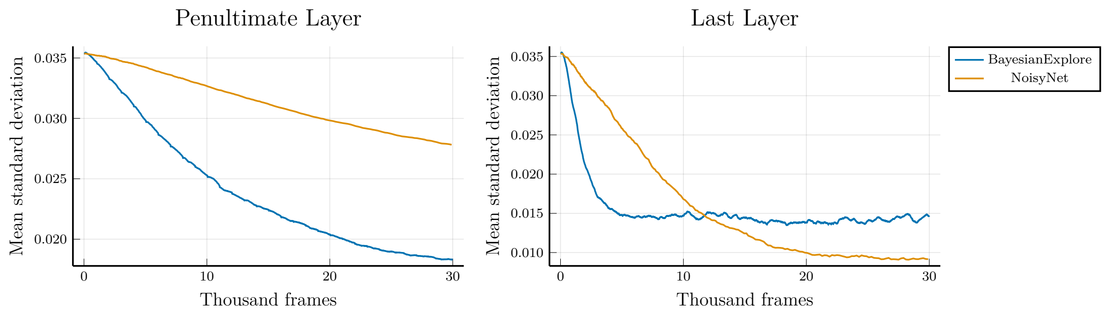

Figure 2: Comparison of mean standard deviation for NoisyNet and BE on Cartpole.

$\mathcal{D}^n_{\boldsymbol{\omega}} = \{(\boldsymbol{s}_i, a_i, q_i)\}^n_{i=1}$; the subscript $\boldsymbol{\omega}$ is used to denote the version of the target net used to generate the q_i-values. In the following the set $\mathbf{X}$ is defined as the set of all state-action pairs for states we have already explored:

$$\mathbf{X} = \{(\boldsymbol{s}_i, a_j) \mid \forall \boldsymbol{s}_i \in \mathcal{D}^n_{\boldsymbol{\omega}}, \forall a_j \in \mathcal{A}\}.$$

$\mathbf{X}$ will eventually have full support in $\mathcal{S} \times \mathcal{A}$ if the MDP is ergodic.

To calculate $\log P(q_i|\mathbf{f}(\boldsymbol{s}_i, a_i))$ we will assume that the underlying stochastic process is a Gaussian process, $\mathcal{F} \sim \mathrm{GP}$. The network architecture for f is defined to produce two output vectors: the mean $\boldsymbol{\mu}$ and the log standard deviation $\boldsymbol{\tau} = \log \boldsymbol{\sigma}$.

This gives the following loss function when evaluated on a sub-sample $\mathcal{D}_s$:

$$\mathcal{L} = \lambda \cdot \mathrm{KL}[\rho_{\boldsymbol{\phi}} \| p] + \frac{1}{|\mathcal{D}_s|} \sum_{i=1}^{|\mathcal{D}_s|} \tau_i + (\mu_i - q_i)^2 \exp(-\tau_i).$$

Algorithm 1 shows a general DQN-update iteration shared by both NoisyNet and BE where the agent interacts with the environment. The difference between NoisyNet and BE is how the neural network is updated when Algorithm 1 makes the call to UPDATENET in line 10. Algorithm 2 and Algorithm 3 provide two different definitions for the UPDATENET function. Algorithm 2 details the procedure for updating NoisyNet. It begins by sampling one value function f and one target function f'. The target function is used to calculate the target-value for the value function in line 8. After that, the network is updated by minimising the temporal difference error. Algorithm 3 shows the pseudo-code for BE. The first difference to NoisyNet is that the network has two outputs for each action, the mean q_μ and standard deviation q_σ. Note that we use subscript μ when we are only interested in the mean (line 7 and 9) and no subscript when we fetch both results (line 8). The next difference is that we sample N functions from the network (line 3) and target network (line 4). Instead of the temporal difference error, a Gaussian log-likelihood loss is used instead (line 10). The gradient KL-loss term is calculated using the spectral Stein gradient estimator as outlined above (call to SSGE in line 13).

Algorithm 1 DQN-update

```
 1: $\mathcal{D} \leftarrow \emptyset$
 2: $\boldsymbol{s} \sim P_0$
 3: Initialise $\boldsymbol{\phi}, \boldsymbol{\omega}$
 4: **for** $t \in \{1, \dots\}$ **do**
 5:     $f \sim \rho_{\boldsymbol{\phi}}$
 6:     $a \sim \arg\max_a f(\boldsymbol{s}, a)$
 7:     $\boldsymbol{s}' \sim P(\cdot|\boldsymbol{s}, a)$
 8:     $r \sim R(\boldsymbol{s}', a, \boldsymbol{s})$
 9:     $\mathcal{D} \leftarrow \mathcal{D} \cup \{\boldsymbol{s}, a, r, \boldsymbol{s}'\}$
10:     UpdateNet($\boldsymbol{\phi}, \boldsymbol{\omega}, \mathcal{D}$)          ▷ Update value network
11:     **if** $t \mod t^- \equiv 0$ **then**
12:         $\boldsymbol{\omega} \leftarrow \boldsymbol{\phi}$                      ▷ Update target network
13:     **end if**
14: **end for**
```

Algorithm 2 NoisyNet Update

```
 1: **function** UpdateNet($\boldsymbol{\phi}, \boldsymbol{\omega}, \mathcal{D}$)
 2:     $\mathcal{D}_s \sim \mathcal{D}$
 3:     $\Delta_{\mathcal{L}} \leftarrow 0$
 4:     $f \sim \rho_{\boldsymbol{\phi}}$
 5:     $f' \sim \rho_{\boldsymbol{\omega}}$
 6:     **for** $\{\boldsymbol{s}, a, r, \boldsymbol{s}'\} \in \mathcal{D}_s$ **do**
 7:         $q \leftarrow \max_a f(\boldsymbol{s}, a)$
 8:         $G \leftarrow r + \gamma \cdot f'(\boldsymbol{s}', a)$
 9:         $\Delta_{\mathcal{L}} \leftarrow \Delta_{\mathcal{L}} - \nabla_{\boldsymbol{\phi}}(q - G)^2$
10:     **end for**
11:     $\boldsymbol{\phi} \leftarrow$ Optimizer($\boldsymbol{\phi}, \Delta_{\mathcal{L}}$)
12: **end function**
```

4. Experiments

Our method most closely resembles NoisyNet [3], so all experiments will mainly compare the performance of BE to that baseline.

4.1. Details and Hyper-parameters

We compare our method with NoisyNet [3] on three different environments in the OpenAI Gym [21]: Cartpole, MountainCar, and LunarLander. Whenever possible we will use the hyper-parameters employed by Han et al. [22]. Additionally, our method has hyper-parameters related to the calculation of the functional KL-divergence. These are reported in Table 1. Note the relatively low number of eigenvectors J and relatively high value for η. Both were chosen to smooth the gradient estimates. Note also that we have used the

Algorithm 3 Functional Bayesian Update

```
1: function UpdateNet($\boldsymbol{\phi}, \boldsymbol{\omega}, \mathcal{D}$)
2:     $\mathcal{D}_s \sim \mathcal{D}$
3:     $f_i \sim \rho_{\boldsymbol{\phi}} \quad i = 1, \ldots, N$
4:     $f'_i \sim \rho_{\boldsymbol{\omega}} \quad i = 1, \ldots, N$
5:     $\Delta_{\mathcal{L}} \leftarrow 0$
6:     for $\{\boldsymbol{s}, a, r, \boldsymbol{s}'\} \in \mathcal{D}_s$ do
7:         $a_i \leftarrow \arg\max_a f_i(\boldsymbol{s}, a)_\mu \quad i = 1, \ldots, N$
8:         $(q_{\mu_i}, q_{\sigma i}) \leftarrow f_i(\boldsymbol{s}, a_i) \quad i = 1, \ldots, N$
9:         $G_i \leftarrow r + \gamma \cdot f'_i(\boldsymbol{s}', a)_\mu \quad i = 1, \ldots, N$
10:        $\Delta_{\mathcal{L}} \leftarrow \Delta_{\mathcal{L}} + \frac{1}{|N|} \nabla_{\boldsymbol{\phi}} \sum_{i=1}^{N} \log P(G_i | q_{\mu_i}, q_{\sigma i})$
11:    end for
12:    $\mathbf{X} \leftarrow$ CreateMeasurementSet($\mathcal{D}_s$)
13:    $\Delta_{\text{KL}} \leftarrow \text{SSGE}(\{\mathbf{f}_{1:N}(\boldsymbol{x})\}_{\boldsymbol{x} \in \mathbf{X}})$
14:    $\boldsymbol{\phi} \leftarrow$ Optimizer($\boldsymbol{\phi}, \Delta_{\mathcal{L}} - \lambda \Delta_{\text{KL}}$)
15: end function
```

local reparameterization trick (LRT) [23] in our implementation. Using LRT, we sample pre-activations rather than the weights themselves, which results in significant speedup. This is especially important for BE, which does N times as many samples per iteration. LRT also reduces the variance of the gradient, which stabilizes training.

All experiments have been implemented in Julia, and the source code is available at https://github.com/XXX.[1]

Table 1
Hyperparameters for BE.

	Hyperparameter	Value
N	Number of function samples	100
J	Number of eigenvectors	6
λ	KL regularization parameter	$1/\lvert\mathcal{D}_s\rvert$
η	Regularization parameter for SSGE	0.95

4.2. Flat Prior

First, we will examine how BE compares to NoisyNet when we do not encode a priori knowledge about an environment into the model. To investigate this we employ a "flat" prior, namely an improper prior with constant probability on $\mathbb{R}^n$. An improper prior is not a probability density (since it does not integrate to 1), but this is not problematic as BE only requires the *gradients* and does not need to evaluate the probabilities themselves.

[1]The URL to the repository containing all the source code including scripts to reproduce our results will be made available once the article is accepted for publication.

Table 2
Number of episodes to solve select environments. Dash indicated the agent was not able to solve the environment within the allotted number of frames (30k for Cartpole, 500k for MountainCar, and 2m for LunarLander).

Prior	Cartpole	MountainCar	LunarLander
$\nu = 1, \sigma = 1$	**14**	**100**	—
$\nu = 1, \sigma = 10$	33	1700	—
$\nu = 1, \sigma = 50$	113	1250	**500**
Flat prior	117	1050	800
$\nu = -1, \sigma = 50$	177	—	—
$\nu = -1, \sigma = 20$	232	—	—
$\nu = -1, \sigma = 10$	—	—	—
NoisyNet	219	750	1200

Table 3
Prior parameters.

Environment	λ_1	λ_2	State	Actions			
				A_{Left}	A_{Right}		
Cartpole	0	100	Cart Position	0	0		
			Cart Velocity	0	0		
			Pole Angle	-1	1		
			Pole Velocity	-1	1		
				A_{Left}	A_{Right}	A_{Nothing}	
MountainCar	-100	1000	Cart Position	0	0	0	
			Cart Velocity	-1	1	0	
				A_{Left}	A_{Right}	A_{Nothing}	A_{Up}
LunarLander	0	100	Position x	0	0	0	0
			Position y	0	0	0	0
			Velocity x	0	0	0	0
			Velocity y	0	0	0	0
			Angle	-1	1	0	0
			Angular Velocity	-1	1	0	0
			Left Leg Touching	0	0	0	0
			Right Leg Touching	0	0	0	0

The training curves for BE and standard NoisyNet-DQN on the three selected OpenAI Gym environments are shown in Figure 1. While both methods can solve all environments, BE uses considerably fewer frames to find an optimal policy for Cartpole. The methods are comparable in the two other environments.

Fortunato et al. [3] noted that the learned variance in their weights increased during exploration in some environments despite there existing an optimal deterministic solution, and the loss provides no incentive to maintain uncertainty. Figure 2 shows the mean standard deviation for the penultimate and final layer for BE and NoisyNet evaluated on Cartpole. It is interesting to see that the last layer's standard deviation for NoisyNet continues to decrease throughout the training while BE's uncertainty initially decreases faster but then stabilises at a higher degree of uncertainty than NoisyNet's. Figure 1 shows that BE has found a near-optimal solution after approximately 5k frames and an optimal policy after 10k frames. Interestingly, the standard deviation of the weight parameters for BE in the last layer stops decreasing after 5k iterations. This seems to indicate that BE is satisfied that is has found a stable policy, where optimising further would be overfitting to noise. Given that the gradient in the last layer is sufficiently small, the nature of the chain rule will cause the gradient in the penultimate layer to be smaller, which can explain why the mean standard deviation in the penultimate layer decreases more slowly.

4.3. Informative Prior

One of the benefits of a functional prior is that we can incorporate domain knowledge to get more efficient exploration. We will now see how our method can utilise domain knowledge to improve sample efficiency. To measure the effectiveness of priors we will use a distribution that has a higher concentration of large Q-values for actions that we want to incentivize.

We purposefully do not do an extensive search for a "good" prior distributions. Rather, we are interested in the effect a "simple" prior can have on performance. The results reported in Table 2 will reveal the effectiveness of the prior distributions. To this end, we have chosen to define the prior as a Gaussian process with the following mean and kernel function:

$$\boldsymbol{\mu}(\boldsymbol{s}, a) = \lambda_1 + \nu \cdot \lambda_2 \cdot \mathrm{A}_a^\intercal \boldsymbol{s},$$
$$k(\boldsymbol{s}, \boldsymbol{s}') = \sigma^2 I(\boldsymbol{s} = \boldsymbol{s}'),$$

where we use the notation that $I(\mathcal{T}) = 1$ if $\mathcal{T}$ is true and 0 otherwise. Values for λ_1, λ_2, and A_a for each environment can be found in Table 3. ν is either $+1$, indicating a "helpful" prior, or -1, indicating an "unhelpful" prior. A_a was selected based on vague information such as *"In Cartpole, it is good to move left if the pole is leaning to the left, and vice versa"* and *"In MountainCar, it is better to move left if your cart is already moving to the left, and vice versa"*. λ_1 and λ_2 were set so that the prior mean for the Q-values would be roughly at the true mean, though we suspect this could be a restricting factor of our prior. Experimental results for varied values of ν and σ, can be seen in Table 2.

An alternative approach to defining the prior could be to focus on smoothness, i.e., use $k(\boldsymbol{s}, \boldsymbol{s}')$ to incorporate that states that are similar also are likely to have similar Q-values. This would also be a prior that does not necessarily need much domain knowledge to be effective.

For Cartpole, strong helpful priors result in a substantial benefit in the number of episodes to solve the environment, and even a weak unhelpful prior outperforms NoisyNet here. MountainCar benefits from a strong and helpful prior, solving the environment in as little as 100 episodes. However, a vague and presumably helpful prior appears to be harmful in this environment. For LunarLander, a strong helpful prior prevented the algorithm from solving the environment, yet a more vague prior was beneficial. This seems to indicate that the prior used in that environment was not very precise. Unsurprisingly, strong unhelpful priors prevented the algorithm from solving any environment, with runs terminated at 30k iterations for Cartpole, 500k iterations for MountainCar, and 2mill iterations for LunarLander. We observe that the strongest priors (both "helpful" and "unhelpful") may restrict the exploration too much, and unless the prior is focused on an optimal strategy, the environment is not solved. Overall, the results show that some effort has to be put into creating effective priors for certain environments, but that domain knowledge can be extremely valuable if it is available. Finally, BE with an appropriately defined prior outperformed NoisyNet [3] on all environments. We conclude that the Bayesian formulation combined with well-functioning priors can be an alternative to other strategies to provide domain knowledge, like reward shaping.

5. Conclusion and Discussion

This paper presents BayesianExplore (BE), a fully Bayesian reinforcement learning algorithm. This is valuable because it is known that posterior sampling of Q-values for reinforcement learning in finite horizon MDPs has (close to) optimal regret bound [20]. Initial experiments show that BE is comparable to NoisyNet in well-known test environments.

Next, since we utilized recent breakthroughs in function-space variational inference [16] to formulate the model as a stochastic process, we have the opportunity to encode domain knowledge into prior information that can lead to faster learning. BE with an informative prior outperforms NoisyNet in all environments.

One interesting avenue for future work is to extend the approach to methods other than the standard DQN. We hypothesise that BE can be adapted and used to improve exploration in any algorithm that is compatible with NoisyNet. A functional Bayesian approach for policy evaluation, where the Gaussian process we used in this paper would be replaced by a Dirichlet process, which would permit prior distributions in policy space rather than in value space. This can be a more intuitive representation of a priori knowledge in many situations.

References

[1] W. R. Thompson, On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples, Biometrika 25 (1933) 285–294. URL: https://www.jstor.org/stable/2332286. doi:10.2307/2332286, publisher: [Oxford University Press, Biometrika Trust].

[2] I. Osband, B. Van Roy, Z. Wen, Generalization and Exploration via Randomized

Value Functions, arXiv:1402.0635 [cs, stat] (2016). URL: http://arxiv.org/abs/1402.0635, arXiv: 1402.0635.

[3] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, S. Legg, Noisy Networks for Exploration, arXiv:1706.10295 [cs, stat] (2019). URL: http://arxiv.org/abs/1706.10295, arXiv: 1706.10295.

[4] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, D. Silver, Rainbow: Combining Improvements in Deep Reinforcement Learning, arXiv:1710.02298 [cs] (2017). URL: http://arxiv.org/abs/1710.02298, arXiv: 1710.02298.

[5] M. G. Bellemare, W. Dabney, R. Munos, A Distributional Perspective on Reinforcement Learning, arXiv:1707.06887 [cs, stat] (2017). URL: http://arxiv.org/abs/1707.06887, arXiv: 1707.06887.

[6] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, N. de Freitas, Dueling Network Architectures for Deep Reinforcement Learning, arXiv:1511.06581 [cs] (2016). URL: http://arxiv.org/abs/1511.06581, arXiv: 1511.06581.

[7] H. van Hasselt, A. Guez, D. Silver, Deep Reinforcement Learning with Double Q-learning, arXiv:1509.06461 [cs] (2015). URL: http://arxiv.org/abs/1509.06461, arXiv: 1509.06461.

[8] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized Experience Replay, arXiv:1511.05952 [cs] (2016). URL: http://arxiv.org/abs/1511.05952, arXiv: 1511.05952.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous Methods for Deep Reinforcement Learning, arXiv:1602.01783 [cs] (2016). URL: http://arxiv.org/abs/1602.01783, arXiv: 1602.01783.

[10] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight Uncertainty in Neural Network, in: F. Bach, D. Blei (Eds.), Proceedings of the 32nd International Conference on Machine Learning, volume 37 of *Proceedings of Machine Learning Research*, PMLR, Lille, France, 2015, pp. 1613–1622. URL: http://proceedings.mlr.press/v37/blundell15.html.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing Atari with Deep Reinforcement Learning, arXiv:1312.5602 [cs] (2013). URL: http://arxiv.org/abs/1312.5602, arXiv: 1312.5602.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.

[13] D. J. Rezende, S. Mohamed, D. Wierstra, Stochastic Backpropagation and Approximate Inference in Deep Generative Models, arXiv:1401.4082 [cs, stat] (2014). URL: http://arxiv.org/abs/1401.4082, arXiv: 1401.4082.

[14] H. Ritter, A. Botev, D. Barber, A Scalable Laplace Approximation for Neural Networks, 2018. URL: https://openreview.net/forum?id=Skdvd2xAZ.

[15] W. Maddox, T. Garipov, P. Izmailov, D. Vetrov, A. G. Wilson, A Simple Baseline for Bayesian Uncertainty in Deep Learning (2019). URL: https://arxiv.org/abs/1902.02476v2.

[16] S. Sun, G. Zhang, J. Shi, R. Grosse, Functional Variational Bayesian Neural Networks, arXiv:1903.05779 [cs, stat] (2019). URL: http://arxiv.org/abs/1903.05779, arXiv: 1903.05779.

[17] J. Shi, S. Sun, J. Zhu, A Spectral Approach to Gradient Estimation for Implicit Distributions, arXiv:1806.02925 [cs, stat] (2018). URL: http://arxiv.org/abs/1806.02925, arXiv: 1806.02925.

[18] E. J. Nyström, Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben, Acta Mathematica 54 (1933) 185–204.

[19] C. Williams, M. Seeger, Using the nyström method to speed up kernel machines, in: T. Leen, T. Dietterich, V. Tresp (Eds.), Advances in Neural Information Processing Systems 13 (NIPS 2000), MIT Press, 2001, pp. 682–688.

[20] I. Osband, B. Van Roy, Why is Posterior Sampling Better than Optimism for Reinforcement Learning?, arXiv:1607.00215 [cs, stat] (2017). URL: http://arxiv.org/abs/1607.00215, arXiv: 1607.00215.

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, arXiv:1606.01540 [cs] (2016). URL: http://arxiv.org/abs/1606.01540, arXiv: 1606.01540.

[22] S. Han, W. Zhou, J. Liu, S. Lü, NROWAN-DQN: A Stable Noisy Network with Noise Reduction and Online Weight Adjustment for Exploration, arXiv:2006.10980 [cs, stat] (2020). URL: http://arxiv.org/abs/2006.10980, arXiv: 2006.10980.

[23] D. P. Kingma, T. Salimans, M. Welling, Variational dropout and the local reparameterization trick, 2015. arXiv:1506.02557.