Randomized Prior Functions for Deep Reinforcement Learning

Ian Osband
DeepMind
iosband@google.com

John Aslanides
DeepMind
jaslanides@google.com

Albin Cassirer
DeepMind
cassirer@google.com

Abstract

Dealing with uncertainty is essential for efficient reinforcement learning. There is a growing literature on uncertainty estimation for deep learning from fixed datasets, but many of the most popular approaches are poorly-suited to sequential decision problems. Other methods, such as bootstrap sampling, have no mechanism for uncertainty that does not come from the observed data. We highlight why this can be a crucial shortcoming and propose a simple remedy through addition of a randomized untrainable 'prior' network to each ensemble member. We prove that this approach is efficient with linear representations, provide simple illustrations of its efficacy with nonlinear representations and show that this approach scales to large-scale problems far better than previous attempts.

1 Introduction

Deep learning methods have emerged as the state of the art approach for many challenging problems [28, 66]. This is due to the statistical flexibility and computational scalability of large and deep neural networks, which allows them to harness the information in large and rich datasets. Deep reinforcement learning combines deep learning with sequential decision making under uncertainty. Here an agent takes actions inside an environment in order to maximize some cumulative reward [61]. This combination of deep learning with reinforcement learning (RL) has proved remarkably successful [64, 40, 58].

At the same time, elementary decision theory shows that the only admissible decision rules are Bayesian [12, 67]. Colloquially, this means that any decision rule that is not Bayesian can be improved (or even exploited) by some Bayesian alternative [14]. Despite this fact, the majority of deep learning research has evolved outside of Bayesian (or even statistical) analysis [53, 30]. This disconnect extends to deep RL, where the majority of state of the art algorithms have no concept of uncertainty [40, 39] and can fail spectacularly even in simple problems where success requires its consideration [38, 43].

There is a long history of research in Bayesian neural networks that never quite became mainstream practice [35, 41]. Recently, Bayesian deep learning has experienced a resurgence of interest with a myriad of approaches for uncertainty quantification in fixed datasets and also sequential decision problems [27, 10, 19, 45]. In this paper we highlight the surprising fact that many of these well-cited and popular methods for uncertainty estimation in deep learning can be poor choices for sequential decision problems. We show that this disconnect is more than a technical detail, but a serious shortcoming that can lead to arbitrarily poor performance. We support our claims by a series of simple lemmas for simple environments, together with experimental evidence in more complex settings.

Our approach builds on an alternative method for uncertainty in deep RL inspired by the statistical bootstrap [15]. This approach trains an ensemble of models, each on perturbed

Preprint. Work in progress.

versions of the data. The resulting distribution *of the ensemble* is used to approximate the uncertainty in the estimate [45]. Although typically regarded as a frequentist method, bootstrapping gives near-optimal convergence rates when used as an approximate Bayesian posterior [18, 17]. However, these ensemble-based approaches to uncertainty quantification approximate a 'posterior' without an effective methodology to inject a 'prior'. This can be a crucial shortcoming in sequential decision problems.

In this paper, we present a simple modification where each member of the ensemble is initialized together with a random but fixed *prior function*. Predictions are then taken as the sum of the trainable neural network and the prior function. We show that this approach passes a sanity check by demonstrating an equivalence to Bayesian inference with linear models. We also present a series of simple experiments designed to extend this intuition to deep learning. We show that many of the most popular approaches for uncertainty estimation in deep RL do *not* pass these sanity checks, and crystallize these shortcomings in a series of lemmas and small examples. We demonstrate that our simple modification can facilitate aspiration in difficult tasks where previous approaches for deep RL fail. We believe that this work presents a simple and practical approach to encoding prior knowledge with deep reinforcement learning.

2 Why do we need a 'prior' mechanism for deep RL?

We model the environment as a Markov decision process $M = (\mathcal{S}, \mathcal{A}, R, P, \gamma)$ [9]. Here $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space and $\gamma \in [0, 1)$ is the discount factor. At each time step t, the agent observes state $s_t \in \mathcal{S}$, takes action $a_t \in \mathcal{A}$, receives reward $r_t \sim R(s_t, a_t)$ and transitions to $s_{t+1} \sim P(s_t, a_t)$. A policy $\pi : \mathcal{S} \rightarrow \Delta\mathcal{A}$ maps states to distributions over actions and let $\mathcal{H}_t$ denote the history of observations before time t. An RL algorithm maps $\mathcal{H}_t$ to policies; we assess its quality through the cumulative reward over unknown environments. To perform well, an RL algorithm must learn to optimize its actions, combining both learning and control [61]. A 'deep' RL algorithm uses neural networks for nonlinear function approximation in its learning [30, 40].

The scale and scope of problems that might be approached through deep RL is vast, but there are three key aspects an efficient (and general) agent must address:

1. **Generalization**: be able to learn from data it collects.
2. **Exploration**: prioritize the best experiences to learn from.
3. **Long-term consequences**: consider external effects beyond a single time step.

In this paper we focus on the importance of some form of 'prior' mechanism for efficient exploration. As a motivating example we consider a sparse reward task where random actions are very unlikely to ever see a reward. If an agent has never seen a reward then it is essential that some other form of aspiration, motivation, drive or curiosity direct its learning. We call this type of drive a 'prior' effect, since it does not come from the observed data, but are ambivalent as to whether this effect is philosophically 'Bayesian'. Agents that do not have this prior drive will be left floundering aimlessly and thus may require exponentially large amounts of data in order to learn even simple problems [25].

To solve a specific task, it can be possible to attain superhuman performance without significant prior mechanism [40, 39]. However, if our goal is artificial *general* intelligence, then it is disconcerting that our best agents can perform very poorly even in simple problems [31, 37]. One potentially general approach to decision making is given by the Thompson sampling heuristic[1]: 'randomly take action according to the probability you believe it is the optimal action' [65]. In recent years there have been several attempts to apply this heuristic to deep reinforcement learning, each attaining significant outperformance over deep RL baselines on certain tasks [19, 45, 33, 10, 16]. In this section we outline several failure cases for each of these methods. This sets the scene for Section 3, where we introduce a simple and practical alternative that passes each of our simple sanity checks: bootstrapped

[1]This heuristic is general in the sense that Thompson sampling can be theoretically justified in many of the domains where these other approaches fail [1, 46, 32, 56].

ensembles with randomized prior functions. In Section 4 we demonstrate that this approach scales gracefully to complex domains with deep RL.

2.1 Dropout as posterior approximation

One of the most popular modern approaches to regularization in deep learning is dropout sampling [59]. During training, dropout applies an independent random Bernoulli mask to the activations and thus guards against excessive co-adaptation of weights. Recent work has sought to understand dropout through a Bayesian lens, highlighting the connection to variational inference and claiming that the resultant dropout distribution approximates a Bayesian posterior [19]. This narrative has proved popular despite the fact that dropout distribution can be a poor approximation to most reasonable Bayesian posteriors [20, 44]:

Lemma 1 (Dropout distribution does not concentrate with observed data)**.**
Consider any loss function $\mathcal{L}$, regularizer $\mathcal{R}$ and data $\mathcal{D}=\{(x,y)\}$. Let θ be parameters of any neural network architecture f trained with dropout $p\in(0,1)$ to optimize

$$\theta_p^* \in \arg\min_{\theta} \mathbb{E}_{W\sim\mathrm{Ber}(p),(x,y)\sim\mathcal{D}}\left[\mathcal{L}(x,y \mid \theta, W) + \mathcal{R}(\theta)\right]. \tag{1}$$

Then the dropout distribution $f_{\theta_p^*,W}$ is invariant to duplicates of the dataset $\mathcal{D}$.[2]

Lemma 1 is somewhat contrived, but suggests that the distribution obtained by dropout sampling may be ill-suited to sequential decision problems. No agent employing dropout for posterior approximation can tell the difference between observing a set of data once and observing it $N \gg 1$ times. We show that this can lead to arbitrarily poor decision making, even when combined with an efficient strategy for exploration such as Thompson sampling. We provide more background and details in Appendix A.1.

2.2 Variational inference

Dropout as posterior is motivated by its connection to variational inference [20]. Although we have outlined a potential inadequacy of dropout in Section 2.1, this is not a reason to discount alternative forms of variational inference [10]. As we will now outline, there are some delicate issues to be aware of for reinforcement learning.

First, many applications of variational inference (VI) model the distribution over network weights as a product of independent Gaussians [10]. These models facilitate efficient computation, but can underestimate the uncertainty and may be a poor choice for encoding prior knowledge. Even if one were given a mapping of prior knowledge to weights, the confounding demands of good initialization for SGD training may interfere [21]. For this reason practical applications of VI to RL typically use very little prior effect, or even no prior regularization at all [33, 16]. However, there is a much deeper issue that befalls many naïve applications of VI to RL, which we outline in Lemma 2.

Lemma 2 (Independent VI on Bellman error does not propagate uncertainty)**.**
Let $Y\sim N(\mu_Y,\sigma_Y^2)$ be a target value. If we train $X\sim N(\mu,\sigma^2)$ on the variational loss [16]

$$\mu^*, \sigma^* \in \arg\min_{\mu,\sigma} \mathbb{E}\left[(X-Y)^2\right] \qquad \text{for } X, Y \text{ independent}, \tag{2}$$

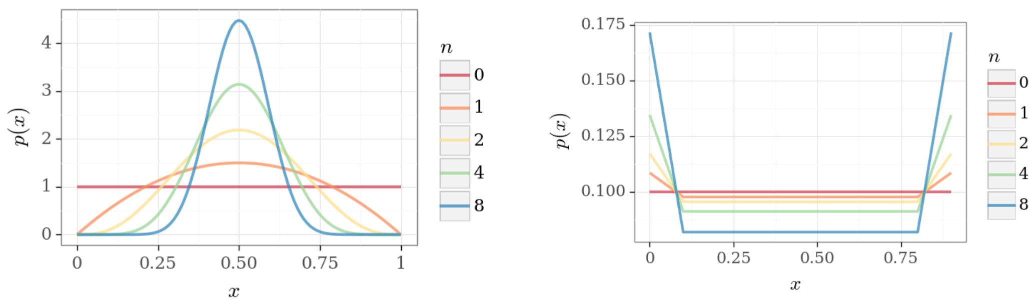
then the solution $\mu^* = \mu_Y, \sigma^* = 0$ propagates zero uncertainty from Y to X.

To understand the significance of Lemma 2, imagine a deterministic system that transitions from s_1 to s_2 without reward. Suppose an agent is able to correctly quantify their posterior uncertainty for the value $V(s_2)=Y\sim N(\mu_Y,\sigma_Y^2)$. Training $V(s_1)=X$ according to the loss of [16, 33] will lead to zero uncertainty estimates at s_1, when in fact $V(s_1)\sim N(\mu_Y,\sigma_Y^2)$. Note that this occurs without decision making, function approximation and even when the true posterior lies within the variational class. We expand on this example in Appendix A.2 and outline an alternative approach to VI for RL.

[2] Recent work suggests adaptively tuning the dropout rate from data, but does not change the behavior of Lemma 1; the resultant distribution is still invariant under copies of the data.

2.3 ‘Distributional reinforcement learning’

The key ingredient for a Bayesian formulation for sequential decision making is to consider beliefs not simply as a point estimate, but as a *distribution*. Recently an approach called ‘distributional RL’ has shown great success in improving stability and performance in deep RL benchmark algorithms [7]. Despite the name, these two ideas are quite distinct. ‘Distributional RL’ replaces a scalar estimate for the value function by a distribution that is trained to minimize a loss against the distribution of data it observes. This distribution of observed data is an orthogonal concept to that of Bayesian uncertainty.



(a) Posterior beliefs concentrate around $p = 0.5$. (b) ‘Distributional’ tends to mass at 0 and 1.

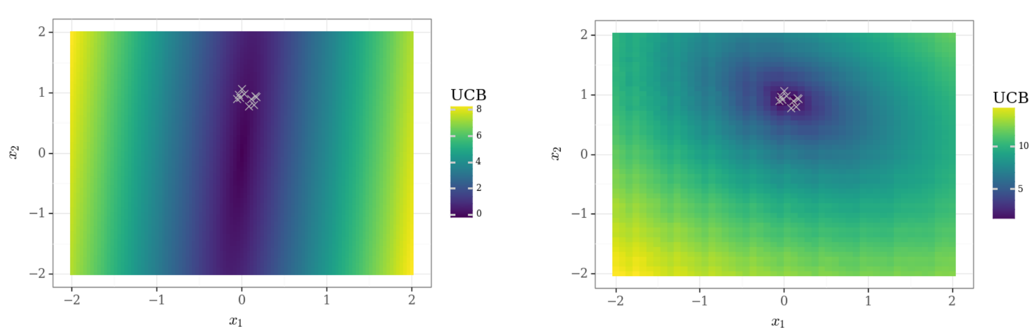
Figure 1: Output distribution after observing n heads and n tails of a coin.

Figure 1 presents an illustration of these two distributions after observing flips of a coin. As more data is gathered the posterior distribution concentrates around the mean whereas the ‘distributional RL’ estimate approaches that of the generating Bernoulli. Although both approaches might reasonably claim a distributional perspective on RL, these two distributions have orthogonal natures and behave quite differently. Conflating one for the other can lead to arbitrarily poor decisions; we push these details to Appendix A.3.

2.4 Count-based uncertainty estimates

Another popular method for incentivizing exploration is with a density model or ‘pseudocount’ [6]. Inspired by the analysis of tabular systems, these models assign a bonus to states and actions that have been visited infrequently according to a density model. This method can perform well, but only when the generalization of the density model is aligned with the task objective. Crucially, this generalization is not learned from the task [51].

Even with an optimal state representation and density, a count-based bonus on states can be poorly suited for efficient exploration. Consider a linear bandit with reward $r_t(x_t) = x_t^T \theta^* + \epsilon_t$ for some $\theta^* \in \mathbb{R}^d$ and $\epsilon_t \sim N(0, 1)$ [54]. Figure 2 compares the uncertainty in the expected reward $\mathbb{E}[x^T \theta^*]$ with that obtained by density estimation on the observed x_t. A bonus based upon the state density does not correlate with the *uncertainty* over the unknown optimal action. This disconnect can lead to arbitrarily poor decisions; see Appendix A.4.



(a) Uncertainty bonus from posterior over $x^T \theta^*$. (b) Bonus from Gaussian pseudocount $p(x)$.

Figure 2: Count-based uncertainty leads to a poorly aligned bonus even in a linear system.

3 Randomized prior functions for deep ensembles

Section 2 motivates the need for effective uncertainty estimates in deep RL. We note that crucial failure cases of several popular approaches can arise even with simple linear models. As a result, we take a moment to review the setting of Bayesian linear regression. Let $\theta \in \mathbb{R}^d$ with prior $N(\overline{\theta}, \lambda I)$ and data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ for $x_i \in \mathbb{R}^d$ and $y_i = \theta^T x_i + \epsilon_i$ with $\epsilon_i \sim N(0, \sigma^2)$ iid. Then, conditioned on $\mathcal{D}$, the posterior for θ is Gaussian:

$$\mathbb{E}[\theta|\mathcal{D}] = \left(\frac{1}{\sigma^2} X^T X + \frac{1}{\lambda} I\right)^{-1} \left(\frac{1}{\sigma^2} X^T y + \frac{1}{\lambda}\overline{\theta}\right), \quad \mathrm{Cov}[\theta|\mathcal{D}] = \left(\frac{1}{\sigma^2} X^T X + \frac{1}{\lambda} I\right)^{-1}. \qquad (3)$$

Equation 3 relies on Gaussian conjugacy and linear models, which cannot easily be extended to deep neural networks. The following result shows that we can replace this analytical result with a simple computational procedure.

Lemma 3 (Computational generation of posterior samples)**.**
Let $f_\theta(x) = x^T\theta$, $\tilde{y}_i \sim N(y_i, \sigma^2)$ and $\tilde{\theta} \sim N(\overline{\theta}, \lambda I)$. Then the either of the following optimization problems generate a sample $\theta \mid \mathcal{D}$ according to (3)*:*

$$\underset{\theta}{\mathrm{argmin}} \sum_{i=1}^n \|\tilde{y}_i - f_\theta(x_i)\|_2^2 + \frac{\sigma^2}{\lambda}\|\tilde{\theta} - \theta\|_2^2, \qquad (4)$$

$$\underset{\theta}{\mathrm{argmin}} \sum_{i=1}^n \|\tilde{y}_i - (f_{\tilde{\theta}} + f_\theta)(x_i)\|_2^2 + \frac{\sigma^2}{\lambda}\|\theta\|_2^2. \qquad (5)$$

Proof. To prove (4) note that the solution is Gaussian and then match moments; equation (5) then follows by relabeling [47]. □

Lemma 3 is revealing since it allows us to view Bayesian regression through a purely computational lens, "generate posterior samples by training on noisy versions of the data, together with some random regularization". Even for nonlinear f_θ, we can still compute (4) or (5). Although the resultant f_θ will no longer be an exact posterior, at least it passes the 'sanity check' in this simple linear setting (unlike the approaches of Section 2). We argue this method is quite intuitive: the perturbed data $\tilde{\mathcal{D}} = \{(x_i, \tilde{y}_i)\}_{i=1}^n$ is generated according to the estimated noise process ϵ_t and the sample $\tilde{\theta}$ is drawn from prior beliefs. Intuitively (4) says to fit to $\tilde{\mathcal{D}}$ and regularize weights to a prior sample of weights $\tilde{\theta}$; (5) says to generate a prior *function* $f_{\tilde{\theta}}$ and then fit an additive term to noisy data $\tilde{\mathcal{D}}$ with regularized complexity.

This paper explores the performance of each of these methods for uncertainty estimation with deep learning. We find empirical support that method (5) coupled with a *randomized prior function* significantly outperforms ensemble-based approaches without prior mechanism. We also find that (5) significantly outperforms (4) in deep RL. We suggest a major factor in this comes down to the huge dimensionality of neural network weights, whereas the output policy or value is typically far smaller. In this case, it makes sense to enforce prior beliefs in the low dimensional space. Further, the initialization of neural network weights plays an important role in their generalization properties and optimization via SGD [21, 36]. As such, (5) may help to decouple the dual roles of initial weights as both 'prior' and training initializer. Algorithm 1 describes our approach applied to modern deep learning architectures.

Algorithm 1 Randomized prior functions for ensemble posterior.

Require: Data $\mathcal{D} \subseteq \{(x,y) | x \in \mathcal{X}, y \in \mathcal{Y}\}$, loss function $\mathcal{L}$, neural model $f_\theta : \mathcal{X} \to \mathcal{Y}$,
Ensemble size $K \in \mathbb{N}$, noise procedure `data_noise`, distribution over priors $\mathcal{P} \subseteq \{\mathbb{P}(p) | p : \mathcal{X} \to \mathcal{Y}\}$.

for $k = 1, .., K$ **do**
  initialize $\theta_k \sim$ Glorot initialization [21].
  form $\mathcal{D}_k =$ `data_noise`($\mathcal{D}$) (e.g. Gaussian noise or bootstrap sampling [48]).
  sample prior function $p_k \sim \mathcal{P}$.
  optimize $\mathcal{L}(f_{\theta_k} + p_k; \mathcal{D}_k)$ wrt θ_k via ADAM [26].
end for
return ensemble $\{f_{\theta_k} + p_k\}_{k=1}^K$.

4 Deep reinforcement learning

Algorithm 1 might be fruitfully applied to model or policy learning approaches, but this paper focuses on value learning. We apply Algorithm 1 to *deep Q networks* (DQN) [40] on a series of RL tasks designed to require good uncertainty estimates. We train K networks $\{Q_k\}_{k=1}^K$ in parallel, each on a perturbed version of the observed data $\mathcal{H}_t$ and each with a distinct random, but fixed, prior function p_k. Each episode, the agent selects $j \sim \mathrm{Unif}([1, .., K])$ and follows the greedy policy w.r.t. Q_j for the duration of the episode. This algorithm is essentially bootstrapped DQN [45] except for the addition of the prior function p_k.

Lemma 3 highlights the connection between this approach and RLSVI in the linear setting recomputing $K = 1$ at the start of each episode [50]. RLSVI has a bound on expected regret[3] of $\tilde{O}(\sqrt{|\mathcal{S}||\mathcal{A}|T})$ in the tabular setting [47]. An analysis for the bandit setting establishes that $K = \tilde{O}(|\mathcal{A}|)$ models trained online can attain similar performance to full resampling each episode [34]. Our experiments push beyond the boundaries of these analyses, with nonlinear function approximation and ensemble sampling applied to the RL setting. We use bootstrapping rather than additive Gaussian noise as this automatically estimates a state-dependent variance from the data [18]. We find empirical evidence that many of these insights extend to this nonlinear setting and highlight the practical role of the prior function.

Our experiments focus on a series of environments that require deep exploration together with increasing state complexity [25, 47]. In each of our domains, random actions are very unlikely to achieve a reward and exploratory actions may even come at a cost. Any algorithm without prior motivation will have no option but to explore randomly, or worse, eschew exploratory actions completely in the name of premature and sub-optimal exploitation. In our experiments we focus on a *tabula rasa* setting where the prior function is drawn as a random neural network. Although it is clear that the prior distribution $\mathcal{P}$ could encode task-specific knowledge (e.g. through sampling the true Q^*), we leave this investigation to future work.

4.1 Chain environments

We begin our experiments with a family of chain-like environments that highlight the need for deep exploration [60]. The environments are indexed by problem scale $N \in \mathbb{N}$ and action mask $W \sim \mathrm{Ber}(0.5)^{N \times N}$, with $\mathcal{S} = \{0,1\}^{N \times N}$ and $\mathcal{A} = \{0,1\}$. The agent begins each episode in the upper left-most state in the grid and deterministically falls one row per time step. The state encodes the agent's row and column as a one-hot vector $s_t \in \mathcal{S}$. The actions $\{0,1\}$ move the agent left or right depending on the action mask W at state s_t, which remains fixed. The agent incurs a cost of $0.01/N$ for moving right in all states except for the right-most, in which the reward is 1. The reward for action left is always zero. An episode ends after N time steps so that the optimal policy is to move right each step and receive a total return of 0.99; all other policies receive zero or negative return. Crucially, algorithms without deep exploration take $\Omega(2^N)$ episodes to learn the optimal policy [50].[4]

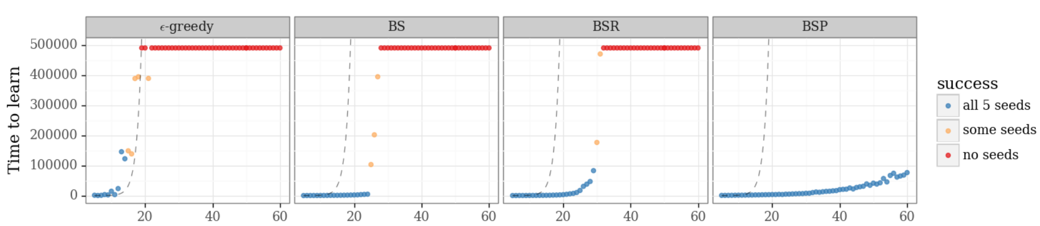


Figure 3: Only bootstrap with additive prior network (BSP) scales gracefully to large problems. Plotting BSP on a log-log scale suggests an empirical scaling $T_{\mathrm{learn}} = \tilde{O}(N^3)$; see Figure 10.

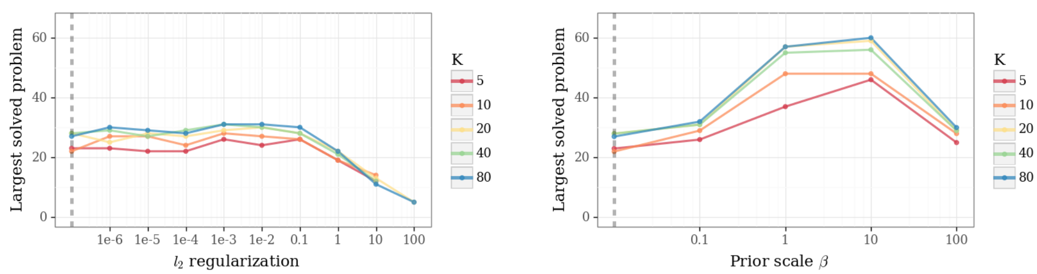
Figure 3 presents the average time to learn for $N = 5, .., 60$ up to 500K episodes over 5 seeds and ensemble $K = 20$. We say that an agent has learned the optimal policy when

[3] Regret measures the shortfall in cumulative rewards compared to that of the optimal policy.

[4] The dashed lines indicate the 2^N dithering lower bound. The action mask W means this cannot be solved easily by evolution or policy search evolution, unlike previous 'chain' examples [45, 52].

the average regret per episode drops below 0.9. We compare three variants of Bootstrapped DQN, depending on their mechanism for 'prior' effects. **BS** is bootstrap without prior mechanism. **BSR** is bootstrap with l_2 regularization on weights (4). **BSP** is bootstrap with additive prior function (5). In each case we initialize a random 20-MLP; BSR regularizes to these initial weights and BSP trains an additive network. Although all bootstrap methods significantly outperform ϵ-greedy, only BSP successfully scales to large problem sizes.

Figure 4 presents a more detailed analysis of the sensitivity of our approach to the tuning parameters of different regularization approaches. We repeat the experiments of Figure 3 and examine the size of the largest problem solved before 50K episodes. In each case we can see that larger ensembles lead to better performance, but that this effect tends to plateau relatively early. Figure 4a shows that regularization provides little or no benefit to BSR. Figure 4b examines the effect of scaling the randomly initialized MLP by a scalar hyperparameter β.



(a) l_2 regularization has a very limited effect. (b) Additive prior greatly improves performance.

Figure 4: Comparing effects of different styles of prior regularization in Bootstrapped DQN.

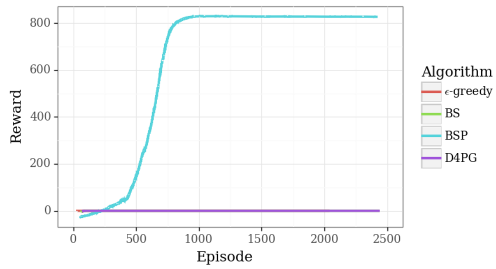
We can understand these results by the connection of BSP (5) to RLSVI with neural networks [50] (Lemma 3). The empirical scaling of Figure 3 mirrors the theory and simulation in this tabular domain [49]. The additive random MLP plays a role similar to the prior; it provides motivation for the agent to explore even when the observed data has low (or no) reward. It is not necessary that the prior function leads to a good policy itself; in fact this is exponentially unlikely according to our initialization scheme. The prior function simply functions to foster diversity over unseen data and washes out as more experience is gathered. Its scale implies a scale of *aspiration* for the agent: even when the observed rewards are low, it is motivated by the imagined rewards according to the prior p_k.

However, this connection to linear RLSVI does not inform why BSP should outperform BSR. To account for this, we appeal to the functional dynamics of deep learning architectures (see Section 3). In large networks weight decay (per BSR) may be an ineffective mechanism on the *output* Q-values. Instead, training an additive network via SGD (per BSP) may provide a more effective regularization on the output function [69, 36, 5]. We expand on this hypothesis and further details of these experiments in Appendix C.1. This includes investigation of NoisyNets [16] and dropout [19], which both perform poorly in our experiments.

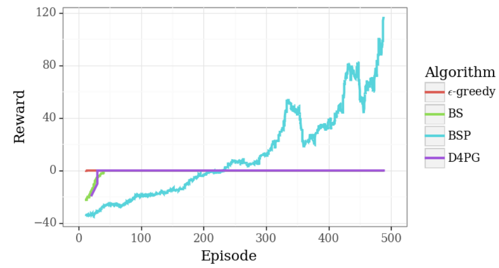
4.2 Cartpole swing-up

The experiments of Section 4.1 show that the choice of prior mechanism can be absolutely essential for efficient exploration via randomized value functions. However, since the underlying system is a small finite MDP we might observe similar performance through a tabular algorithm. In this section we investigate a classic benchmark problem that necessitates nonlinear function approximation: cartpole [61]. We modify the classic formulation so that the pole begins hanging down and the agent only receives a reward when the pole is upright, balanced, and centered[5]. We also and add a cost of 0.1 for moving the cart. This problem embodies many of the same aspects of 4.1, but since the agent interacts with the environment through state $s_t = (\cos(\theta_t), \sin(\theta_t), \dot{\theta}_t, x_t, \dot{x}_t)$, the agent must also learn nonlinear generalization. Tabular approaches are not practical due to the curse of dimensionality.

[5]We use the DeepMind control suite [63] with reward +1 only when $\cos(\theta) > 0.95$, $|x| < 0.1$, $|\dot{\theta}| < 1$, and $|\dot{x}| < 1$. Each episode lasts 1,000 time steps, simulating 10 seconds of interaction.



(a) Only BSP learns a performant policy.



(b) Inspecting the first 500 episodes.

Figure 5: Learning curves for the modified cartpole swing-up task.

Figure 5 compares the performance of DQN with ϵ-greedy, bootstrap without prior (BS), bootstrap with prior networks (BSP) and the state-of-the-art continuous control algorithm D4PG, itself an application of 'distributional RL' [4]. Only BSP learns a performant policy; no other approach ever attains any positive reward. We push experimental details, including hyperparameter analysis, to Appendix C.2. These results are significant in that they show that our intuitions translate from simple domains to more complex nonlinear settings, although the underlying state is relatively low dimensional. Our next experiments investigate performance in a high dimensional and visually rich domain.

4.3 Montezuma's revenge

Our final experiment comes from the Arcade Learning Environment and the canonical sparse reward game, Montezuma's Revenge [8]. The agent interacts directly with the pixel values and, even under an optimal policy, there can be hundreds of time steps between rewarding actions. This problem presents a significant exploration challenge in a visually rich environment; many published algorithms are essentially unable to attain any reward here [40, 39]. We compare performance against a baseline distributed DQN agent with double Q-learning, prioritized experience replay and dueling networks [23, 22, 57, 68]. To save computation we follow previous work and use a shared convnet for the ensemble uncertainty [45, 3]. Figure 6 presents the results for varying prior scale β averaged over three seeds. Once again, we see that the prior network can be absolutely critical to successful exploration.

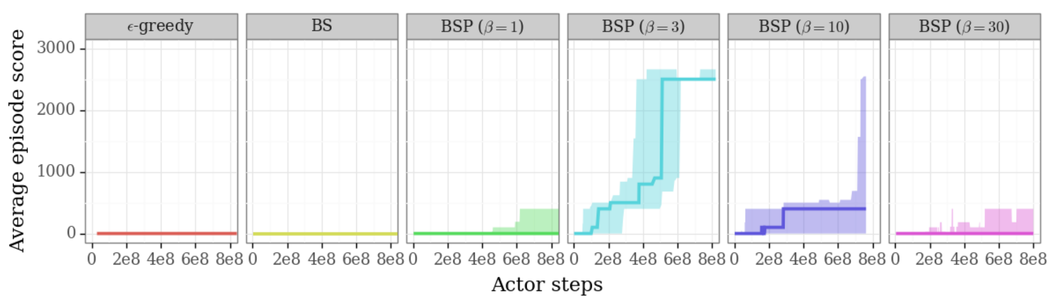


Figure 6: The prior network qualitatively changes behavior on Montezuma's revenge.

5 Conclusion

This paper highlights the need for effective uncertainty estimates in deep RL; particularly in domains that require efficient exploration. We highlight some alarming shortcomings of existing methods and suggest bootstrapped ensembles with randomized prior functions as a simple, practical alternative. We support our claims through an analysis of this method in the linear setting, together with a series of simple experiments designed to highlight the key issues. Our work leaves several open questions. What kinds of prior functions are appropriate for deep RL? Can they be optimized or 'meta-learned'? Can we distill the ensemble process to a single network? We hope this work helps to inspire solutions to these problems, and also build connections between the theory of efficient learning and practical algorithms for deep RL.

Acknowledgements
The authors wish to acknowledge the helpful comments from Benjamin Van Roy, Justin Sirignano, Mohammad Gheshlaghi Azar, David Budden and more generally to the wonderful colleagues at DeepMind.

References

[1] Shipra Agrawal and Navin Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In *Conference on Learning Theory*, pages 39–1, 2012.

[2] Shipra Agrawal and Navin Goyal. Further optimal regret bounds for Thompson sampling. In *Artificial Intelligence and Statistics*, pages 99–107, 2013.

[3] Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. *arXiv preprint arXiv:1802.04412*, 2018.

[4] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

[5] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems 30*, pages 6241–6250, 2017.

[6] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems 29*, pages 1471–1479. 2016.

[7] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458, 2017.

[8] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

[9] Dimitri P. Bertsekas and John Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, September 1996.

[10] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[11] Richard Y. Chen, Szymon Sidor, Pieter Abbeel, and John Schulman. UCB and infogain exploration via q-ensembles. *CoRR*, abs/1706.01502, 2017.

[12] David Roxbee Cox and David Victor Hinkley. *Theoretical statistics*. CRC Press, 1979.

[13] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[14] Bruno De Finetti. La prévision: ses lois logiques, ses sources subjectives. In *Annales de l'institut Henri Poincaré*, volume 7, pages 1–68, 1937.

[15] Bradley Efron. *The jackknife, the bootstrap and other resampling plans*, volume 38. SIAM, 1982.

[16] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. In *Proc. of ICLR*, 2018.

[17] Tadayoshi Fushiki. Bootstrap prediction and bayesian prediction under misspecified models. *Bernoulli*, pages 747–758, 2005.

[18] Tadayoshi Fushiki, Fumiyasu Komaki, Kazuyuki Aihara, et al. Nonparametric bootstrap prediction. *Bernoulli*, 11(2):293–307, 2005.

[19] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, 2016.

[20] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, 2016.

[21] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[22] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2094–2100. AAAI Press, 2016.

[23] Daniel Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. In *6th International Conference on Learning Represenations*, 2018.

[24] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

[25] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49, 2002.

[26] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *Proceedings of the International Conference on Learning Representations*, 2015.

[27] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105, 2012.

[29] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6405–6416, 2017.

[30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[31] Shane Legg, Marcus Hutter, et al. A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and applications*, 157:17, 2007.

[32] Jan Leike, Tor Lattimore, Laurent Orseau, and Marcus Hutter. Thompson sampling is asymptotically optimal in general environments. *Uncertainty in Artificial Intelligence*, 2016.

[33] Zachary C Lipton, Jianfeng Gao, Lihong Li, Xiujun Li, Faisal Ahmed, and Li Deng. Efficient exploration for dialogue policy learning with bbq networks & replay buffer spiking. *arXiv preprint arXiv:1608.05081*, 2016.

[34] Xiuyuan Lu and Benjamin Van Roy. Ensemble sampling. In *Advances in Neural Information Processing Systems*, pages 3260–3268, 2017.

[35] David JC MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[36] Hartmut Maennel, Olivier Bousquet, and Sylvain Gelly. Gradient descent quantizes ReLU network features. *arXiv preprint arXiv:1803.08367*, 2018.

[37] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

[38] Oliver Mihatsch and Ralph Neuneier. Risk-sensitive reinforcement learning. *Machine learning*, 49(2-3):267–290, 2002.

[39] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proc. of ICML*, 2016.

[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[41] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[42] Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. *arXiv preprint arXiv:1709.05380*, 2017.

[43] Ian Osband. *Deep Exploration via Randomized Value Functions*. PhD thesis, Stanford University, 2016.

[44] Ian Osband. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS Bayesian Deep Learning Workshop*, 2016.

[45] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances In Neural Information Processing Systems 29*, pages 4026–4034, 2016.

[46] Ian Osband, Daniel Russo, and Benjamin Van Roy. (More) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems 26*, pages 3003–3011. 2013.

[47] Ian Osband, Daniel Russo, Zheng Wen, and Benjamin Van Roy. Deep exploration via randomized value functions. *arXiv preprint arXiv:1703.07608*, 2017.

[48] Ian Osband and Benjamin Van Roy. Bootstrapped Thompson sampling and deep exploration. *arXiv preprint arXiv:1507.00300*, 2015.

[49] Ian Osband and Benjamin Van Roy. Why is posterior sampling better than optimism for reinforcement learning? In *Proceedings of the 34th International Conference on Machine Learning*, pages 2701–2710, 2017.

[50] Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2377–2386, 2016.

[51] Georg Ostrovski, Marc G Bellemare, Aaron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *Proc. of ICML*, 2017.

[52] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[53] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[54] Paat Rusmevichientong and John N. Tsitsiklis. Linearly parameterized bandits. *Math. Oper. Res.*, 35(2):395–411, 2010.

[55] Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014.

[56] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, and Ian Osband. A tutorial on Thompson sampling. *arXiv preprint arXiv:1707.02038*, 2017.

[57] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[58] David Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[59] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[60] Malcolm Strens. A Bayesian framework for reinforcement learning. In *International Conference on Machine Learning*, pages 943–950, 2000.

[61] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. 2017.

[62] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[63] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[64] Gerald Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[65] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[66] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[67] Abraham Wald. Statistical decision functions. In *Breakthroughs in Statistics*, pages 342–357. Springer, 1992.

[68] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.

[69] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530, 2016.

A Why do we need a 'prior' mechanism for deep RL?

Section 2 outlines the need for a prior mechanism in deep RL, together with key failure cases for several of the most popular approaches. Due to space limitations we provide only simple illustrations of potential inadequacies of each method and this does not preclude their efficacy on any particular domain. In this appendix we expand on the details provided in Section 2 and provide suggestions for how these approaches might be remedied by future work.

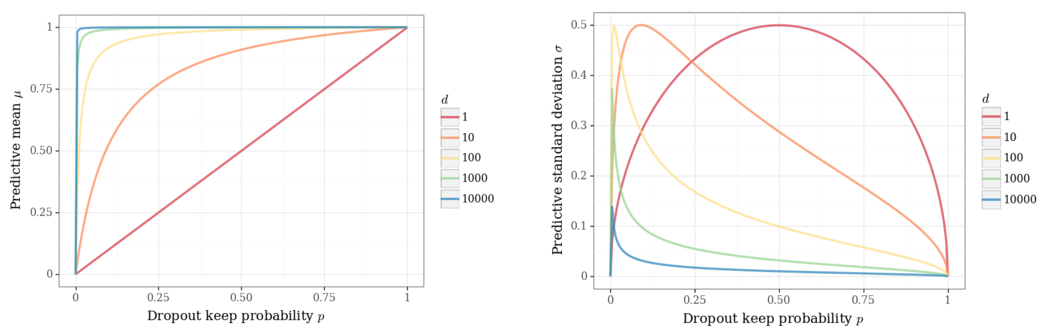
A.1 Dropout as posterior approximation

Previous work has suggested that dropout works as an effective variational approximation to the Bayesian posterior in neural networks, without special consideration for the network architecture [19]. However, Lemma 1 is a general statement that gives us cause to question the quality of this approximation. In this subsection we dig deeper into an extremely simple estimation problem, a linear network with d units to estimate the mean of a random variable $Y \in \mathbb{R}$. Even in this simple setting dropout performs poorly as a Bayesian posterior.

We form predictions $f_\theta = \sum_{i=1}^d w_i \theta_i$ with $w_i \sim \mathrm{Ber}(p)$, square loss and regularizer $\mathcal{R}(\theta) = \lambda \|\theta\|^2$ for $\lambda > 0$. Then for any data $\mathcal{D}$ with empirical mean $\overline{y}$, the expected loss solution to (1) is given by[6] [59]

$$\theta_p^* = \overline{\theta}\mathbb{1} \text{ for } \overline{\theta} = \frac{\overline{y}}{1 + p(d-1) + \frac{\lambda}{d}}. \tag{6}$$

The resultant distribution has mean $\mu = \overline{\theta} dp$ and standard deviation $\sigma = \overline{\theta}\sqrt{dp(1-p)}$. Figure 7 presents illustrations of this mean and standard deviation as d, p vary.



(a) Mean of dropout sampling.

(b) Standard deviation of dropout sampling.

Figure 7: Dropout posterior approximation depends only on d, p and $\overline{y}$ (which we set to 1).

If we are to understand dropout as an approximation to a Bayesian posterior, then we should note that this behavior is unusual. First, the only connection to the data is through the empirical mean $\overline{y}$; any possible dataset with the same mean would result in the same 'posterior' distribution. Second we note that $\sigma = \mu\sqrt{(1-p)/dp}$. This coupling means it is not possible for $\sigma \to 0$ and $\mu \nrightarrow 0$, regardless of λ. More typically we would imagine $\mu \to \mathbb{E}[Y]$ and $\sigma \to 0$ according to the Bayesian central limit theorem [12].

This disconnect is not simply an analytical mistake, but can lead to arbitrarily bad decisions in even the simplest problems. Imagine a simple two-armed bandit problem with one arm's rewards $\sim \mathrm{Ber}(1/2)$ and the other's $\sim \mathrm{Ber}(1/2+\epsilon)$, and the agent does not know which arm is which a priori. This style of problem is particularly well understood with guarantees that Thompson sampling with more reasonable forms of posterior approximation incur regret $\tilde{O}(\log(T))$ in this setting [2]. We refer to this problem as $\dagger$. The following result highlights that dropout as posterior approximation can perform poorly even on this simple domain.

[6] This corrects an errant derivation in [44], but maintains the same overall message.

Lemma 4 (Dropout sampling attains linear regret in †)**.**
Fix any d, p, λ and consider the problem of † with an agent employing Thompson sampling by dropout (6) *for action selection. Then the expected regret* $\mathbb{E}\left[\mathrm{Regret}(T)\right] = \Omega(T)$.

Proof. For any d, p, λ and any observed data $\mathcal{H}_t$, there exists a non-zero probability $P_1(s, p, \lambda, \mathcal{H}_t) > \frac{p^d}{2}$ of selecting action 1 over action action 2. We can see this by imagining all units estimating action 2 are set to zero, then there is at least 50% chance of selection action 1. This proves[7] that $\mathbb{E}\left[\mathrm{Regret}(T)\right] \geq \frac{\epsilon p^d}{2} T$ for all T. □

Although our analysis of dropout has focused on an exceedingly simple functional form, the key insight that the degree of variability in the posterior distribution does not concentrate with data extends to any neural network architecture. Figure 8 presents the dropout posterior on a simple regression task with a (20,20)-MLP with rectified linear units. We display the predictive distribution under varying amounts of data. The dropout sampling distribution does not converge with increasing amounts of data, whereas the bootstrapped sampling approach behaves much more reasonably.

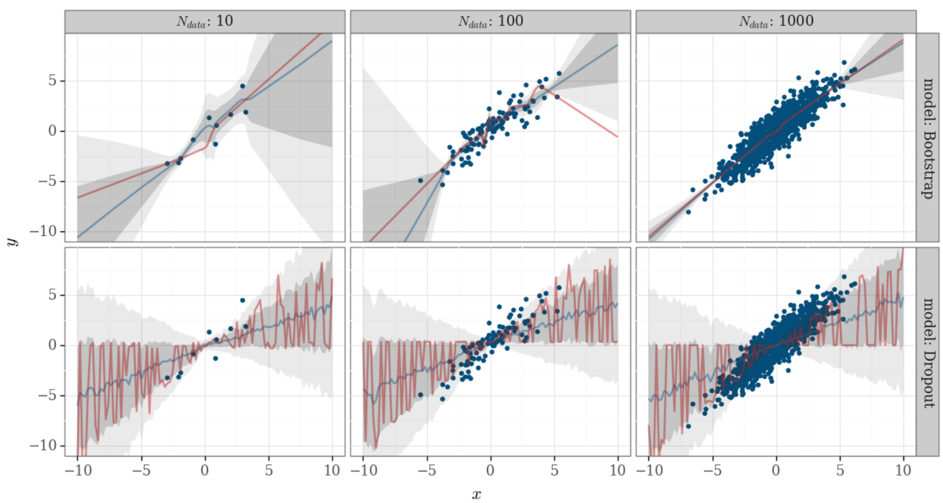


Figure 8: Dropout does not converge with increasing data even with a complex neural network. Grey regions indicate $\pm 1, 2$ standard deviations, the mean is shown in blue and a single posterior sample in red.

Unsurprisingly, the failings that arise in simple estimation problems also cause problems for deep RL applications of dropout for posterior inference. In Appendix C we present results of dropout used within a deep RL problem designed to necessitate efficient exploration, in which dropout performs poorly as well.

A.2 Variational inference

Section 2.2 highlights two main issues with common applications of variational inference to deep reinforcement learning. First, the space of prior functions is typically factorized to a simplistic independent Gaussian representation on the weights. The independence assumptions over weights are clearly false, since most neural networks offer an exponential number of equivalent parameterizations simply by relabeling. Prior elicitation can be a difficult field in the best of times, but specifying a meaningful prior over a deep neural network with millions of weights is close to impossible, even in situations where a reasonable prior over *output functions* can easily be estimated. We argue that this makes them ill-suited for online decision problems without the ability to carefully tune hyperparameters.

[7]Note that this lower bound is very conservative and provided only for illustration. A more precise analysis would show poor performance even for large d.

A more problematic observation comes from Lemma 2, which highlights that the basic loss most commonly used by variational approximations to the value function are fundamentally ill-suited to the problem at hand [33, 16]. In Appendix C we present results of such a variational approach, NoisyNet, to some of our benchmark reinforcement learning tasks. As expected, the algorithm performs poorly even after extensive tuning. At the heart of this issue is a sample-based loss that trains to match the *expectation* of the target distribution, but does not attempt to match the higher moments of the uncertainty. However, we could imagine an alternative approach that does aim to match the entire resultant distribution, for example via parameterized distribution and cross entropy loss; we leave this to future work.

A.3 'Distributional reinforcement learning'

Unlike the objections of Appendix A.2, 'distributional RL'[8] does learn a value function estimate through a distributional loss. However, this distribution is a distribution over *outcomes* and not a distribution over the *epistemic uncertainty* in the mean beliefs. This distinction between two types of uncertainty, (1.) things that you don't know and (2.) things that are stochastic, is a delicate one and is important to characterize correctly. Both are discussed under many names:

1. 'Reducible uncertainty' $\iff$ 'epistemic uncertainty' $\iff$ 'uncertainty',
2. 'Irreducible uncertainty' $\iff$ 'aleatoric uncertainty' $\iff$ 'risk'.

Typical decision problems may include elements of both types of uncertainty. Flipping a coin we might want to know both (1) our posterior beliefs over the probability of heads and (2) a distribution that categorizes the likely possible outcomes. However, it should be clear that the two concepts are fundamentally distinct. For the purposes of exploration, the Bayesian uncertainty over (1) should prioritize the acquisition of new knowledge. 'Distributional RL' approximates (2) and its role is not exchangeable with (1).

Lemma 5 (Using 'distributional RL' as a posterior can lead to arbitrarily bad decisions)**.** *Consider an agent with full information that decides between action* 1 *with reward* $\sim \mathrm{Ber}(0.5)$ *and action* 2 *with reward* $1-\epsilon$ *for* $0 < \epsilon < \frac{1}{2}$*. If the agent employs Thompson sampling correctly then it will pick action* 2 *at every step with zero regret. If the agent mistakenly employs Thompson sampling over its 'distributional value function' then it will incur*

$$\mathbb{E}\left[\mathrm{Regret}(T)\right] \geq \frac{1}{2}\left(\frac{1}{2} - \epsilon\right) T.$$

Lemma 5 shows that using the 'distributional' value function approximating (2.) can be a poor proxy for the Bayesian uncertainty. However, the Bayesian uncertainty can be a similarly poor proxy for the 'distributional' value function. This can be equally damaging, particularly if the agent has some some risk-sensitive utility with respect to cumulative rewards. It is entirely possible to combine both notions of uncertainty in an agent, although for the goal of maximizing expected cumulative it is not entirely clear what is the benefit of modeling (2.). Certainly, 'distributional' agents have recently attained strong scores in Atari 2600 benchmarks but it is so far unclear exactly what the source of this outperformance comes from [7, 13]. Possible explanations may include more stable gradients, bounded values and the 'many predictions' hypothesis [62]: that learning a distribution may effectively create a series of auxiliary losses. We leave these questions for future work.

A.4 Count-based uncertainty estimates

Count-based approaches to exploration give a bonus to states that have not been visited frequently according to some density measure $p(x)$. These methods have performed well in many sparse reward tasks such as Montezuma's revenge, where visiting new states acts as a shaping reward for the true reward [6]. However, a count-based bonus is generally a poor approach to exploration beyond the tabular setting. To see why this is the case note that the

[8] Any method for Bayesian RL might reasonably claim to be a distributional perspective on reinforcement learning. For this reason, we use quotation marks when we want to distinguish the specific form of distributional RL popularized by [6].

density measure of the states may not correlate well with the agent's uncertainty over the optimal policy in that state. We can imagine situations both where the state is visually new, but an agent should still know exactly what to do; and also settings that are only delicately different to a common situation but still necessitate exploration of the optimal policy.

This disconnect shows up in problems as simple as linear bandit.[9] Via a packing argument we can see that an agent with count-based uncertainty will require $\tilde{O}\left(\frac{1}{\epsilon^d}\right)$ measurements to cover the space up to radius ϵ. By contrast, an agent that explores this space efficiently can resolve its uncertainty in only $\tilde{O}(d)$ measurements. Thompson sampling with a linear model naturally recovers this performance [55]. The fact that this failure can arise even in a linear system, and even when the density can be estimated precisely, suggests that count-based exploration is not *in general* an effective method for simultaneous exploration with generalization; even if it may be effective at some specific tasks.

A.5 Ensembles without priors

This paper builds upon a line of research that uses an ensemble of trained models to approximate a posterior distribution. Compared to previous works, our main contribution is to highlight the importance of a 'prior' mechanism in ensemble uncertainty. Figure 9 presents an extremely simple example of 1D regression with a (20,20)-MLP and rectified linear units. The data consists of $x_i = \frac{i-5}{5}$ for $i = 0, .., 10$ and $y_i = 5\mathbb{1}\{i = 10\}$.

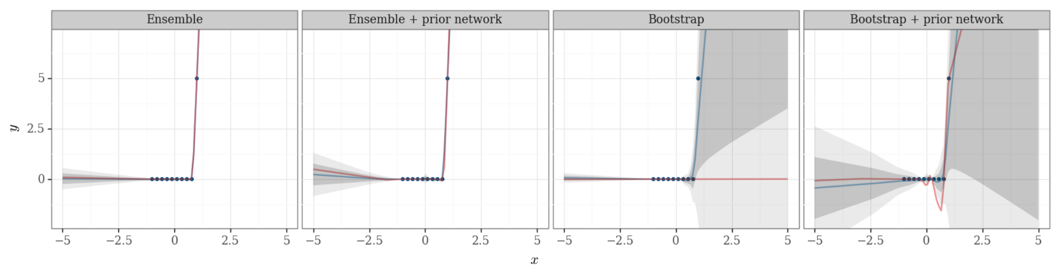


Figure 9: Posterior predictive distributions for ensemble uncertainty.

The results above highlight the drawbacks of naive ensembles. A pure ensemble trained from random initializations fits the data exactly and leads to almost zero uncertainty anywhere in the space [29]. A bootstrapped ensemble takes the variability of the data into account and thus has a wide predictive uncertainty as x grows large and positive. However, where the data has target value zero, bootstrapping will always produce a zero target and consequently the ensemble has almost zero predictive uncertainty as x becomes large and negative [45].

This lack of prior uncertainty can lead to arbitrarily poor decisions, as outlined in [48]. If an agent has only ever observed zero reward, then no amount of bootstrapping or ensembling will cause it to simulate positive rewards. This issue is easily remedied by the addition of a prior mechanism, either through l_2 regularization to initial random weights (4), or the addition of a fixed additive random 'prior network' (5).

A.6 Summary

We summarize the issues raised in Section 2 in Table 1. This table is meant only as a rough summary and should not be taken as rigorous statement. Roughly speaking, a green tick means success, red cross means failure and a yellow circle means something in between. This paper proposes a combination of bootstrap sampling with prior function as an effective computational approximation to Bayesian inference in deep RL. Although our method is somewhat computationally expensive, since it requires training an ensemble of models instead of one, this computation can be done in parallel and so is amenable to large scale distributed computation.

[9] Reward $r_t(x_t) = x_t^T \theta^* + \epsilon_t$ for some $\theta^* \in \mathbb{R}^d$ and $\epsilon_t \sim N(0, 1)$ [54].

Table 1: Important issues in posterior approximations for deep reinforcement learning.

	Data conc.	Learned metric	Multi step	Works in noise	Prior effect	Cheap compute
Dropout [19]	✗	✓	✗	●	✗	✓
NoisyNet [16]	●	✓	✗	✓	✗	✓
BBB / VI [10]	●	✓	✗	✓	●	✓
Density count [6]	✓	✗	✓	✗	●	✓
'Distributional' RL [7]	✗	●	✓	●	✗	✓
Ensemble [29]	✗	✓	✓	✗	✗	✗
Bootstrap [45]	✓	✓	✓	✓	✗	✗
Bootstrap + prior	✓	✓	✓	✓	✓	✗
Exact Bayes	✓	✓	✓	✓	✓	✗✗✗

B Reinforcement learning algorithm

Below we present Bootstrapped DQN with prior networks. Note that this scheme can be combined with numerous algorithms; for example, combining with UCB Q-Ensembles [11] simply requires setting $p = 1$ and using UCB rather than Thompson sampling in lines 4 & 6 in Algorithm 14. In all cases Q_k and P_k are functions with the same signature[10], of the form $\mathcal{S} \times A \to \mathbb{R}$. In the case of UCB Ensembles, the Q-function is optimistically generated from the ensemble via the upper confidence interval:

$$Q_{\text{UCB}}(s_t, a) = \mathbb{E}[Q_k(s_t, a)] + \beta \ \text{StdDev}[Q_k(s_t, a)].$$

For Thompson sampling we draw a Q-function at random from the ensemble:

$$Q_{\text{TS}} \sim \text{Uniform}\left(\{Q_k\}_{k=1}^K\right)$$

Algorithm 2 Bootstrapped DQN with prior networks.

Require: Ensemble of value function networks $\{Q_k\}_{k=1}^K$, target networks $\{Q'_k\}_{k=1}^K$, and prior networks $\{P_k\}_{k=1}^K$; prior scale β; replay buffer B; RL update rule `do_update`.

1: **for** each episode **do**
2: Obtain initial state s_0 from environment
3: Sample a value function $k \sim \text{Uniform}(1, \ldots, K)$ ▷ Thompson sampling
4: **for** $t = 1, 2, \ldots$ until end of episode **do**
5: Select action $a_t \in \arg\max_a \left(Q_k(s_t, a) + \beta P_k(s_t, a)\right)$
6: Receive state s_{t+1} and reward r_t from environment
7: Sample bootstrap mask $m_t \sim \text{Bern}(p)^K$
8: Add $(s_t, a_t, r_{t+1}, s_{t+1}, m_t)$ to replay buffer B
9:
10: Sample batch of N transitions $\left\{b^{(i)} = \left(s_t^{(i)}, a_t^{(i)}, r_{t+1}^{(i)}, s_{t+1}^{(i)}, m_t^{(i)}\right)\right\}_{i=1}^N$ from B
11: Compute TD error $\delta^{(i)} = \texttt{do\_update}(b^{(i)})$
12: Do SGD on masked TD error $\delta^{(i)} \odot m^{(i)}$ using $Q'_k(s_t, a) + \beta P_k(s_t, a)$ as target
13: **end for**
14: **end for**

C Reinforcement learning experiments

In this appendix, we expand on details for the experimental set-up together with some additional results. Unless otherwise stated, we train using the Adam optimizer [26] with TensorFlow defaults using the DQN learning rule with single-step temporal-difference bootstrap [40], and uniformly random sampling from experience replay with a batch size of 128. For

[10] In practice Q_k and P_k are implemented as functions of the form $\mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$.

the ϵ-greedy DQN baseline, we anneal epsilon linearly over 2000 episodes and perform hyper-parameter sweeps over the initial epsilon ϵ_0. All other agents (NoisyNet, Dropout, Ensemble, Bootstrap) use greedy policies unless otherwise stated. Naive ensembling corresponds to $p = 1$; for all bootstrapped agents, we use $p = 0.5$.

C.1 Chain environments

In our experiments we use a fully tabular one-hot encoding in which each cell is one if the agent is in that cell and zero otherwise. Alternatively, one could use some variant of a 'thermometer' encoding, in which either all cells above and/or to the left of the agent are one and all other cells are zero. In our experiments neither encoding changed the results, so we use the tabular encoding for simplicity. All agents use a single layer MLP with 20 hidden units, rectified linear activations and Glorot initialization.

Figure 3 shows the time it takes each agent to learn a problem of size N. Figure 10 reproduces these results but on a log-log scale, which helps to reveal the problem scaling as N increases. As in Figure 3, the dashed line corresponds to a dithering lower bound $T_{\text{learn}} = 2^N$. We also include a solid line with slope equal to three, corresponding to a polynomial growth $T_{\text{learn}} = \tilde{O}(N^3)$.

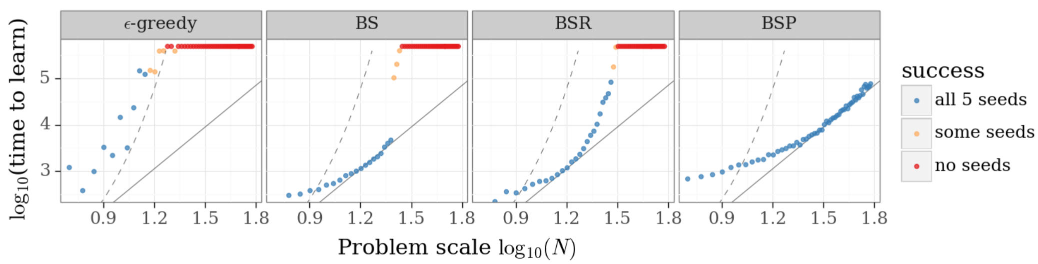


Figure 10: Log-log plot demonstrates scaling of learning behaviour.

In addition to BSP, BSR, BS and ϵ-greedy displayed in Figure 3, we also ran parameter sweeps for dropout, NoisyNet and a count-based exploration strategy. Figure 11 presents the result for NoisyNet and dropout, each individually tuned up to 50k episodes. Even after tuning dropout rate and sampling frequency (by episode or by timestep) neither dropout nor NoisyNet scale successfuly to large domains.

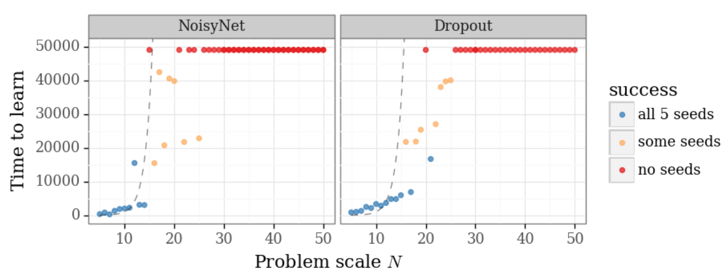


Figure 11: Learning time for noisy and dropout; neither approach scales well.

For comparison to count-based exploration we implement a version of DQN that optimizes the true reward plus a UCB exploration bonus $\frac{\beta}{\sqrt{N_t(s)}}$, where $N(s)$ is the number of visits to state s prior to time t [24, 6]. Figure 12 shows that this count-based exploration strategy performs much worse than BSP, even after sweeping over bonus scale β and even with access to the true state visit-counts. This mirrors the outperformance of PSRL vs UCRL in tabular reinfocement learning. One explanation for this discrepancy comes from the inefficient way UCB-style algorithms propagate uncertainty over many timestep [49, 42].

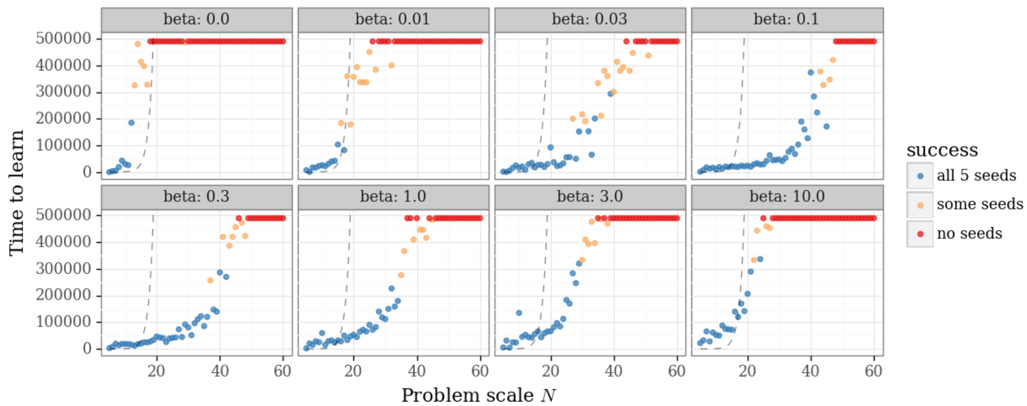


Figure 12: Sweeping over optimistic bonus; no scale of β matches BSP performance.

For all of our algorithms we tune agent hyperparameters by grid search. For completenes we present the optimized parameters:

- **ϵ-greedy**: $\epsilon = 0.1$, linearly annealed to zero.
- **BSP**: prior scale $\beta = 10$ (Figure 4b).
- **BSR**: l_2 regularizer scale $\lambda = 0.1$ (Figure 4a).
- **Dropout**: Resample mask every step with $p_{\text{keep}} = 0.1$.
- **NoisyNet**: Resample noise every step.
- **UCB**: Optimistic bonus $\beta = 0.1$ (Figure 12).

C.2 Sparse cartpole swing-up

In Section 4.2 we presented experiments showing that BSP outperforms benchmark algorithms. Figure 13 presents the sensitivity of BSP sensitivity to the prior scale β on this domain. Small values of β prematurely and suboptimally converge to the stationary policy, and so receive zero cumulative reward. Larger values of β take longer to wash away their prior effect, but we expect them to learn a performant policy eventually. This behaviour mirrors the scaling we saw in the chain environments, which is reassuring.

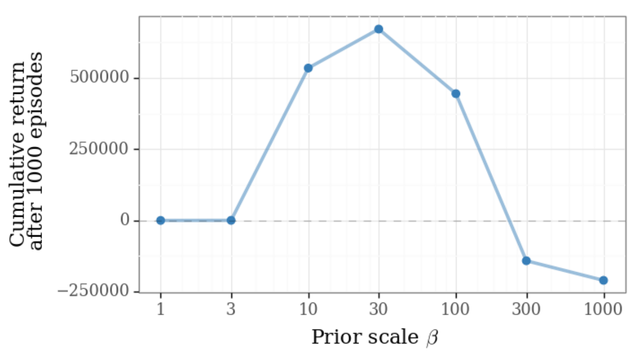


Figure 13: Sensitivity of performance to prior scale β.

C.3 Montezuma's revenge

In our experiments, we use the standard Atari configuration and preprocessing including greyscaling, frame stacking, action repeats, and random no-op starts [40], and the same agent hyperparameters as those used in the Ape-X paper [23]. However, our agent implementation is somewhat different and so our baseline results are not directly comparable across all games.