Approximate Thompson Sampling via Epistemic Neural Networks

Ian Osband[1] **Zheng Wen**[1] **Seyed Mohammad Asghari**[1] **Vikranth Dwaracherla**[1]
Morteza Ibrahimi[1] **Xiuyuan Lu**[1] **Benjamin Van Roy**[1]

[1]Efficient Agent Team, DeepMind , Mountain View, CA

Abstract

Thompson sampling (TS) is a popular heuristic for action selection, but it requires sampling from a posterior distribution. Unfortunately, this can become computationally intractable in complex environments, such as those modeled using neural networks. Approximate posterior samples can produce effective actions, but only if they reasonably approximate joint predictive distributions of outputs across inputs. Notably, accuracy of marginal predictive distributions does not suffice. Epistemic neural networks (ENNs) are designed to produce accurate joint predictive distributions. We compare a range of ENNs through computational experiments that assess their performance in approximating TS across bandit and reinforcement learning environments. The results indicate that ENNs serve this purpose well and illustrate how the quality of joint predictive distributions drives performance. Further, we demonstrate that the *epinet* — a small additive network that estimates uncertainty — matches the performance of large ensembles at orders of magnitude lower computational cost. This enables effective application of TS with computation that scales gracefully to complex environments.

1 INTRODUCTION

Thompson sampling (TS) is one of the oldest heuristics for action selection in reinforcement learning [Thompson, 1933, Russo et al., 2018]. It has also proved to be effective across a range of environments [Chapelle and Li, 2011]. At a high level, it says to 'randomly select an action, according to the probability it is optimal.' This approach naturally balances exploration with exploitation, as the agents favours more promising actions, but does not disregard any action that has a chance of being optimal. However, in its exact form, TS requires sampling from a posterior distribution, which becomes computationally intractable for complex environments [Welling and Teh, 2011].

Approximate posterior samples can also produce performant decisions [Osband et al., 2019]. Recent analysis has shown that, if a sampled model is able to make reasonably accurate *predictions* it can drive good decisions [Wen et al., 2022]. But these results stress the importance of *joint* predictive distributions — or joint predictions, for short. In particular, accurate marginal predictive distributions do not suffice.

Epistemic neural networks (ENNs) are designed to make good joint predictions [Osband et al., 2021]. ENNs were introduced with a focus on classification problems, but we will show in this paper that the techniques remain useful in producing regression models for decision making. This paper empirically evaluates the performance of approximate TS schemes that use ENNs to approximate posterior samples. We build upon *deep Q-networks* [Mnih et al., 2015], but using ENNs to represent uncertainty in the state-action value function.

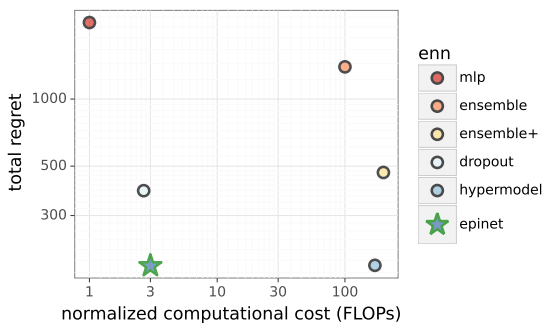


Figure 1: Performance of an approximate TS agent in a neural bandit using different ENNs. Epinet beats large ensembles at fraction of computational cost (Section 5).

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

Figure 1 offers a preview of our results. Among ENNs we consider are ensembles of base models [Osband and Van Roy, 2015, Lakshminarayanan et al., 2017] and a single base model enhanced with the recently proposed epinet, which is a small additive network that estimates uncertainty. **We find that, using an epinet, we can outperform large ensembles at orders of magnitude lower computational cost**. More generally, we find that ENNs that produce better joint predictions in synthetic classification problems also perform better in decision problems.

1.1 KEY CONTRIBUTIONS

We introduce ENN-DQN, which unifies algorithms that combine DQN and approximate TS. **We release open-source library for all our experiments at `enn_acme`** (Section 4). This provides a valuable resource for clear and reproducible research in the field and the first extensive investigation into the effectiveness of posterior samples in deep RL. Our work builds on the existing `acme` library for RL [Hoffman et al., 2020].

We demonstrate a clear empirical relationship between quality of joint predictions produced by an ENN and the performance of resulting decisions. ENNs that offer better joint prediction tend to produce better decisions in our benchmark tasks. Interestingly, this is true not only for bandit environments of the neural testbed [Osband et al., 2022], but also in `bsuite` benchmark reinforcement learning tasks designed to highlight key aspects of decision making [Osband et al., 2020].

Importantly, **we show that epinets outperform large ensembles, but at orders of magnitude lower computational cost.** This holds even for regression models, as in temporal difference (TD) learning, not just classification. These results are significant since prior work on ENNs had focused only on the quality of joint predictions [Osband et al., 2021]. We show that these results also extend to empirical decision making with deep learning systems.

1.2 RELATED WORK

This paper builds on a long literature around TS for efficient exploration [Thompson, 1933, Lai et al., 1985, Russo and Van Roy, 2014]. Much of this work has been focused on extending and refining performance guarantees around particular problem classes, where exact Bayesian inference allows for efficient generalization between states and actions. From bandits with structure [Russo and Van Roy, 2013], to MDPs [Osband et al., 2013] or MDPs with generalization [Osband and Van Roy, 2014b,a, Gopalan and Mannor, 2015]. However, in complex environments, even planning with full information may be intractable [Silver et al., 2016]. For this reason, so-called deep reinforcement learning (RL) algorithms use neural networks to directly assess the value and/or policy functions [Mnih et al., 2015]. Most of these schemes employ simple dithering schemes for exploration, such as epsilon-greedy or boltzmann exploration. There are relatively few approximate TS schemes that have modified these algorithms to attempt to combine the best of this deep RL with so-called 'deep exploration' [Osband et al., 2019].

Bootstrapped DQN [Osband et al., 2016] maintains an ensemble of networks as a proxy for neural network uncertainty, but this is just one particular approach popular in the Bayesian deep learning community. Other popular approaches include dropout [Gal and Ghahramani, 2016], variational inference [Blundell et al., 2015], or even stochastic Langevin MCMC [Welling and Teh, 2011]. However, research in this area has focused mainly on supervised learning tasks [Izmailov et al., 2021], with relatively little attention paid to the use of these Bayesian network in driving effective decision making. Our work fits into a wider literature of approximate Thompson sampling approaches for deep reinforcement learning [Zhang et al., 2020, Dai et al., 2022].

2 PROBLEM FORMULATION

This section outlines the notation and problem setting. We begin with a review of the family of sequential decision problems we will consider. Next, we provide a quick overview on epistemic neural networks, which can make joint predictions without being Bayesian. Finally, we introduce the ENN-DQN variant that allows for an approximate of Thompson sampling.

2.1 REINFORCEMENT LEARNING

We consider the problem of learning to optimize a random finite-horizon Markov decision problem (MDP) $M^*=(\mathcal{S},\mathcal{A},R^*,P^*,\bar{s},\rho)$ over repeated episodes of interaction, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\bar{s} \in \mathcal{S}$ is the terminal state, and ρ is the initial state distribution. At the start of each episode the initial state s_1 is drawn from the distribution ρ. In each time period $h = 1, 2, ...$ within an episode, the agent observes a state $s_h \in \mathcal{S}$. If $s_h \neq \bar{s}$, the agent also selects an action $a_h \in \mathcal{A}$, receives a reward $r_{h+1} \sim R^*(\cdot|s_h, a_h)$, and transitions to a new state $s_{h+1} \sim P^*(\cdot|s_h, a_h)$. An episode terminates once the agent arrives at the terminal state $\bar{s}$. We use H to denote the horizon of an episode. Note that H is a random variable in general[1]

[1]More precisely, H is a stopping time.

and the agent arrives at $\bar{s}$ in period $H+1$. The agent is given knowledge about $\mathcal{S}$, $\mathcal{A}$, $\bar{s}$, and ρ, but is uncertain about R^* and P^*. The unknown MDP M^*, together with reward function R^* and transition function P^*, are modeled as random variables [Lu et al., 2021].

A policy $\mu : \mathcal{S} \to \mathcal{A}$ maps a state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$. For each MDP M with state space $\mathcal{S}$ and action space $\mathcal{A}$, and each policy μ, we define the associated state-action value function as:

$$Q_\mu^M(s,a) := \mathbb{E}_\mu \left[\sum_{h=1}^H r_{h+1} \middle| s_1 = s, a_1 = a, M^* = M \right], \tag{1}$$

where the subscript μ next under the expectation is a shorthand for indicating that actions over periods $h=2,\ldots,H$ are selected according to the policy μ. Let $V_\mu^M(s) := Q_\mu^M(s,\mu(s))$. We say a policy μ^M is optimal for the MDP M if $\mu^M(s) \in \arg\max_\mu V_\mu^M(s)$ for all $s \in \mathcal{S}$. To simplify the exposition, we assume that under any MDP M and any policy μ, $H < \infty$ with probability 1.

We use k to index the episode, and we use $\mathcal{H}_k$ to denote the history of observations made *prior* to episode k. An RL algorithm is a deterministic sequence of functions, $\{\pi_k | k = 1, 2, \ldots\}$, each mapping $\mathcal{H}_k$ to a probability distribution $\pi_k(\cdot|\mathcal{H}_k)$ over policies, from which the agent samples a policy μ_k for the k^{th} episode. Denote the regret of a policy μ_k over episode k by

$$\Delta_k := \textstyle\sum_{s \in \mathcal{S}} \rho(s)(V_{\mu^*}^{M^*}(s) - V_{\mu_k}^{M^*}(s)), \tag{2}$$

where μ^* is an optimal policy for M^*. We define the expected regret incurred by an RL algorithm π up to episode K as

$$\text{Regret}(K, \pi) := \mathbb{E}_\pi \left[\textstyle\sum_{k=1}^K \Delta_k \right], \tag{3}$$

where the subscript π under the expectation indicates that policies are generated through algorithm π. Note that the expectation in (3) is over the random transitions and rewards, the possible randomization in the learning algorithm π, and also the unknown MDP M^* based on the agent designer's prior distribution.

2.2 EPISTEMIC NEURAL NETWORKS

We construct RL agents based on epistemic neural networks (ENN) [Osband et al., 2021]. A conventional neural network is specified by a parameterized function class f, which produces an output $f_\theta(x)$ given parameters θ and an input x. An ENN is specified by a parameterized function class f *and* a reference distribution P_Z. The output $f_\theta(x, z)$ of an ENN depends additionally on an *epistemic index* z, sampled from the reference distribution P_Z. Variation of the network output with z indicates uncertainty that might be resolved by future data. All conventional neural networks can be written as ENNs, but this more general framing allows an ENN to represent the kinds of uncertainty necessary for effective sequential decision-making [Wen et al., 2022]. In particular, it allows for an ENN to represent useful joint predictions.

Consider a classification problem. Given inputs $x_1, \ldots, x_\tau$, a joint prediction assigns a probability $\hat{P}_{1:\tau}(y_{1:\tau})$ to each class combination $y_1, \ldots, y_\tau$. Importantly, and unlike 'standard' deep learning classification problems, these predictions are made *jointly* and not independently for each input. For insight into how this distinction can be critical for decision making we refer to Wen et al. [2022], Osband et al. [2022].

Epistemic neural networks output class logits for each input, but can also express nuanced joint predictions by integrating over the epistemic index.

$$\hat{P}_{1:\tau}^{\text{ENN}}(y_{1:\tau}) = \int_z P_Z(dz) \prod_{t=1}^{\tau} \text{softmax}\left(f_\theta(x_t, z)\right)_{y_t}. \tag{4}$$

This sort of nuanced joint prediction share many similarities with Bayesian neural networks (BNNs), which maintain a posterior distribution over plausible neural nets. However, unlike BNNs, ENNs do not necessarily ascribe Bayesian semantics to the unknown parameters of interest, and they do not generally update with Bayes rule. All BNNs can be expressed as ENNs; for example, an ensemble of K networks $f_{\theta_1}, .., f_{\theta_K}$ can be written as an ENN $\tilde{f}$ with reference distribution $P_Z = \text{Unif}(\{1, .., K\})$ and $\tilde{f}_\theta(x, z) := f_{\theta_z}(x)$ [Osband and Van Roy, 2015, Lakshminarayanan et al., 2017]. However, there are some ENNs that cannot be expressed naturally as BNNs.

2.3 THE EPINET

One such example of novel ENNs is the *epinet*: a small additional network designed to estimate uncertainty [Osband et al., 2021]. An epinet is added to a *base network*: a conventional NN with base parameters ζ that takes input x and outputs $\mu_\zeta(x)$. The epinet acts on a subset of *features* $\phi_\zeta(x)$ derived from the base network, as well as an epistemic index z sampled from the standard normal in D_Z dimensions. For concreteness, you might think of μ as a large neural network and ϕ as the last layer features. For epinet parameters η, this produces a combined output:

$$\underbrace{f_\theta(x,z)}_{\text{ENN}} = \underbrace{\mu_\zeta(x)}_{\text{base net}} + \underbrace{\sigma_\eta(\text{sg}[\phi_\zeta(x)], z)}_{\text{epinet}}. \tag{5}$$

The ENN parameters $\theta = (\zeta, \eta)$ include those of the base network and epinet[2]. The epinet σ_η has a simple

[2]The "stop gradient" notation sg[·] indicates the argument is treated as fixed when computing a gradient. For

MLP-like architecture, with an internal *prior function* designed to create an initial variation in index z [Osband et al., 2018]. That means, for $\tilde{x} := \text{sg}[\phi_\zeta(x)]$,

$$\underbrace{\sigma_\eta(\tilde{x}, z)}_{\text{epinet}} = \underbrace{\sigma_\eta^L(\tilde{x}, z)}_{\text{learnable}} + \underbrace{\sigma^P(\tilde{x}, z)}_{\text{prior net}}. \quad (6)$$

The prior network σ^P represents prior uncertainty and has no trainable parameters. The learnable network σ_η^L can adapt to the observed data with training. We refer readers to the original paper of Osband et al. [2018] for an overview of this prior function technique.

This paper focuses on simple neural networks based around MLPs with ReLU activation. Let C denote the number of classes and D_Z denote the index dimension. The learnable network $\sigma_\eta^L(\phi_\zeta(x), z) = g_\eta([\phi_\zeta(x), z])^T z$, where $g_\eta(\cdot)$ is an MLP with outputs in $\mathbb{R}^{D_Z \times C}$, and $[\phi_\zeta(x), z]$ is concatenation of $\phi_\zeta(x)$ and z. The prior network σ^P is a mixture of an ensemble of D_Z particles sampled from the distribution of the data generating model that acts directly on the input x (Section 4).

2.4 ENN-DQN

We now motivate and develop ENN-DQN, a novel DQN-type agent for large-scale RL problems with value function approximation. Specifically, it uses an ENN to maintain a probability distribution over the state-action value function Q^*, which may be thought of as an approximate posterior of the optimal state-action value function. We consider ENNs $f_\theta(s, a) \in \Re^{|\mathcal{A}|}$ that take a state and an epistemic index, and output a real value for each action in $\mathcal{A}$, similar to an DQN. ENN-DQN selects actions using Thompson sampling (TS). It can be viewed as a value-based approximate TS algorithm via ENN.

Similar to existing work on ENNs [Osband et al., 2022], the agent needs to define a loss function to update the ENN parameters. In general, for a given ENN f_θ, a *target ENN* $f_{\theta^{\text{target}}}$, and an observed dataset $\mathcal{D}$, the agent updates its ENN of the state-action value function by minimizing

$$\mathcal{L}(\theta, \theta^{\text{target}}, \mathcal{D}) = \mathbb{E}_{z\sim P_Z}\left[\sum_{d\in\mathcal{D}} \ell(d, z; \theta, \theta^{\text{target}})\right] + \psi(\theta), \quad (7)$$

where $\ell(d, z; \theta, \theta^{\text{target}})$ is the loss associated with the observed transition $d = (s, a, r, s')$ as well as the epistemic index z, and $\psi(\theta)$ is a regularization term. In this paper we use $\psi(\theta) = \lambda\|\theta\|_2^2$ for some $\lambda > 0$, which corresponds to a Gaussian prior over θ. We will discuss the specific choices of ℓ at the end of this section. Note that the target ENN is necessary for the stability of learning in many problems, as discussed in [Mnih et al., 2015].

We optimize $\mathcal{L}$ through stochastic gradient descent. At each gradient step, we sample a mini-batch of data $\tilde{\mathcal{D}}$ and a batch of indices $\tilde{\mathcal{Z}}$ from P_z, and we take a gradient step with respect to the loss

$$\tilde{\mathcal{L}}(\theta, \theta^{\text{target}}, \tilde{\mathcal{D}}, \tilde{\mathcal{Z}}) = \frac{|\mathcal{D}|}{|\tilde{\mathcal{D}}|}\frac{1}{|\tilde{\mathcal{Z}}|}\sum_{z\in\tilde{\mathcal{Z}}}\sum_{d\in\tilde{\mathcal{D}}} \ell(d, z; \theta, \theta^{\text{target}}) + \psi(\theta). \quad (8)$$

Algorithm 1 describes the ENN-DQN agent. Specifically, at each episode k, the agent samples an epistemic index z_k and takes actions greedily with respect to the associated state-action value function $f_\theta(\cdot, z_k)$. The agent updates the ENN parameters θ in each episode according to (8), and it updates the target parameters θ^{target} periodically.

Algorithm 1 ENN-DQN agent

Input: initial parameters θ_0, ENN for action-value function $f_\theta(s = \cdot, z = \cdot)$ with reference distribution P_Z.

$\theta^{\text{target}} \leftarrow \theta_0$
initialize buffer
for episode $k = 1, 2, \ldots$ **do**
  sample index $z_k \sim P_z$
  $h \leftarrow 1$
  observe $s_{k,1}$
  while $s_{k,h} \neq \bar{s}$ **do**
    apply $a_{k,h} \in \arg\max_a f_\theta(s_{k,h}, z_k)_a$
    observe $r_{k,h+1}, s_{k,h+1}$
    buffer.add($(s_{k,h}, a_{k,h}, r_{k,h+1}, s_{k,h+1})$)
    $\theta,\ \theta^{\text{target}} \leftarrow$ update(buffer, $\theta, \theta^{\text{target}}$)
    $h \leftarrow h + 1$

Finally, we discuss the choices of data loss function ℓ. Note that the choices of ℓ are usually problem-dependent. For bandit problems with discrete rewards, such as either the finite Bernoulli bandits we consider in Section 3, or the neural bandit we consider in Section 5, we use the classic cross-entropy loss. For general RL problems, such as the ones we consider in Section 6, we use the quadratic temporal difference (TD) loss

$$\ell(d, z; \theta, \theta^{\text{target}}) = \left(f_\theta(s, z)_a - r - \gamma \max_{a'} f_{\theta^{\text{target}}}(s', z)_{a'}\right)^2,$$

where $\gamma \in [0, 1]$ is a discount factor chosen by the agent which reflects its planning horizon. Our next section examines the performance of this style of agent in a simplistic decision problem.

example, $\nabla_\theta f_\theta(x, z) = [\nabla_\zeta \mu_\zeta(x), \nabla_\eta \sigma_\eta(\phi_\zeta(x), z)]$.

3 ANALYSIS IN BANDITS

The quality of decision-making in RL relies crucially on the quality of *joint* predictions. As established in [Wen et al., 2022], accurate *joint* predictions are both necessary and sufficient for effective decision-making in bandit problems. To help build intuition, we present a simple, didactic bandit example in this section.

Example 1 (Bandit with one unknown action)**.** *Consider a bandit problem with A actions. The rewards for actions $1, .., A-1$ are known to be independently drawn from Bernoulli(0.5). The final action A is deterministic, but either rewards 0 or 1 and both environments are equally likely.*

The optimal strategy to maximize the cumulative reward in Example 1 is to first select the uncertain action A and, if that is rewarding, then pick that one for all future timesteps, otherwise default to any of the first $1, .., A-1$. Exact Thompson sampling algorithm will incur an $\mathcal{O}(1)$ regret in this example. However, depending on the quality of ENN approximation, approximate TS based on an ENN can sometimes do much worse. To see it, note that action A is indistinguishable from other actions based on marginal predictions. Consequently, any agent making decisions only based on marginal predictions cannot perform better than a random guess and will incur an $\mathcal{O}(A)$ regret in Example 1.

On the other hand, the results of Wen et al. [2022] show that suitably-accurate *joint* predictions, that is predictions over the possible rewards $r_1, .., r_\tau$ for τ time steps into the future *do* suffice to ensure good decision performance of a variant of approximate TS algorithm (see Theorem 5.1 of that paper). Indeed, for Example 1 even $\tau = 2$ will suffice, as the agent can distinguish the informative action A that has all probability on either both rewards being rewarding, or both being non-rewarding if it is selected.

The remainder of this paper will show that the sort of stylized and theoretical issue highlighted by Example 1 actually shows up in real problems with deep reinforcement learning. We will go on to see that the ENNs that perform better *joint* prediction, are the ones that drive more effective decision making.

4 BENCHMARK ENNS

Our results build on open-source implementations of Bayesian deep learning, tuned for performance in the Neural Testbed [Osband et al., 2022]. Table 1 shows the agents we consider. This section will review the key results and evaluation of these agents in Neural Testbed benchmark, then outline the open-source libraries that we release together with our paper submission.

4.1 NEURAL TESTBED

The Neural Testbed sets a prediction problem generated by a random neural network. The generative model is a simple 2-layer MLP with ReLU activations and 50 hidden units in each layer. We outline the agent implementations in Table 1, together with the hyperparameters that were tuned for their performance. Since we are taking open-source implementations we do not re-tune the settings for either testbed or decision problem, except where explicitly mentioned.

For our `epinet` agent, we initialize base network $\mu_\zeta(x)$ as per the baseline `mlp` agent. The agent architecture follows Section 2.3 and we tune the index dimension and hidden widths for performance and compute. After tuning, we chose epinet hidden layer widths $(15, 15)$, with an index dimension of 8 and standard Gaussian reference distribution.

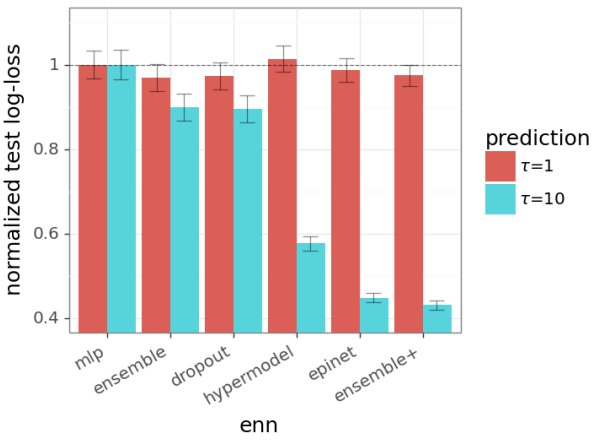


Figure 2: Evaluating quality of marginal and joint predictions on the Neural Testbed.

Figure 2 shows the results of evaluating these benchmark agents on the Neural Testbed in both marginal ($\tau = 1$) and joint ($\tau = 10$) predictions over 10 random seeds, each seed working over many internal generative model instances. After tuning, most of the agents perform similarly in terms of marginal predictions, and are statistically indistinguishable from the well-tuned baseline MLP at 2 standard errors. However, once we look at *joint* predictions, we can see significant differences in agent performance. Importantly, the epinet matches the performance of large ensembles, but at orders of magnitude lower computational cost. In the rest of this paper we will see that this difference in joint prediction is highly correlated with the resultant agent performance in decision problems.

Table 1: Summary of benchmark agents, taken from Neural Testbed [Osband et al., 2022].

agent	description	hyperparameters
`mlp`	Vanilla MLP	L_2 decay
`ensemble`	'Deep Ensemble' [Lakshminarayanan et al., 2017]	L_2 decay, ensemble size
`dropout`	Dropout [Gal and Ghahramani, 2016]	L_2 decay, network, dropout rate
`hypermodel`	Hypermodel [Dwaracherla et al., 2020]	L_2 decay, prior, index dimension
`ensemble+`	Ensemble + prior functions [Osband et al., 2018]	L_2 decay, ensemble size, prior scale
`epinet`	Last-layer epinet [Osband et al., 2021]	L_2 decay, network, prior, index dimension

4.2 OPEN-SOURCE CODE

As part of our research effort we release all code necessary to reproduce our experimental results. These do not require access to specialized hardware, and can be run on typical cloud computing for less than 10 USD. Our code builds principally on two existing opensource libraries `enn` [Osband et al., 2021] and `acme` [Hoffman et al., 2020]. These provide frameworks for ENN and RL agent design, respectively.

To run our experiments on Neural Bandit, we make minor edits to the `neural_testbed` library [Osband et al., 2022], which we anonymize as part of our submission. Our main contribution comes in the `enn_acme` library, that contains the ENN-DQN algorithm, together with the experiments and implementation details. This library allows for simple comparison between different Bayesian (and non-Bayesian) ENNs for use in deep RL experiments. We believe that it will provide a useful base for future research in the area.

5 NEURAL BANDIT

In this section we present an empirical evaluation of the ENNs from Table 1 on a 'neural bandit' problem. We begin by describing the environment, which is derived from the open-source Neural Testbed for evaluating joint predictions [Osband et al., 2022]. Then, we review the agent structure, with the details of the ENN-DQN variant we employ. Finally, we review the results which show that ENNs that perform better in joint prediction tend to drive better decisions.

5.1 ENVIRONMENT

The neural bandit [Osband et al., 2022] is an environment where rewards are generated by neural-network-based generating processes. We take the 2-layer MLP generative model from the Neural Testbed (Section 4). We consider $N = 1000$ actions, drawn i.i.d. from a 100-dimensional standard normal distribution. At each timestep, the reward of selecting an action a is generated by first forwarding the vector a through the MLP, which gives 2 logit outputs. The reward $\in \{0, 1\}$ is then sampled according to the class probabilities obtained from applying softmax to the logits. Our agents reuse the ENN architectures from Section 4 to estimate value functions that predict immediate rewards (i.e. apply discount factor 0). We run the agents for 50,000 timesteps and average results over 30 random seeds.

We consider this problem as a simple sanitised problem where we have complete control over the generative model, but also know that a deep learning architecture is appropriate for inference. We hope that this clean and simple proof of concept can help to facilitate understanding. This problem represents a neural network variant of the finite armed bandit problem of Section 3.

5.2 AGENTS

We run the ENN-DQN agents for all of the ENNs of Table 1. Since the problem is only one timestep we train with the cross-entropy loss on observed rewards. We apply an L_2 weight decay scheme that anneals with $1/N$ for N observed datapoints. As outlined in Table 1 we tune the L_2 decay for each of these agents to maximize performance.

We use a replay buffer of size 10,000 and update the ENN parameters after each observation with one stochastic gradient step computed using a batch of 128 observations from the replay buffer and a batch of i.i.d index samples from P_Z. To compute the gradient, `epinet` agent used a batch of 5 index samples and other agents used the respective default values specified in `https://github.com/deepmind/neural_testbed`. We use Adam optimizer [Kingma and Ba, 2015] with a learning rate of 0.001 for updating the ENN parameters based on the gradient.

5.3 RESULTS

The results of Figure 1 clearly show that, the epinet leads to lower total regret than other ENNs. These results are particularly impressive once you compare the computational costs of the epinet against the other methods. Figure 3 looks at the average regret through time over the 50,000 steps of interaction. We can clearly

see that the epinet leads to better regret at all stages of learning. These results are significant in that they are some of the first to actually show the benefits of epinet in an actual decision problem.

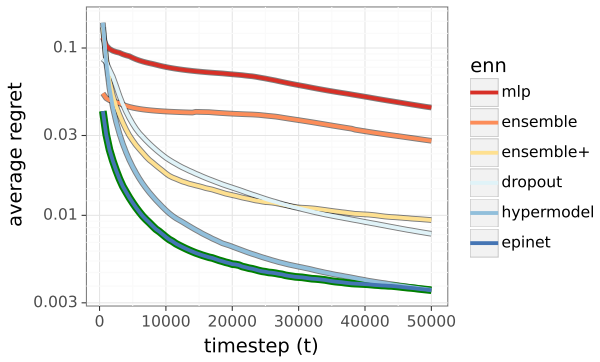
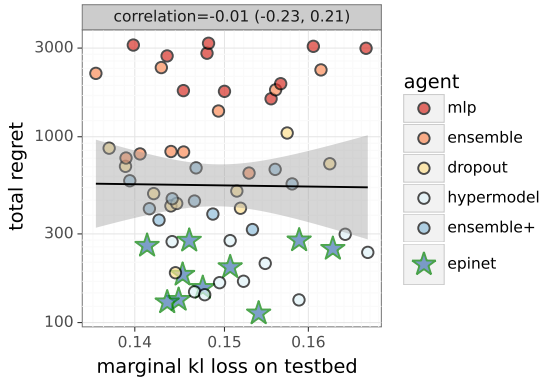
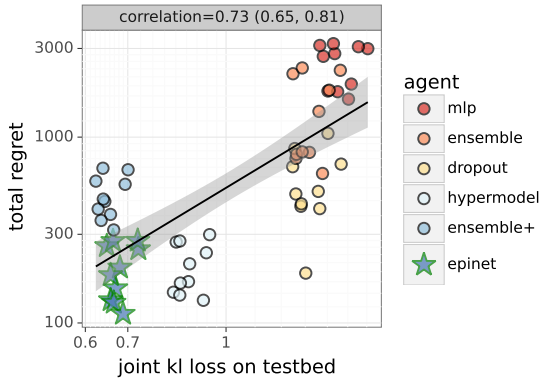


Figure 3: Regret through time for different ENNs.

The scatter plots of Figure 4 report the correlation between prediction quality on the Neural Testbed and bandit performance. The multiple points for any given agent represent results generated with different random seeds. The plot titles provide the estimated correlation, together with bootstrapped confidence intervals at the 5th and 95th percentiles. Concretely, 'correlation=-0.01 (-0.23, 0.21) in Figure 4a means that the correlation is estimated at -0.01, but the bootstrapped distribution of correlation estimates has a 5th percentile at -0.23 and a 95th percentile at 0.21. However, examining the corresponding correlation of 0.73 in Figure 4b, with confidence intervals at (0.65, 0.81) we can see that agents with accurate joint predictions tend to perfom better in the neural bandit. These results mirror the previous results of Osband et al. [2022], but now include the epinet agent, which continues to follow this trend.



(a) Marginal quality is not correlated with performance.



(b) Joint quality is highly correlated with performance.

Figure 4: Relating bandit performance to prediction quality in the Neural Testbed.

6 BEHAVIOUR SUITE FOR RL

This section repeats the evaluation of Section 5, but in reinforcement learning problems with long-term consequences. We review the set of environments and benchmarks included in bsuite [Osband et al., 2020]. Next, we provide implementation details of our ENN-DQN algorithms. Finally, we present the results which, at a high level, mirror those of the bandit setting.

6.1 ENVIRONMENT

The behaviour suite for reinforcement learning, or `bsuite` for short, is a collection of environments carefully-designed to investigate core capabilities of RL agents [Osband et al., 2020]. We repeat our analysis of ENNs applied to these environments. We use the ENNs from Section 5 to estimate value functions with discount $\gamma = 0.99$. For all agents using prior functions (`ensemble+`, `hypermodel`, and `epinet`) we scale the value prior to have mean 0 and variance 1 based on the observations over the first 100 timesteps under a random action policy.

We choose to work with `bsuite` since these are challenging environments designed by RL researchers and *not* given by neural network generative models. In addition, these problems are created with particularly challenging issues in exploration, credit assignment and memory that do not arise in the neural testbed. Evaluating on these extreme, but simple, tasks allows us to stress test our methodology.

6.2 AGENTS

We run the ENN-DQN agents for all of the ENNs of Table 1. All agents use a replay buffer of size 10,000 and update the ENN parameters after each interaction with the environment. Each update consists of taking a step in the direction of the gradient of the loss function, Equation (1), using a batch of 128 observations from the

replay buffer and a batch of 20 i.i.d index samples from the reference distribution. We make use of discount factor $\gamma = 0.99$ for all ENN agents in our experiments.

For `epinet` we use a similar architecture to Section 4 but only a single-hidden layer epinet with 50 hidden units along with a 2-hidden layer MLP base model, 2-dimensional normal Gaussian distribution as the reference distribution.

We use a single set of hyperparameters for all the bsuite environments. However, different bsuite environments have different maximum possible rewards, and a single value of prior scale might not suffice for all the environments. To overcome this, we first run a uniform random action policy, which samples actions with equal probability from the set of possible actions, for 100 time steps. We use this data to scale the output of the prior value functions to have a mean 0 and variance 1 for all the agents which use prior functions (`hypermodel`, `ensemble+`, and `epinet`). The supplementary material presents a detailed breakdown of performance of different agents across environments.

6.3 RESULTS

In `bsuite`, an agent is assigned a score for each experiment. Figure 5 plots the "bsuite loss", which we define to be one minus the average score against computational cost. Once again, epinet performs similarly with large ensembles, but at orders of magnitude less computational cost. Empirically, we observe the biggest variation with ENN design in the 'DeepSea' environments designed to test efficient exploration. Here, only the epinet and ensemble+ agents are able to consistently solve large problem sizes. We include a more detailed breakdown of agent performance by competency in the supplementary material.

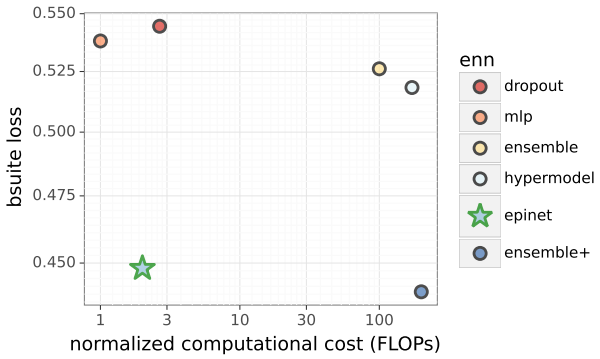
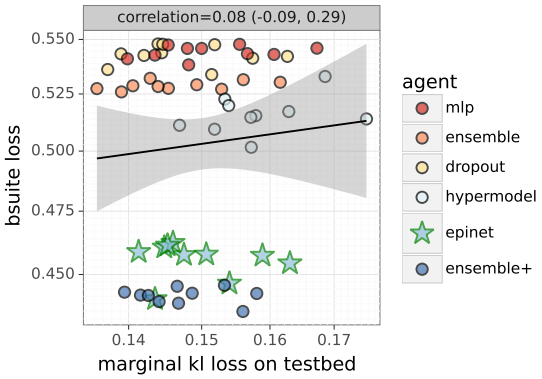


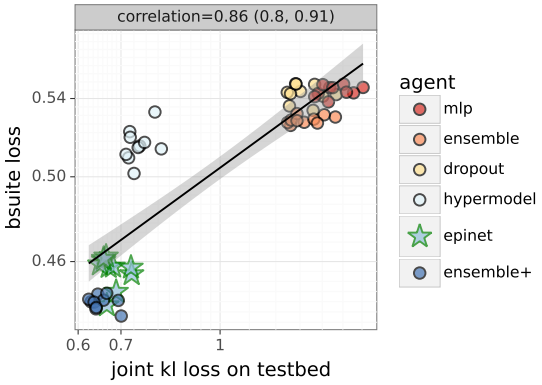
Figure 5: Evaluating performance and computational costs on `bsuite` reinforcement learning benchmark.

The scatter plots of Figure 6 report the correlation between prediction quality on the Neural Testbed and bsuite performance. The multiple points for any given agent represent results generated with different random seeds. The plot titles provide the estimated correlation, together with bootstrapped confidence intervals at the 5th and 95th percentiles, just as in Figure 6. Once again, our results mirror those of the Neural Testbed. Agents that produced accurate joint predictions performed well in the bsuite. However, the quality of marginal predictions showed no strong relation with performance on bsuite.

These results are significant for several reasons. First, we show that the high level observation that joint prediction quality relates to decision performance extends beyond synthetic neural network generative models. Further, these results occur even when we move beyond the simple classification setting of one-step rewards, towards a multi-step TD learning algorithm. Taken together, these provide a broader form of robustness around the efficacy of learning with epinet, and the importance of predictions beyond marginals.



(a) Marginal quality is not correlated with performance.



(b) Joint quality is highly correlated with performance.

Figure 6: Relating bsuite performance to prediction quality in the Neural Testbed.

7 CONCLUSION

This paper investigates the use of different epistemic neural networks to drive approximate Thompson sampling in decision problems. We find that, on average, ENNs that perform better in joint prediction on the Neural Testbed also tend to perform better in decision problems. These results are particularly significant in that they appear to be somewhat robust to the structure of the environment's generative model, with predictive power even when the tasks are very different from a 2-layer ReLU MLP.

Importantly, our experiments show that novel ENN architectures such as the epinet are able to match or even outperform existing approaches at orders of magnitude lower computational cost. This is the first paper to extend those results from the somewhat synthetic task of joint prediction, to actual decision making. We believe that this work, together with the open source code, can help set a base for future research into effective ENN architectures for better decision making in large deep learning systems.

Acknowledgements

We thank John Maggs for organization and management of this research effort and Rich Sutton, Yee Whye Teh, Geoffrey Irving, Koray Kavukcuoglu, Vlad Firoiu, Botao Hao, Grace Lam, Mehdi Jafarnia and Satinder Singh for helpful discussions and feedback.

References

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.

Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. *Advances in neural information processing systems*, 24, 2011.

Zhongxiang Dai, Yao Shu, Bryan Kian Hsiang Low, and Patrick Jaillet. Sample-then-optimize batch neural thompson sampling. *arXiv preprint arXiv:2210.06850*, 2022.

Vikranth Dwaracherla, Xiuyuan Lu, Morteza Ibrahimi, Ian Osband, Zheng Wen, and Benjamin Van Roy. Hypermodels for exploration. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=ryx6WgStPB`.

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

Aditya Gopalan and Shie Mannor. Thompson sampling for learning parameterized Markov decision processes. In *Proceedings of the 28th Annual Conference on Learning Theory*, 2015.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. URL `https://arxiv.org/abs/2006.00979`.

Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Wilson. What are Bayesian neural network posteriors really like? *arXiv preprint arXiv:2104.14421*, 2021.

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

Tze Leung Lai, Herbert Robbins, et al. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

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.

Xiuyuan Lu, Benjamin Van Roy, Vikranth Dwaracherla, Morteza Ibrahimi, Ian Osband, and Zheng Wen. Reinforcement learning, bit by bit. *arXiv preprint arXiv:2103.04047*, 2021.

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.

Ian Osband and Benjamin Van Roy. Model-based reinforcement learning and the eluder dimension. In *Advances in Neural Information Processing Systems 27*, pages 1466–1474, 2014a.

Ian Osband and Benjamin Van Roy. Near-optimal reinforcement learning in factored MDPs. In *Advances in Neural Information Processing Systems 27*, pages 604–612, 2014b.

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

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

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.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8617–8629. Curran Associates, Inc., 2018. URL `https://bit.ly/rpf_neurips`.

Ian Osband, Benjamin Van Roy, Daniel J Russo, and Zheng Wen. Deep exploration via randomized value functions. *Journal of Machine Learning Research*, 20(124):1–62, 2019.

Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvári, Satinder Singh, Benjamin Van Roy, Richard Sutton, David Silver, and Hado van Hasselt. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rygf-kSYwH`.

Ian Osband, Zheng Wen, Mohammad Asghari, Morteza Ibrahimi, Xiyuan Lu, and Benjamin Van Roy. Epistemic neural networks. *arXiv preprint arXiv:2107.08924*, 2021.

Ian Osband, Zheng Wen, Seyed Mohammad Asghari, Vikranth Dwaracherla, Botao Hao, Morteza Ibrahimi, Dieterich Lawson, Xiuyuan Lu, Brendan O'Donoghue, and Benjamin Van Roy. The neural testbed: Evaluating joint predictions. In *Advances in Neural Information Processing Systems*, volume 35. Curran Associates, Inc., 2022.

Daniel Russo and Benjamin Van Roy. Eluder dimension and the sample complexity of optimistic exploration. In *Advances in Neural Information Processing Systems 26*, pages 2256–2264. 2013.

Daniel Russo and Benjamin Van Roy. Learning to optimize via information-directed sampling. In *Advances in Neural Information Processing Systems 27*, pages 1583–1591. 2014.

Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on Thompson sampling. *Found. Trends Mach. Learn.*, 11(1): 1–96, July 2018. ISSN 1935-8237.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

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.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

Zheng Wen, Ian Osband, Chao Qin, Xiuyuan Lu, Morteza Ibrahimi, Vikranth Dwaracherla, Mohammad Asghari, and Benjamin Van Roy. From predictions to decisions: The importance of joint predictive distributions, 2022.

Weitong Zhang, Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural thompson sampling. *arXiv preprint arXiv:2010.00827*, 2020.