DEEP BAYESIAN BANDITS SHOWDOWN

AN EMPIRICAL COMPARISON OF BAYESIAN DEEP NETWORKS FOR THOMPSON SAMPLING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recently, deep reinforcement learning has made significant strides in performance on applications such as Go and Atari games. However, practical methods for balancing exploration and exploitation in complex domains remain largely unsolved. Thompson Sampling and its extension to reinforcement learning provide an elegant approach that requires access to posterior samples of deep models. At the same time, advances in approximate Bayesian methods have made posterior approximation for neural networks practical. As a first step towards understanding the impact of approximate posteriors on Thompson Sampling, we compare a variety of well-established and recent methods for approximate posterior sampling combined with Thompson Sampling over a series of contextual bandit problems.

1 INTRODUCTION

Recent advances in reinforcement learning have sparked renewed interest in sequential decision making with deep neural networks. Neural networks have proven to be powerful and flexible function approximators, allowing one to learn mappings directly from complex states (e.g., pixels) to estimates of expected return. While such models can be accurate on data they have been trained on, quantifying model uncertainty on new data remains challenging. However, having an understanding of what is not yet known or well understood is critical to some central tasks of machine intelligence, such as effective exploration for decision making.

A fundamental aspect of sequential decision making is the exploration-exploitation dilemma: in order to maximize cumulative reward, agents need to trade-off what is expected to be best at the moment, (i.e., exploitation), with potentially sub-optimal exploratory actions. Solving this trade-off in an efficient manner to maximize cumulative reward is a significant challenge as it requires uncertainty estimates. Furthermore, exploratory actions should be coordinated throughout the entire decision making process, known as deep exploration, rather than performed independently at each state.

Thompson Sampling (Thompson, 1933) and its extension to reinforcement learning, known as Posterior Sampling, provide an elegant approach that tackles the exploration-exploitation dilemma by maintaining a posterior over models and choosing actions in proportion to the probability that they are optimal. Unfortunately, maintaining such a posterior is intractable for all but the simplest models. As such, significant effort has been dedicated to approximate Bayesian methods for deep neural networks. These range from variational methods (Graves, 2011; Blundell et al., 2015; Kingma et al., 2015) to stochastic minibatch Markov Chain Monte Carlo (Neal, 1994; Welling & Teh, 2011; Li et al., 2016; Ahn et al., 2012; Mandt et al., 2016), among others. Because the exact posterior is intractable, evaluating these approaches is hard. Furthermore, these methods are rarely compared on benchmarks that measure the quality of their estimates of uncertainty for downstream tasks.

To address this challenge, we develop a benchmark for exploration methods using deep neural networks. We compare a variety of well-established and recent Bayesian approximations under the lens of Thompson Sampling for contextual bandits, a classical task in sequential decision making. All code and implementations to reproduce the experiments will be available open-source, to provide a reproducible benchmark [1] for future development.

Exploration in the context of reinforcement learning is a highly active area of research. Simple strategies such as epsilon-greedy remain extremely competitive (Mnih et al., 2015; Schaul et al.,

[1] Available in Python and Tensorflow at www.github.com/<anonymized>.

2016). However, a number of promising techniques have recently emerged that encourage exploration though carefully adding random noise to the parameters (Plappert et al., 2017; Fortunato et al., 2017; Gal & Ghahramani, 2016) or bootstrap sampling (Osband et al., 2016) before making decisions. These methods rely explicitly or implicitly on posterior sampling for exploration.

In this paper, we investigate how different posterior approximations affect the performance of Thompson Sampling from an empirical standpoint. For simplicity, we restrict ourselves to one of the most basic sequential decision making scenarios: that of contextual bandits.

No single algorithm bested the others in every bandit problem, however, we observed some general trends. We found that although bootstrapping, dropout, and injecting random noise helped for some problems, these strategies did not significantly and systematically improve performance in general. This suggests that the intrinsic randomness of stochastic gradient descent is enough to explore in many cases. Other algorithms, like Variational Inference and Monte Carlo approaches, strongly couple their complex representation and uncertainty estimates. This proves problematic when decisions are made based on partial optimization of both, as online scenarios usually require. On the other hand, making decisions according to a Bayesian linear regression on the representation provided by the last layer of a deep network offers a robust and easy-to-tune approach. It would be interesting to try this approach on more complex reinforcement learning domains.

In Section 2 we discuss Thompson Sampling, and present the contextual bandit problem. The different algorithmic approaches that approximate the posterior distribution fed to Thompson Sampling are introduced in Section 3, while the linear case is described in Section 4. The main experimental results are presented in Section 5, and discussed in Section 6. Finally, Section 7 concludes.

2 Decision-Making via Thompson Sampling

The contextual bandit problem works as follows. At time $t = 1, \ldots, n$ a new context $X_t \in \mathbf{R}^d$ arrives and is presented to algorithm $\mathcal{A}$. The algorithm —based on its internal model and X_t— selects one of the k available actions, a_t. Some reward $r_t = r_t(X_t, a_t)$ is then generated and returned to the algorithm, that may update its internal model with the new data. At the end of the process, the reward for the algorithm is given by $r = \sum_{t=1}^{n} r_t$, and cumulative regret is defined as $R_{\mathcal{A}} = \mathbf{E}[r^* - r]$, where r^* is the cumulative reward of the optimal policy (i.e., the policy that always selects the action with highest expected reward given the context). The goal is to minimize $R_{\mathcal{A}}$.

The main research question we address in this paper is how approximated model posteriors affect the performance of decision making via Thompson Sampling (Algorithm 1) in contextual bandits. We study a variety of algorithmic approaches to approximate a posterior distribution, together with different empirical and synthetic data problems that highlight several aspects of decision making. We consider distributions π over the space of parameters that completely define a problem instance $\theta \in \Theta$. For example, θ could encode the reward distributions of a set of arms in the multi-armed bandit scenario, or –more generally– all the parameters of an MDP in reinforcement learning.

Thompson Sampling is a classic algorithm (Thompson, 1933) which requires only that one can sample from the posterior distribution over plausible problem instances (for example, values or rewards). At each round, it draws a sample and takes a greedy action under the optimal policy for the sample. The posterior distribution is then updated after the result of the action is observed. Thompson Sampling has been shown to be extremely effective for bandit problems both in practice (Chapelle & Li, 2011; Granmo, 2010) and theory (Agrawal & Goyal, 2012). It is especially appealing for deep neural networks as one rarely has access to the full posterior but can often approximately sample from it.

In the following sections we rely on the idea that, if we had access to the actual posterior π_t given the observed data at all times t, then choosing actions using Thompson Sampling would lead to near-optimal cumulative regret or, more informally, to good performance. It is important to remark that in some problems this is not necessarily the case; for example, when actions that have no chance of being optimal still convey useful information about other actions. Thompson Sampling (or UCB approaches) would never select such actions, even if they are worth their cost (Russo & Van Roy, 2014). In addition, Thompson Sampling does *not* take into account the time horizon where the process ends, and if known, exploration efforts should be tuned accordingly (Russo et al., 2017). Nonetheless, under the assumption that very accurate posterior approximations lead to efficient decisions, the question is: what happens when the approximations are not so accurate? In some

Algorithm 1 Thompson Sampling

1: *Input:* Prior distribution over models, $\pi_0 : \theta \in \Theta \to [0, 1]$.
2: **for** time $t = 0, \ldots, N$ **do**
3: Observe context $X_t \in \mathrm{R}^d$.
4: Sample model $\theta_t \sim \pi_t$.
5: Compute $a_t = \text{BestAction}(X_t, \theta_t)$.
6: Select action a_t and observe reward r_t.
7: Update posterior distribution π_{t+1} with (X_t, a_t, r_t).

cases, the mismatch in posteriors may not hurt in terms of decision making, and we will still end up with good decisions. Unfortunately, in other cases, this mismatch together with its induced feedback loop will degenerate in a significant loss of performance. We would like to understand the main aspects that determine which way it goes. This is an important practical question as, in large and complex systems, computational sacrifices and statistical assumptions are made to favor simplicity and tractability. But, what is their impact?

3 ALGORITHMS

In this section, we describe the different algorithmic design principles that we considered in our simulations of Section 5. These algorithms include linear methods, Neural Linear and Neural Greedy, variational inference, expectation-propagation, dropout, Monte Carlo methods, bootstrapping, direct noise injection, and Gaussian Processes. In Figure 5, we visualize the posteriors of the nonlinear algorithms on a synthetic one dimensional problem.

Linear Methods We apply well-known closed-form updates for Bayesian linear regression (Bishop, 2006) for exact posterior inference in linear models. We provide the specific formulas below, and note that they admit a computationally-efficient online version. We consider exact linear posteriors as a baseline; i.e., these formulas compute the posterior when the data was generated according to $Y = X^T\beta + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$, and Y represents the reward. Importantly, we model the *joint* distribution of β and σ^2 for each action. Sequentially estimating the noise level σ^2 for each action allows the algorithm to adaptively improve its understanding of the volume of the hyperellipsoid of plausible β's; in general, this leads to a more aggressive initial exploration phase (in both β and σ^2).

The posterior at time t for action i, after observing X, Y, is $\pi_t(\beta, \sigma^2) = \pi_t(\beta \mid \sigma^2)\, \pi_t(\sigma^2)$, where we assume $\sigma^2 \sim \text{IG}(a_t, b_t)$, and $\beta \mid \sigma^2 \sim \mathcal{N}(\mu_t, \sigma^2\, \Sigma_t)$, an Inverse Gamma and Gaussian distribution, respectively. Their parameters are given by

$$\Sigma_t = \left(X^T X + \Lambda_0\right)^{-1}, \qquad \mu_t = \Sigma_t \left(\Lambda_0 \mu_0 + X^T Y\right), \tag{1}$$

$$a_t = a_0 + t/2, \qquad b_t = b_0 + \frac{1}{2}\left(Y^T Y + \mu_0^T \Sigma_0 \mu_0 - \mu_t^T \Sigma_t^{-1} \mu_t\right). \tag{2}$$

We set the prior hyperparameters to $\mu_0 = 0$, and $\Lambda_0 = \lambda\, \mathrm{Id}$, while $a_0 = b_0 = \eta > 1$. It follows that initially, for $\sigma_0^2 \sim IG(\eta, \eta)$, we have the prior $\beta \mid \sigma_0^2 \sim \mathcal{N}(0, \sigma_0^2/\lambda\, \mathrm{Id})$, where $\mathbf{E}[\sigma_0^2] = \eta/(\eta - 1)$. Note that we independently model and regress each action's parameters, β_i, σ_i^2 for $i = 1, \ldots, k$.

We consider two approximations to (1) motivated by function approximators where d is large. While posterior distributions or confidence ellipsoids should capture dependencies across parameters as shown above (say, a dense Σ_t), in practice, computing the correlations across all pairs of parameters is too expensive, and diagonal covariance approximations are common. For linear models it may still be feasible to exactly compute (1), whereas in the case of Bayesian neural networks, unfortunately, this may no longer be possible. Accordingly, we study two linear approximations where Σ_t is diagonal. Our goal is to understand the impact of such approximations in the simplest case, to properly set our expectations for the loss in performance of equivalent approximations in more complex approaches, like mean-field variational inference or Stochastic Gradient Langevin Dynamics.

Assume for simplicity the noise standard deviation is known. In Figure 2a, for $d = 2$, we see the posterior distribution $\beta_t \sim \mathcal{N}(\mu_t, \Sigma_t)$ of a linear model based on (1), in red, together with two diagonal approximations. Each approximation tries to minimize a different objective. In blue, the *PrecisionDiag* posterior approximation finds the diagonal $\hat{\Sigma} \in \mathbf{R}^{d \times d}$ minimizing $\mathrm{KL}(\mathcal{N}(\mu_t, \hat{\Sigma}) \mid\mid$

$\mathcal{N}(\mu_t, \Sigma_t)$), like in mean-field variational inference. In particular, $\hat{\Sigma} = \text{Diag}(\Sigma_t^{-1})^{-1}$. On the other hand, in green, the *Diag* posterior approximation finds the diagonal matrix $\bar{\Sigma}$ minimizing $\text{KL}(\mathcal{N}(\mu_t, \Sigma_t) \,||\, \mathcal{N}(\mu_t, \bar{\Sigma}))$ instead. In this case, the solution is simply $\bar{\Sigma} = \text{Diag}(\Sigma_t)$.

Neural Linear The main problem linear algorithms face is their lack of representational power, which they complement with accurate uncertainty estimates. A natural attempt at getting the best of both worlds consists in performing a Bayesian linear regression on top of the representation of the last layer of a neural network, similarly to Snoek et al. (2015). The predicted value v_i for each action a_i is given by $v_i = \beta_i^T z_x$, where z_x is the output of the last hidden layer of the network for context x. While linear methods directly try to regress values v on x, we can independently train a deep net to learn a representation z, and then use a Bayesian linear regression to regress v on z, obtain uncertainty estimates on the β's, and make decisions accordingly via Thompson Sampling. Note that we do not explicitly consider the weights of the linear output layer of the network to make decisions; further, the network is *only* used to find good representations z. In addition, we can update the network and the linear regression at different time-scales. It makes sense to keep an exact linear regression (as in (1) and (2)) at all times, adding each new data point as soon as it arrives. However, we only update the network after a number of points have been collected. In our experiments, after updating the network, we perform a forward pass on all the training data to obtain z_x, which is then fed to the Bayesian regression. In practice this may be too expensive, and z could be updated periodically with online updates on the regression. We call this algorithm *Neural Linear*.

Neural Greedy We refer to the algorithm that simply trains a neural network and acts greedily (i.e., takes the action whose predicted score for the current context is highest) as **RMS**, as we train it using the RMSProp optimizer. This is our non-linear baseline, and we tested several versions of it (based on whether the training step was decayed, reset to its initial value for each re-training or not, and how long the network was trained for). We also tried the ϵ-greedy version of the algorithm, where a random action was selected with probability ϵ for some decaying schedule of ϵ.

Variational Inference Variational approaches approximate the posterior by finding a distribution within a tractable family that minimizes the KL divergence to the posterior (Hinton & Van Camp, 1993). These approaches formulate and solve an optimization problem, as opposed, for example, to sampling methods like MCMC (Jordan et al., 1999; Wainwright et al., 2008). Typically (and in our experiments), the posterior is approximated by a mean-field or factorized distribution where strong independence assumptions are made. For instance, each neural network weight can be modeled via a –conditionally independent– Gaussian distribution whose mean and variance are estimated from data. Recent advances have scaled these approaches to estimate the posterior of neural networks with millions of parameters (Blundell et al., 2015). A common criticism of variational inference is that it underestimates uncertainty (e.g., (Bishop, 2006)), which could lead to under-exploration.

Expectation-Propagation The family of expectation-propagation algorithms (Opper & Winther, 2000; Minka, 2001b;a) is based on the message passing framework (Pearl, 1986). They iteratively approximate the posterior by updating a single approximation factor (or *site*) at a time, which usually corresponds to the likelihood of one data point. The algorithm sequentially minimizes a set of local KL divergences, one for each site. Most often, and for computational reasons, likelihoods are chosen to lie in the exponential family. In this case, the minimization corresponds to moment matching. See Gelman et al. (2014) for further details. We focus on methods that directly optimize the *global* EP objective via stochastic gradient descent, as, for instance, Power EP (Minka, 2004). In particular, in this work, we implement the black-box α-divergence minimization algorithm (Hernández-Lobato et al., 2016), where local parameter sharing is applied to the Power EP energy function. Note that different values of $\alpha \in \mathbf{R}\backslash\{0\}$ correspond to common algorithms: $\alpha = 1$ to EP, and $\alpha \to 0$ to Variational Bayes. The optimal α value is problem-dependent (Hernández-Lobato et al., 2016).

Dropout Dropout is a training technique where the output of each neuron is independently zeroed out with probability p at each forward pass (Srivastava et al., 2014). Once the network has been trained, dropout can still be used to obtain a distribution of predictions for a specific input. Following the best action with respect to the *random* dropout prediction can be interpreted as an implicit form of Thompson sampling. Dropout can be seen as optimizing a variational objective (Kingma et al., 2015; Gal & Ghahramani, 2016).

Monte Carlo Monte Carlo sampling remains one of the simplest and reliable tools in the Bayesian toolbox. Rather than parameterizing the full posterior, Monte Carlo methods estimate the posterior

through drawing samples. This is naturally appealing for highly parameterized deep neural networks for which the posterior is intractable in general and even simple approximations such as multivariate Gaussian are too expensive (i.e. require computing and inverting a covariance matrix over all parameters). Among Monte Carlo methods, Hamiltonian Monte Carlo (Neal, 1994) (HMC) is often regarded as a gold standard algorithm for neural networks as it takes advantage of gradient information and momentum to more effectively draw samples. However, it remains unfeasible for larger datasets as it involves a Metropolis accept-reject step that requires computing the log likelihood over the whole data set. A variety of methods have been developed to approximate HMC using mini-batch stochastic gradients. These Stochastic Gradient Langevin Dynamics (SGLD) methods (Neal, 1994; Welling & Teh, 2011) add Gaussian noise to the model gradients during stochastic gradient updates in such a manner that each update results in an approximate sample from the posterior. Different strategies have been developed for augmenting the gradients and noise according to a preconditioning matrix. Li et al. (2016) show that a preconditioner based on the RMSprop algorithm performs well on deep neural networks. Patterson & Teh (2013) suggested using the Fisher information matrix as a preconditioner in SGLD. Unfortunately the approximations of SGLD hold only if the learning rate is asymptotically annealed to zero. Ahn et al. (2012) introduced Stochastic Gradient Fisher Scoring to elegantly remove this requirement by preconditioning according to the Fisher information (or a diagonal approximation thereof). Mandt et al. (2016) develop methods for approximately sampling from the posterior using a constant learning rate in stochastic gradient descent and develop a prescription for a stable version of SGFS. We evaluate the diagonal-SGFS and constant-SGD algorithms from Mandt et al. (2016) in this work. Specifically for constant-SGD we use a constant learning rate for stochastic gradient descent, where the learning rate, ϵ is given by $\epsilon = 2\frac{S}{N}\mathbf{B}\mathbf{B}^T$ where S is the batch size, N the number of data points and $\mathbf{B}\mathbf{B}^T$ is an online average of the diagonal empirical Fisher information matrix. For Stochastic Gradient Fisher Scoring we use the following stochastic gradient update for the model parameters θ at step t:

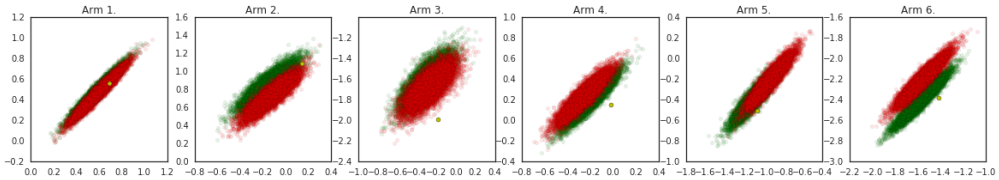
$$\theta(t+1) = \theta(t) - \epsilon\, \mathbf{H}\, g(\theta(t)) + \sqrt{\epsilon}\, \mathbf{H}\, \mathbf{E}\, \nu, \qquad \nu \sim \mathcal{N}(0, I) \tag{3}$$

where we take the noise covariance $\mathbf{E}\mathbf{E}^T$ to also be $\mathbf{B}\mathbf{B}^T$ and $\mathbf{H} = \frac{2}{N}(\epsilon\mathbf{B}\mathbf{B}^T + \mathbf{E}\mathbf{E}^{\mathbf{T}})^{-1}$.

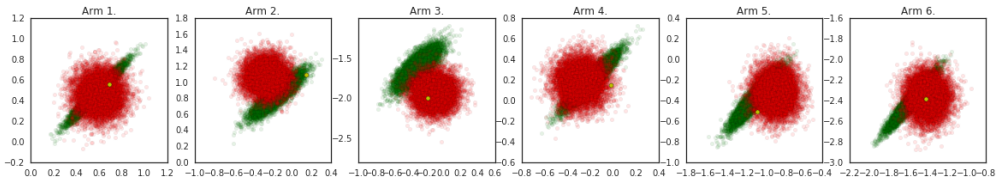
Bootstrap A simple empirical approach to approximate the sampling distribution of any estimator is the Bootstrap (Efron, 1982). The main idea is to simultaneously train q models, where each model i is based on a different dataset D_i. When all the data D is available in advance, D_i is typically created by sampling $|D|$ elements from D at random with replacement. In our case, however, the data grows one example at a time. Accordingly, we set a parameter $p \in (0, 1]$, and append the new datapoint to each D_i independently at random with probability p. In order to emulate Thompson Sampling, we sample a model uniformly at random (i.e., with probability $p_i = 1/q$.) and take the action predicted to be best by the sampled model. We tested versions where $q = 5, 10$ and $p = 0.8, 1.0$, with neural network models. Note that even when $p = 1$ and the datasets are identical, the random initialization of each network, together with the randomness from SGD, leads to different predictions.

Direct Noise Injection Parameter-Noise (Plappert et al., 2017) is a recently proposed approach for exploration in deep RL that has shown promising results. The training updates for the network are unchanged, but when selecting actions, the network weights are perturbed with isotropic Gaussian noise. Crucially, the network uses layer normalization (Ba et al., 2016), which ensures that all weights are on the same scale. The magnitude of the Gaussian noise is adjusted so that the overall effect of the perturbations is similar in scale to ϵ-greedy with a linearly decaying schedule (see (Plappert et al., 2017) for details). Because the perturbations are done on the model parameters, we might hope that the actions produced by the perturbations are more sensible than ϵ-greedy.

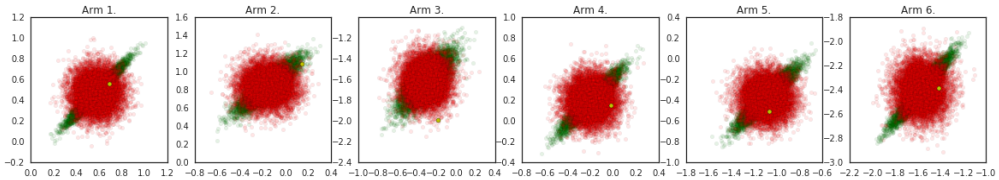
Bayesian Non-parametric Gaussian processes (Rasmussen & Williams, 2005) are a gold-standard method for modeling distributions over non-linear continuous functions. It can be shown that, in the limit of infinite hidden units and under a Gaussian prior, a Bayesian neural network converges to a Gaussian process (Neal, 1994). As such, GPs would appear to be a natural baseline. Unfortunately, standard GPs computationally scale cubically in the number of observations, limiting their applicability to relatively small datasets. There are a wide variety of methods to approximate Gaussian processes using, for example, pseudo-observations (Snelson & Ghahramani, 2006) or variational inference (Titsias, 2009). We implemented both a standard and sparse GP. For the standard GP, due to the scaling issue, we stop adding inputs to the GP after 1000 observations. This performed significantly better than randomly sampling inputs. Our implementation is a multi-task Gaussian process (Bonilla et al., 2008) with a linear and Matern $3/2$ product kernel over the inputs and an



(a) Two independent instances of Thompson Sampling with the true linear posterior.



(b) Thompson Sampling with the true linear posterior (green), and the diagonalized version (red).



(c) Linear posterior versus diagonal posterior fitted to the data collected by the former.

Figure 1: Visualizations of the posterior approximations in a linear example.

exponentiated quadratic kernel over latent vectors for the different tasks. The hyperparameters of this model and the latent task vectors are optimized over the GP marginal likelihood. This allows the model to learn correlations between the outputs of the model. Specifically, the covariance function $K(\cdot)$ of the GP is given by:

$$K(\mathbf{x}^k, \hat{\mathbf{x}}^l) = k_{\text{matern}}(\mathbf{x}^k, \hat{\mathbf{x}}^l) \cdot k_{\text{lin}}(\mathbf{x}^k, \hat{\mathbf{x}}^l) \cdot k_{\text{task}}(\mathbf{v}^k, \mathbf{v}^l) \tag{4}$$

$$k_{\text{matern}}(\mathbf{x}^k, \hat{\mathbf{x}}^l) = \alpha(1 + \sqrt{3} r_{\lambda_m}(\mathbf{x}, \hat{\mathbf{x}})) \exp(-\sqrt{3} r_{\lambda_m}(\mathbf{x}, \hat{\mathbf{x}}))) \tag{5}$$

$$k_{\text{lin}}(\mathbf{x}^k, \hat{\mathbf{x}}^l) = \beta(\mathbf{x} \oslash \lambda_l)(\hat{\mathbf{x}} \oslash \lambda_\mathbf{l})^T \tag{6}$$

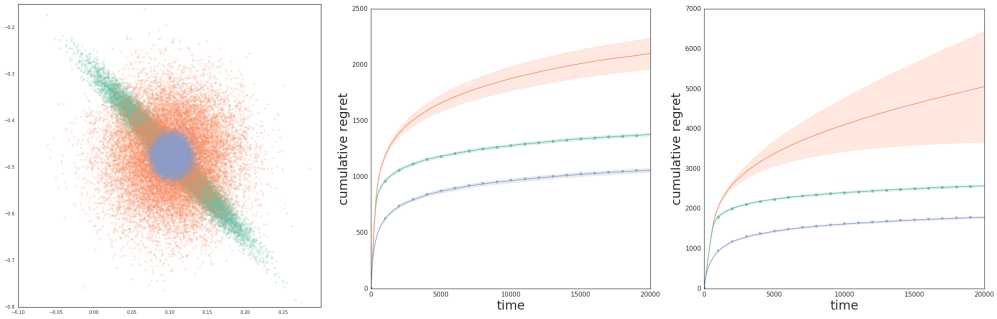
and the task kernel between tasks t and l are $k_t(\mathbf{x}^k, \hat{\mathbf{x}}^l) = \exp(-r_{\lambda_m}(\mathbf{v}^k, \hat{\mathbf{v}}^l))^2)$ where $\mathbf{v}^l$ indexes the latent vector for task l and $r_\lambda(\mathbf{x}, \hat{\mathbf{x}}) = |(\mathbf{x} \oslash \lambda) - (\hat{\mathbf{x}} \oslash \lambda)|$. The length-scales, λ_m and λ_l, and amplitude parameters α, β are optimized via the log marginal likelihood. For the sparse version we used a Sparse Variational GP Hensman et al. (2015) with the same kernel and with 300 inducing points, trained via minibatch stochastic gradient descent Matthews et al. (2017).

4 Feedback Loop in the Linear Case

In this section, we illustrate some of the subtleties that arise when uncertainty estimates drive sequential decision-making using simple linear examples.

There is a fundamental difference between *static* and *dynamic* scenarios. In a static scenario, e.g. supervised learning, we are given a model family Θ (like the set of linear models, trees, or neural networks with specific dimensions), a prior distribution π_0 over Θ, and some observed, and importantly assumed i.i.d., data $\mathbf{D}$. Our goal is to return an approximate posterior distribution: $\tilde{\pi} \approx \pi = \mathbf{P}(\theta \mid \mathbf{D})$. We define the quality of our approximation by means of some distance $d(\tilde{\pi}, \pi)$.

On the other hand, in *dynamic* settings, our estimate at time t, say $\tilde{\pi}_t$, will be used via some mechanism $\mathbf{M}$, in this case Thompson sampling, to collect the next data-point, which is then appended to $\mathbf{D}_t$. In this case, the data-points in $\mathbf{D}_t$ are no longer independent. $\mathbf{D}_t$ will now determine two distributions: the posterior given the data that was actually observed, $\pi_{t+1} = \mathbf{P}(\theta \mid \mathbf{D}_t)$, and our estimate $\tilde{\pi}_{t+1}$. When the goal is to make good sequential decisions in terms of cumulative regret, the distance $d(\tilde{\pi}_t, \pi_t)$ is in general no longer a definitive proxy for performance. For instance, a



(a) Posterior Approximations. (b) Case $d = 15$. (c) Case $d = 30$.

Figure 2: The impact on regret of different approximated posteriors. We show (green) the actual linear posterior, (orange) the diagonal posterior approximation and (blue) the precision approximation in 2a. In 2b and 2c we visualize the impact of the approximations on cumulative regret.

poorly-approximated decision boundary could lead an algorithm, based on $\tilde{\pi}$, to get stuck repeatedly selecting a single sub-optimal action a. After collecting lots of data for that action, $\tilde{\pi}_t$ and π_t could start to agree (to their capacity) on the models that explain what was observed for a, while both would stick to something close to the prior regarding the other actions. At that point, $d(\tilde{\pi}_t, \pi_t)$ may show relatively little disagreement, but the regret would already be terrible.

Let π_t^* be the posterior distribution $\mathbf{P}(\theta \mid \mathbf{D}_t)$ under the Thompson Sampling assumption, that is, data was always collected according to π_j^* for $j < t$. We follow the idea that $\tilde{\pi}_t$ being close to π_t^* for all t leads to strong performance. However, this concept is difficult to formalize: once different decisions are made, data for different actions is collected and it is hard to compare posterior distributions.

We illustrate the previous points with a simple example, see Figure 1. Data is generated according to a bandit with $k = 6$ arms. For a given context $X \sim \mathcal{N}(\mu, \Sigma)$, the reward obtained by pulling arm i follows a linear model $r_{i,X} = X^T \beta_i + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_i^2)$. The posterior distribution over $\beta_i \in \mathrm{R}^d$ can be exactly computed using the standard Bayesian linear regression formulas presented in Section 3. We set the contextual dimension $d = 20$, and the prior to be $\beta \sim \mathcal{N}(0, \lambda\, \mathrm{I}_d)$, for $\lambda > 0$.

In Figure 1, we show the posterior distribution for two dimensions of β_i for each arm i after $n = 500$ pulls. In particular, in Figure 1a, two independent runs of Thompson Sampling with their posterior distribution are displayed in red and green. While strongly aligned, the estimates for some arms disagree (especially for arms that are best only for a *small* fraction of the contexts, like Arm 2 and 3, where fewer data-points are available). In Figure 1b, we also consider Thompson Sampling with an approximate posterior with diagonal covariance matrix, *Diag* in red, as defined in Section 3. Each algorithm collects its own data based on its current posterior (or approximation). In this case, the posterior disagreement after $n = 500$ decisions is certainly stronger. However, as shown in Figure 1c, if we computed the approximate posterior with a diagonal covariance matrix based on the data collected *by the actual posterior*, the disagreement would be reduced as much as possible within the approximation capacity (i.e., it still cannot capture correlations in this case). Figure 1b shows then the effect of the feedback loop. We look next at the impact that this mismatch has on regret.

We illustrate with a similar example how inaccurate posteriors sometimes lead to quite different behaviors in terms of regret. In Figure 2a, we see the posterior distribution $\beta \sim \mathcal{N}(\mu, \Sigma)$ of a linear model in green, together with the two diagonal linear approximations introduced in Section 3: the Diag (in orange) and the PrecisionDiag (in blue) approximations, respectively. We now assume there are k linear arms, $\beta_i \in \mathbf{R}^d$ for $i = 1, \ldots, k$, and decisions are made according to the posteriors in Figure 2a. In Figures 2b and 2c we plot the regret of Thompson Sampling when there are $k = 20$ arms, for both $d = 15$ and $d = 30$. We see that, while the PrecisionDiag approximation does even outperform the actual posterior, the diagonal covariance approximation truly suffers poor regret when we increase the dimension d, as it is heavily penalized by *simultaneously* over-exploring in a large number of dimensions and repeateadly acting according to implausible models.

5 EMPIRICAL EVALUATION

In this section, we present the simulations and outcomes of several synthetic and real-world data bandit problems with each of the algorithms introduced in Section 3. In particular, we first explain how the simulations were set up and run, and the metrics we report. We then split the experiments according to how data was generated, and the underlying models fit by the algorithms from Section 3.

Table 1: Detailed description of the algorithms in the experiments. Unless otherwise stated, algorithms use $t_s = 20$ (mini-batches per training period), and $t_f = 20$ (one training period every t_f contexts).

Algorithm	Description
AlphaDiv	BB α-divergence with $\alpha = 0.1$, noise $\sigma = 0.1$, $K = 10$, prior var $\sigma_0^2 = 0.1$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
AlphaDiv1	BB α-divergence with $\alpha = 0.5$, noise $\sigma = 0.1$, $K = 10$, prior var $\sigma_0^2 = 0.1$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
AlphaDiv2	BB α-divergence with $\alpha = 1.0$, noise $\sigma = 0.1$, $K = 10$, prior var $\sigma_0^2 = 0.1$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
AlphaDiv3	BB α-divergence with $\alpha = 0.5$, noise $\sigma = 0.1$, $K = 10$, prior var $\sigma_0^2 = 1.0$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
BBB	BayesByBackprop with noise $\sigma = 0.1$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
BBB2	BayesByBackprop with noise $\sigma = 0.5$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
BBB3	BayesByBackprop with noise $\sigma = 0.75$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
BBB4	BayesByBackprop with noise $\sigma = 1.0$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
BootstrappedBNNs1	Bootstrapped with $q = 5$ models, and $p = 0.85$. Based on RMS3 net.
BootstrappedBNNs2	Bootstrapped with $q = 5$ models, and $p = 1.0$. Based on RMS3 net.
BootstrappedBNNs3	Bootstrapped with $q = 10$ models, and $p = 1.0$. Based on RMS3 net.
DropoutRMS	Dropout with probability $p = 0.8$. Based on RMS3 net.
DropoutRMS2	Dropout with probability $p = 0.8$. Based on RMS2 net.
GP-TS	For computational reasons, it only uses the first 1000 data points.
NeuralLinear	Noise prior $a_0 = 6$, $b_0 = 6$. Ridge prior $\lambda = 0.25$. Based on RMS2 net. Trained only for $t_s = 20$.
NeuralLinear2	Noise prior $a_0 = 3$, $b_0 = 3$. Ridge prior $\lambda = 0.25$. Based on RMS2 net. Trained only for $t_s = 20$.
NeuralLinear3	Noise prior $a_0 = 6$, $b_0 = 6$. Ridge prior $\lambda = 0.25$. Based on RMS2 net. Trained for $t_s = 100$.
NeuralLinear4	Noise prior $a_0 = 10$, $b_0 = 10$. Ridge prior $\lambda = 0.25$. Based on RMS2 net. Trained only for $t_s = 20$.
RMS	Greedy NN approach, fixed learning rate ($\gamma = 0.01$).
RMS2	Learning rate decays, and it is reset every training period.
RMS2b	Similar to RMS2, but training for longer ($t_s = 800$).
RMS3	Learning rate decays, and it is not reset at all. Starts at $\gamma = 1$.
SGFS	Burning $= 500$, learning rate $\gamma = 0.014$, EMA decay $= 0.9$, noise $\sigma = 0.75$.
SparseGP	Maximum number of points is 20000.
constSGD	Burning $= 500$, EMA decay $= 0.9$, noise $\sigma = 0.5$.
epsGreedyRMS1	Initial $\epsilon = 0.01$. Multiplied by 0.999 after every context. Based on RMS1 net.
epsGreedyRMS2	Initial $\epsilon = 0.01$. Multiplied by 0.999 after every context. Based on RMS2 net.
epsGreedyRMS3	Initial $\epsilon = 0.01$. Multiplied by 0.999 after every context. Based on RMS3 net.
linDiagPost	Σ in Eq. 1 is diagonalized. Ridge prior $\lambda = 0.25$. Assumed noise level $\sigma^2 = 0.25$.
linDiagPrecPost	Σ^{-1} in Eq. 1 is diagonalized. Ridge prior $\lambda = 0.25$. Assumed noise level $\sigma^2 = 0.25$.
linGreedy	Takes action with highest predicted reward for Ridge regression, $\lambda = 0.25$. Noise level $\sigma^2 = 0.25$.
linGreedyEps0-01	linGreedy that selects action uniformly at random with prob $p = 0.01$.
linGreedyEps0-05	linGreedy that selects action uniformly at random with prob $p = 0.05$.
linPost	Ridge prior $\lambda = 0.25$. Assumed noise level $\sigma^2 = 0.25$.
linfullDiagPost	Σ in Eq. 1 is diagonalized. Noise prior $a_0 = 6$, $b_0 = 6$. Ridge prior $\lambda = 0.25$.
linfullDiagPrecPost	Σ^{-1} in Eq. 1 is diagonalized. Noise prior $a_0 = 6$, $b_0 = 6$. Ridge prior $\lambda = 0.25$.
linfullPost	Noise prior $a_0 = 6$, $b_0 = 6$. Ridge prior $\lambda = 0.25$.
pNoise	Initial noise $\sigma = 0.01$, and level $\epsilon = 0.01$. Based on RMS3 net.
pNoise2	Initial noise $\sigma = 0.01$, and level $\epsilon = 0.01$. Based on RMS3 net. Trained for longer: $t_s = 800$.
unif	Takes each action at random with equal probability.

5.1 THE EXPERIMENTAL FRAMEWORK

We run the contextual bandit experiments as described at the beginning of Section 2, and discuss below some implementation details of both experiments and algorithms. A detailed summary of the key parameters used for each algorithm can be found in Table 1.

Neural Network Architectures All algorithms based on neural networks as function approximators share the same architecture. In particular, we fit a simple fully-connected feedforward network with two hidden layers with 100 units each and ReLu activations. The input of the network has dimension d (same as the contexts), and there are k outputs, one per action. Note that for each training point (X_t, a_t, r_t) only one action was observed (and algorithms usually only take into account the loss corresponding to the prediction for the observed action).

Updating Models A key question is how often and for how long models are updated. Ideally, we would like to train after each new observation and for as long as possible. However, this may limit the applicability of our algorithms in online scenarios where decisions must be made immediately. We update linear algorithms after each time-step by means of (1) and (2). For neural networks, the default behavior was to train for $t_s = 20$ or 100 mini-batches every $t_f = 20$ timesteps. [2] The size of

[2]For reference, the standard strategy for Deep Q-Networks on Atari is to make one model update after every 4 actions performed (Mnih et al., 2015; Osband et al., 2016; Plappert et al., 2017; Fortunato et al., 2017).

each mini-batch was 512. We experimented with increasing values of t_s, and it proved essential for some algorithms like variational inference approaches. See the details in Table 1.

Metrics We report two metrics: cumulative regret and simple regret. We approximate the latter as the mean cumulative regret in the last 500 time-steps, a proxy for the quality of the final policy (see further discussion on pure exploration settings, Bubeck et al. (2009)). Cumulative regret is computed based on the best expected reward, as is standard. For most real datasets (Statlog, Covertype, Jester, Adult, Census, and Song), the rewards were deterministic, in which case, the definition of regret also corresponds to the highest *realized* reward (i.e., possibly leading to a hard task, which helps to understand why in some cases all regrets look linear). We reshuffle the order of the contexts, and rerun the experiment 50 times to obtain the cumulative regret distribution and report its statistics.

Hyper-Parameter Tuning Deep learning methods are known to be very sensitive to the selection of a wide variety of hyperparameters and many of the algorithms presented are no exception. Moreover, that choice is known to be highly dataset dependent. However, in the bandits scenario, we would not have access to each problem a-priori to perform tuning. Thus, for each algorithm, the same hyperparameters are used across all tasks. We used Bayesian optimization to tune the hyperparameters to minimize cumulative regret over a subset of bandit problems (Dropout, SGFS, RMS, BBBN, pNoise) and some (Const-SGD, Neural Linear and all linear methods) required very little tuning [3].

Buffer After some experimentation, we decided not to use a data buffer as some evidence of catastrophic forgetting was observed. Accordingly, all observations are sampled with equal probability to be part of a mini-batch. In addition, as is standard in bandit algorithms, each action was initially selected $s = 3$ times using round-robin independently of the context.

5.2 Real-World Data Problems with Non-Linear Models

We evaluated the algorithms on a range of bandit problems created from real-world data. In particular, we test on the Mushroom, Statlog, Covertype, Financial, Jester, Adult, Census, and Song datasets (see Appendix Section A for details on each dataset and bandit problem). They exhibit a broad range of properties: small and large sizes, one dominating action versus more homogeneous optimality, learnable or little signal, stochastic or deterministic rewards, etc. For space reasons, the outcome of some simulations are presented in the Appendix. The Statlog, Covertype, Adult, and Census datasets were originally tested in Elmachtoub et al. (2017). We summarize the final cumulative regret for Mushroom, Statlog, Covertype, Financial, and Jester datasets in Table 2 (see Appendix Table 6 and 7 for the full results).

5.3 Real-World Data Problems with Linear Models

As the algorithms from Section 3 can be implemented for any model architecture, in this subsection we use linear models as a baseline comparison across algorithms. This allows us to directly compare the approximate methods against methods that can compute the exact posterior. Datasets are the same as in the previous subsection. Final cumulative regret and plots of the top methods are displayed in Appendix Table 4 and 5.

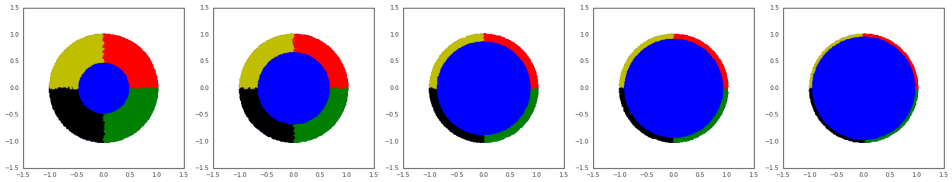
5.4 The Wheel Bandit

Some of the real-data problems presented above do not require significant exploration. We design an artificial problem where the need for exploration is smoothly parameterized. The *wheel* bandit is defined as follows (see Figure 3). Set $d = 2$, and $\delta \in (0, 1)$, the exploration parameter. Contexts are sampled uniformly at random in the unit circle in $\mathbf{R}^2$, $X \sim U(\mathbf{D})$. There are $k = 5$ possible actions. The first action a_1 always offers reward $r \sim \mathcal{N}(\mu_1, \sigma^2)$, independently of the context. On the other hand, for contexts such that $\|X\| \leq \delta$, i.e. inside the blue circle in Figure 3, the other four actions are equal and sub-optimal, with $r \sim \mathcal{N}(\mu_2, \sigma^2)$ for $\mu_2 < \mu_1$. When $\|X\| > \delta$, we are outside the blue circle, and only one of the actions $a_2, \ldots, a_5$ is optimal depending on the sign of context components $X = (X_1, X_2)$. If $X_1, X_2 > 0$, action 2 is optimal. If $X_1 > 0, X_2 < 0$, action 3 is optimal, and so on. Non-optimal actions still deliver $r \sim \mathcal{N}(\mu_2, \sigma^2)$ in this region, except a_1 whose mean reward is always μ_1, while the optimal action provides $r \sim \mathcal{N}(\mu_3, \sigma^2)$, with $\mu_3 \gg \mu_1$. We set $\mu_1 = 1.2, \mu_2 = 1.0, \mu_3 = 50.0$, and $\sigma = 0.01$. Note that the probability of a context randomly

[3] All algorithms are currently being tuned via Bayesian optimization and the paper will be updated once this is complete.

Table 2: Cumulative regret incurred by the algorithms in Section 3 on the bandits described in Section A. Results are relative to the cumulative regret of the Uniform algorithm. We report the mean and standard error of the mean over 50 trials.

	Mean Rank	Mushroom	Statlog	Covertype	Financial	Jester
Alpha Divergences	30.3333	54.12 ± 0.03	19.29 ± 0.84	44.32 ± 0.77	48.12 ± 0.90	70.62 ± 0.71
Alpha Divergences (1)	29.6667	54.24 ± 0.04	19.35 ± 1.72	39.43 ± 0.50	40.40 ± 0.69	71.73 ± 0.53
Alpha Divergences (2)	34.1667	54.38 ± 0.03	19.44 ± 1.91	68.48 ± 0.78	42.16 ± 0.48	76.00 ± 0.52
Alpha Divergences (3)	32.6667	55.10 ± 0.10	43.03 ± 1.33	60.20 ± 0.03	89.44 ± 2.45	67.55 ± 0.85
BBBN	23.8333	3.56 ± 0.20	12.60 ± 1.53	58.20 ± 2.16	22.71 ± 1.28	70.20 ± 0.66
BBBN2	27.3333	5.80 ± 1.16	25.23 ± 0.00	62.98 ± 0.84	39.12 ± 2.83	66.70 ± 0.79
BBBN3	26.3333	4.29 ± 1.06	25.23 ± 0.00	62.88 ± 0.83	45.24 ± 2.53	65.56 ± 0.77
BBBN4	29.5	10.47 ± 2.71	25.23 ± 0.00	61.92 ± 0.68	58.13 ± 3.62	64.26 ± 0.81
Bootstrapped NN	13.8333	7.72 ± 1.13	$\mathbf{0.67 \pm 0.03}$	26.67 ± 0.11	9.43 ± 0.33	72.30 ± 0.63
Bootstrapped NN2	12.1667	5.93 ± 1.20	$\mathbf{0.64 \pm 0.02}$	25.89 ± 0.06	10.82 ± 0.55	70.83 ± 0.55
Bootstrapped NN3	12	5.59 ± 0.60	$\mathbf{0.65 \pm 0.02}$	26.44 ± 0.24	9.28 ± 0.44	72.12 ± 0.61
Dropout (RMS3)	17.8333	5.37 ± 1.04	3.16 ± 0.84	31.04 ± 0.51	18.69 ± 0.72	65.38 ± 0.74
Dropout (RMS2)	15.8333	5.56 ± 1.02	2.35 ± 0.58	30.65 ± 0.23	17.68 ± 0.68	65.10 ± 0.73
GP	23.6	12.06 ± 1.31	4.56 ± 1.48		3.20 ± 0.08	73.66 ± 0.92
Neural Linear	14.3333	$\mathbf{2.29 \pm 0.09}$	1.06 ± 0.01	30.07 ± 0.14	13.10 ± 0.11	74.45 ± 0.50
Neural Linear (2)	13.8333	$\mathbf{2.22 \pm 0.08}$	0.91 ± 0.01	29.91 ± 0.17	11.52 ± 0.11	74.13 ± 0.41
Neural Linear (3)	14.8333	$\mathbf{2.29 \pm 0.09}$	1.09 ± 0.01	30.94 ± 0.21	13.54 ± 0.10	73.25 ± 0.46
Neural Linear (4)	15.5	$\mathbf{2.24 \pm 0.10}$	1.18 ± 0.01	29.99 ± 0.19	14.96 ± 0.10	73.75 ± 0.52
RMS1	21.5	8.64 ± 0.42	1.18 ± 0.36	25.14 ± 0.16	19.55 ± 0.67	73.98 ± 0.76
RMS2	18.1667	6.78 ± 1.60	2.41 ± 0.64	27.61 ± 0.16	12.21 ± 0.79	69.90 ± 0.62
RMS2b	17.8333	13.16 ± 2.35	2.06 ± 0.57	28.55 ± 0.25	9.44 ± 0.52	67.95 ± 0.58
RMS3	17.5	6.67 ± 1.52	2.84 ± 0.73	27.75 ± 0.18	12.83 ± 0.73	68.73 ± 0.55
SGFS	20.6667	5.98 ± 1.02	3.82 ± 0.45	36.58 ± 0.80	29.22 ± 0.53	66.85 ± 0.62
Sparse GP	32				32.31 ± 0.56	
ConstSGD	24.5	26.35 ± 3.14	6.79 ± 1.42	$\mathbf{22.48 \pm 0.78}$	88.82 ± 2.82	68.89 ± 0.79
EpsGreedy (RMS1)	21.3333	9.41 ± 0.32	1.22 ± 0.06	25.22 ± 0.16	20.53 ± 0.60	74.54 ± 0.73
EpsGreedy (RMS2)	14.1667	6.73 ± 1.42	1.89 ± 0.27	27.45 ± 0.14	11.01 ± 0.44	68.33 ± 0.59
EpsGreedy (RMS3)	13.5	4.97 ± 1.04	2.12 ± 0.34	27.43 ± 0.13	12.45 ± 0.47	68.45 ± 0.68
LinDiagPost	25.8333	17.65 ± 0.18	51.28 ± 0.03	95.51 ± 0.02	9.67 ± 0.05	$\mathbf{57.60 \pm 0.48}$
LinDiagPrecPost	14.5	9.47 ± 1.59	7.53 ± 0.02	34.41 ± 0.02	4.61 ± 0.04	$\mathbf{57.57 \pm 0.59}$
LinGreedy	18.1667	14.26 ± 1.98	11.32 ± 0.63	35.30 ± 0.11	$\mathbf{2.20 \pm 0.14}$	$\mathbf{58.66 \pm 0.59}$
LinGreedy (eps=0.01)	12.1667	3.38 ± 0.18	10.41 ± 0.39	34.60 ± 0.08	2.96 ± 0.12	58.92 ± 0.57
LinGreedy (eps=0.05)	18.1667	5.88 ± 0.06	12.74 ± 0.16	37.01 ± 0.03	6.62 ± 0.11	60.56 ± 0.43
LinPost	14.6667	6.12 ± 0.67	7.64 ± 0.02	34.41 ± 0.02	7.31 ± 0.05	$\mathbf{58.12 \pm 0.49}$
LinfullDiagPost	25.6667	86.72 ± 0.13	28.29 ± 0.02	73.84 ± 0.03	7.02 ± 0.06	62.14 ± 0.60
LinfullDiagPrecPost	11.6667	3.47 ± 0.36	7.34 ± 0.03	34.05 ± 0.02	4.07 ± 0.05	59.58 ± 0.43
LinfullPost	11.3333	$\mathbf{2.36 \pm 0.25}$	7.34 ± 0.02	34.01 ± 0.02	5.70 ± 0.04	60.81 ± 0.43
Param-Noise	14.6667	2.77 ± 0.15	1.47 ± 0.17	26.81 ± 0.10	19.19 ± 0.78	67.73 ± 0.52
Param-Noise2	16.1667	3.09 ± 0.14	1.54 ± 0.19	27.89 ± 0.19	18.77 ± 0.55	69.47 ± 0.57
Uniform	39	100.00 ± 0.13	100.00 ± 0.02	100.00 ± 0.02	100.00 ± 1.34	100.00 ± 1.12



(a) $\delta = 0.5$ (b) $\delta = 0.7$ (c) $\delta = 0.9$ (d) $\delta = 0.95$ (e) $\delta = 0.99$

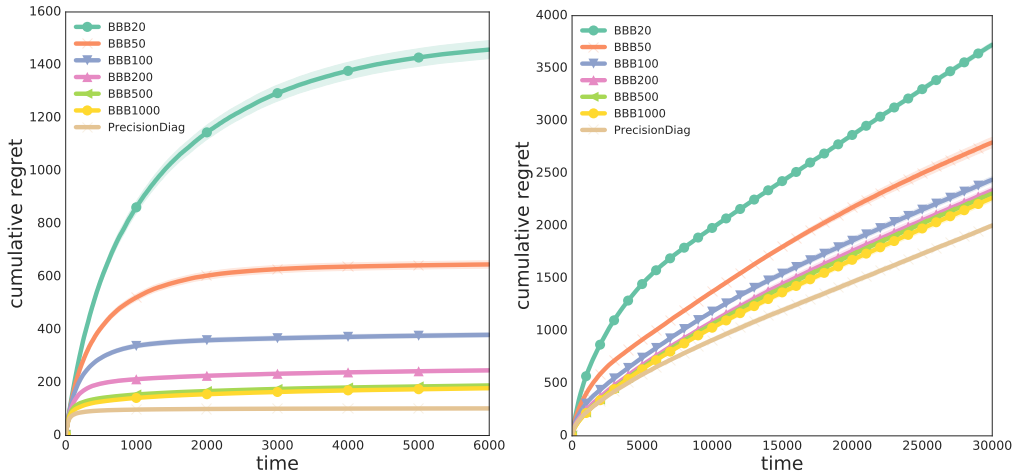
Figure 3: Wheel bandits for increasing values of $\delta \in (0, 1)$. Optimal action for blue, red, green, black, and yellow regions, are actions 1, 2, 3, 4, and 5, respectively.

falling in the high-reward region is $1 - \delta^2$ (not blue). The difficulty of the problem increases with δ, and we expect algorithms to get stuck repeatedly selecting action a_1 for large δ. The problem can be easily generalized for $d > 2$. Results are shown in Table 9.

6 Discussion

We discuss next our main findings for each class of algorithms.

Linear Methods. Linear methods offer a reasonable baseline, strong in many cases. While their representation power is certainly a limiting factor, their ability to compute informative uncertainty measures seems to payoff and balance their initial disadvantage. They do well in several datasets, and are able to react fast to surprising or extreme rewards (maybe as single points can have a heavy impact in fitted models, and their updates are deterministic and exact). Some datasets clearly need



(a) Linear data, $d = 10$ input size, and $k = 6$ actions.

(b) Statlog data.

Figure 4: Cumulative regret for BBB (noise $\sigma = 0.75$) applied to a linear model and an exact mean field solution, denoted PrecisionDiag, with a linear bandit (left) and with the Statlog bandit (right). The suffix of the BBB legend label indicates the number of training epochs in each training step. We emphasize that in this evaluation, all algorithms use the same family of models (i.e., linear). While PrecisionDiag exactly solves the mean field problem, BBB relies on partial optimization via SGD. As the number of training epochs increases, BBB improves performance, but is always outperformed by PrecisionDiag.

more complex non-linear representations, and linear methods are unable to efficiently solve those. In addition, linear methods obviously offer computational advantages.

NeuralLinear. The NeuralLinear algorithm sits near a sweet spot that is worth further studying. In general it seems to improve the RMS neural network it is based on, suggesting its exploration mechanisms add concrete value. We believe its main strength is that it is able to *simultaneously* learn a data representation that greatly simplifies the task at hand, and to accurately quantify the uncertainty over linear models that explain the observed rewards in terms of the proposed representation. While the former process may be noisier and heavily dependent on the amount of training steps that were taken and available data, the latter always offers the exact solution to its approximate parent problem. This, together with the partial success of linear methods with poor representations, may explain its promising results. In some sense, it knows what it knows. In the Wheel problem, which requires increasingly good exploration mechanisms, NeuralLinear is probably the best algorithm. Its performance is an order of magnitude better than any RMS algorithm (and its spinoffs, like Bootstrapped NN, Dropout, or Parameter Noise), and all greedy linear approaches. On the other hand, it is able to successfully solve problems that require non-linear representations (as Statlog or Covertype) where linear approaches fail. While conceptually simple, its deployment to large scale systems may involve some technical difficulties; mainly, to update the Bayesian estimates when the network is re-trained. We believe, however, standard solutions to similar problems (like running averages) could greatly mitigate these issues. In our experiments and compared to other algorithms, as shown in Table 8, NeuralLinear is fast from a computational standpoint.

Variational Inference. Overall, BBB performed poorly, ranking in the bottom half of algorithms across datasets (Table 2). To investigate if this was due to underestimating uncertainty (as variational methods are known to (Bishop, 2006)), to the mean field approximation, or to stochastic optimization, we applied BBB to a linear model, where the mean field optimization problem can be solved in closed form (Figure 4). We found that the performance of BBB slowly improved as the number of training epochs increased, but underperformed compared to the exact mean field solution. Moreover, the difference in performance due to the number of training steps dwarfed the difference between the mean field solution and the exact posterior. This suggests that it is not sufficient to partially optimize the variational parameters when the uncertainty estimates directly affect the data being

collected. In supervised learning, optimizing to convergence is acceptable, however in the online setting, optimizing to convergence at every step incurs unreasonable computational cost.

Monte Carlo. Constant-SGD comes out as the winner on Statlog and Covertype, two problems that both appear to require non-linearity and exploration as evidenced by performance of the linear baseline approaches. The method is especially appealing as it does not require tuning learning rates or exploration parameters. SGFS, however, performs significantly worse on most benchmarks except the hard exploration variant of the wheel problem where it shines. The additional injected noise in SGFS may cause the model to explore more than other approaches and thus incur additional regret.

Bootstrap. The bootstrap offers significant gains with respect to its parent algorithm (RMS3) in Statlog and Financial, while in the remaining datasets its performance is mostly similar. Note that in Statlog one of the actions is optimal around 80% of the time, and the bootstrapped predictions may help to avoid getting stuck, something from which RMS3 could suffer. The simple regret in the Mushroom problem strongly outperforms that of RMS3 too, suggesting better final policies even though the cumulative regret is comparable for the selected horizon. In other scenarios, the randomness from SGD may be enough for exploration, and the bootstrap may not offer important benefits. In those cases, it might not justify the heavy computational overhead of the method. Unfortunately, Bootstrapped NNs were not able to solve the Wheel problem, and only when $\delta = 0.5$ they led to significant improvements with respect to RMS3. In the harder cases ($\delta > 0.5$) the performance was similar. One possible explanation is that —given the sparsity of the reward— all the bootstrapped networks agreed for the most part, and the algorithm simply got stuck selecting action a_1. As opposed to linear models, reacting to unusual rewards could take Bootstrapped NNs some time as good predictions could be overlooked (and useful data even discarded if $p \ll 1$).

Direct Noise Injection. Parameter-Noise provides a small boost in performance over the base learner that it was based on (RMS3), average rank 14.67 and 16.17 compared to 17.5 (Table 2). However, we found the algorithm hard to tune and sensitive to the heuristic controlling the injected noise-level. On the synthetic Wheel problem where exploration is necessary, both parameter-noise and RMS3 suffer from underexploration and perform similarly. In addition, developing an intuition for the heuristic is not straightforward as it lacks transparency and a principled grounding, and thus may require repeated access to the decision-making process for tuning.

Dropout. We mainly experimented with two dropout versions: fixed $p = 0.5$, and $p = 0.8$. The latter consistently delivered better results, and it is the one we report. The algorithm required no extra-tuning, and sometimes offered small improvements (Mushroom and Jester). However, in the rest of cases performance became slightly worse by using dropout. In the Wheel problem, dropout mostly harmed its parent algorithm (RMS2 and RMS3). Only for the hardest problem, $\delta = 0.99$, performance is similar for both approaches as, most likely, dropout had no real impact. Dropout was used both for training and for decision-making; unfortunately, we did not add a baseline where dropout only applies during training. Consequently, it is not obvious how to disentangle the contribution of better training from that of better exploration. This remains as future work.

Bayesian Non-parametrics. Perhaps unsurprisingly, the Gaussian process performs remarkably well on problems with little data but struggles on larger problems. This may motivate the use of sparse GP-based approaches in the future.

7 Conclusions and Future Work

In this work, we empirically studied the impact on performance of approximate model posteriors for decision making via Thompson Sampling in contextual bandits. We found to perform more robustly those methods that exactly measure uncertainty (possibly under the wrong model assumptions) on top of complex representations learned in parallel. More complicated approaches that learn the representation and its uncertainty together seemed to require heavier training, an important drawback in online scenarios, and exhibited stronger hyper-parameter dependence. Further exploring and developing the promising approaches is an exciting avenue for future work.

ACKNOWLEDGMENTS

REFERENCES

Agrawal, Shipra and Goyal, Navin. Analysis of thompson sampling for the multi-armed bandit problem. In *International Conference on Learning Theory*, 2012.

Ahn, Sungjin, Balan, Anoop Korattikara, and Welling, Max. Bayesian posterior sampling via stochastic gradient fisher scoring. In *International Conference on Machine Learning*, 2012.

Asuncion, Arthur and Newman, David. UCI machine learning repository, 2007.

Ba, Jimmy Lei, Kiros, Jamie Ryan, and Hinton, Geoffrey E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bertin-Mahieux, Thierry, Ellis, Daniel P.W., Whitman, Brian, and Lamere, Paul. The million song dataset. In *International Conference on Music Information Retrieval*, 2011.

Bishop, Christopher M. *Pattern recognition and machine learning*. springer, 2006.

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

Bonilla, Edwin V, Chai, Kian M., and Williams, Christopher. Multi-task gaussian process prediction. In *Advances in Neural Information Processing Systems*, 2008.

Bubeck, Sébastien, Munos, Rémi, and Stoltz, Gilles. Pure exploration in multi-armed bandits problems. In *International conference on Algorithmic learning theory*, pp. 23–37. Springer, 2009.

Chapelle, Olivier and Li, Lihong. An empirical evaluation of Thompson sampling. In *Advances in Neural Information Processing Systems 24*, 2011.

Efron, Bradley. *The jackknife, the bootstrap and other resampling plans*. SIAM, 1982.

Elmachtoub, Adam N, McNellis, Ryan, Oh, Sechan, and Petrik, Marek. A practical method for solving contextual bandit problems using decision trees. *arXiv preprint arXiv:1706.04687*, 2017.

Fortunato, Meire, Azar, Mohammad Gheshlaghi, Piot, Bilal, Menick, Jacob, Osband, Ian, Graves, Alex, Mnih, Vlad, Munos, Remi, Hassabis, Demis, Pietquin, Olivier, Blundell, Charles, and Legg, Shane. Noisy networks for exploration. *arXiv:1706.10295*, 2017.

Gal, Yarin and Ghahramani, Zoubin. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International conference on machine learning*, pp. 1050–1059, 2016.

Gelman, Andrew, Vehtari, Aki, Jylänki, Pasi, Robert, Christian, Chopin, Nicolas, and Cunningham, John P. Expectation propagation as a way of life. *arXiv preprint arXiv:1412.4869*, 2014.

Goldberg, Ken, Roeder, Theresa, Gupta, Dhruv, and Perkins, Chris. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

Granmo, OleChristoffer. Solving twoarmed Bernoulli bandit problems using a Bayesian learning automaton. *International Journal of Intelligent Computing and Cybernetics*, 3(2):207–234, 2010.

Graves, Alex. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pp. 2348–2356, 2011.

Hensman, James, Matthews, Alexander G. de G., and Ghahramani, Zoubin. Scalable variational gaussian process classification. In *Proceedings of AISTATS*, 2015.

Hernández-Lobato, José Miguel, Li, Yingzhen, Rowland, Mark, Bui, Thang D., Hernández-Lobato, Daniel, and Turner, Richard E. Black-box alpha divergence minimization. In *International Conference on Machine Learning*, 2016.

Hinton, Geoffrey E and Van Camp, Drew. Keeping the neural networks simple by minimizing the description length of the weights. In *Computational learning theory*, pp. 5–13. ACM, 1993.

Jordan, Michael I, Ghahramani, Zoubin, Jaakkola, Tommi S, and Saul, Lawrence K. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Kingma, Diederik P, Salimans, Tim, and Welling, Max. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

Kohavi, Ron. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *International Conference On Knowledge Discovery and Data Mining*, 1996.

Li, Chunyuan, Chen, Changyou, Carlson, David, and Carin, Lawrence. Preconditioned stochastic gradient langevin dynamics for deep neural networks. In *AAAI Conference on Artificial Intelligence*, 2016.

Mandt, Stephan, Hoffman, Matthew D., and Blei, David M. A variational analysis of stochastic gradient algorithms. In *International Conference on Machine Learning*, 2016.

Matthews, Alexander G. de G., van der Wilk, Mark, Nickson, Tom, Fujii, Keisuke., Boukouvalas, Alexis, León-Villagrá, Pablo, Ghahramani, Zoubin, and Hensman, James. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 2017.

Minka, Thomas. Power ep. 2004.

Minka, Thomas P. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pp. 362–369. Morgan Kaufmann Publishers Inc., 2001a.

Minka, Thomas Peter. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001b.

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

Neal, Radford M. Bayesian learning for neural networks. *Dept. of Computer Science, University of Toronto*, 1994.

Opper, Manfred and Winther, Ole. Gaussian processes for classification: Mean-field algorithms. *Neural computation*, 12(11):2655–2684, 2000.

Osband, Ian, Blundell, Charles, Pritzel, Alexander, and Van Roy, Benjamin. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pp. 4026–4034, 2016.

Patterson, Sam and Teh, Yee Whye. Stochastic gradient riemannian langevin dynamics on the probability simplex. In *Advances in Neural Information Processing Systems*, 2013.

Pearl, Judea. Fusion, propagation, and structuring in belief networks. *Artificial intelligence*, 29(3): 241–288, 1986.

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

Rasmussen, Carl Edward and Williams, Christopher K. I. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

Riquelme, Carlos, Ghavamzadeh, Mohammad, and Lazaric, Alessandro. Active learning for accurate estimation of linear models. In *International Conference on Machine Learning*, 2017.

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

Russo, Daniel, Tse, David, and Van Roy, Benjamin. Time-sensitive bandit learning and satisficing thompson sampling. *arXiv preprint arXiv:1704.09028*, 2017.

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

Schlimmer, Jeff. Mushroom records drawn from the audubon society field guide to north american mushrooms. *GH Lincoff (Pres), New York*, 1981.

Snelson, Edward and Ghahramani, Zoubin. Sparse gaussian processes using pseudo-inputs. In Weiss, Y., Schölkopf, P. B., and Platt, J. C. (eds.), *Advances in Neural Information Processing Systems*, 2006.

Snoek, Jasper, Rippel, Oren, Swersky, Kevin, Kiros, Ryan, Satish, Nadathur, Sundaram, Narayanan, Patwary, Mostofa, Prabhat, Mr, and Adams, Ryan. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, 2015.

Srivastava, Nitish, Hinton, Geoffrey E, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

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

Titsias, Michalis K. Variational learning of inducing variables in sparse gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, 2009.

Wainwright, Martin J, Jordan, Michael I, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

Welling, Max and Teh, Yee Whye. Bayesian learning via stochastic gradient langevin dynamics. In *International Conference on Machine Learning*, 2011.

APPENDIX

Table 3: Detailed description of the algorithms in the experiments. Unless otherwise stated, algorithms use $t_s = 100$ (mini-batches per training period), and $t_f = 20$ (one training period every t_f contexts).

Algorithm	Description
AlphaDiv	BB α-divergence with $\alpha = 0.1$, noise $\sigma = 0.1$, $K = 10$, prior var $\sigma_0^2 = 0.1$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
AlphaDiv1	BB α-divergence with $\alpha = 0.5$, noise $\sigma = 0.1$, $K = 10$, prior var $\sigma_0^2 = 0.1$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
AlphaDiv2	BB α-divergence with $\alpha = 1.0$, noise $\sigma = 0.1$, $K = 10$, prior var $\sigma_0^2 = 0.1$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
AlphaDiv3	BB α-divergence with $\alpha = 0.5$, noise $\sigma = 0.1$, $K = 10$, prior var $\sigma_0^2 = 1.0$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
BBB	BayesByBackprop with noise $\sigma = 0.1$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
BBB2	BayesByBackprop with noise $\sigma = 0.5$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
BBB3	BayesByBackprop with noise $\sigma = 0.75$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
BBB4	BayesByBackprop with noise $\sigma = 1.0$. ($t_s = 100$, first 100 times linear decay from $t_s = 10000$).
BootstrappedBNNs1	Bootstrapped with $q = 5$ models, and $p = 0.85$. Based on RMS3 net.
BootstrappedBNNs2	Bootstrapped with $q = 5$ models, and $p = 1.0$. Based on RMS3 net.
BootstrappedBNNs3	Bootstrapped with $q = 10$ models, and $p = 1.0$. Based on RMS3 net.
DropoutRMS	Dropout with probability $p = 0.8$. Based on RMS3 net.
DropoutRMS2	Dropout with probability $p = 0.8$. Based on RMS2 net.
GP-TS	For computational reasons, it only uses the first 1000 data points.
NeuralLinear	Noise prior $a_0 = 6$, $b_0 = 6$. Ridge prior $\lambda = 0.25$. Based on RMS2 net. Trained only for $t_s = 20$.
NeuralLinear2	Noise prior $a_0 = 3$, $b_0 = 3$. Ridge prior $\lambda = 0.25$. Based on RMS2 net. Trained only for $t_s = 20$.
NeuralLinear3	Noise prior $a_0 = 6$, $b_0 = 6$. Ridge prior $\lambda = 0.25$. Based on RMS2 net. Trained for $t_s = 100$.
NeuralLinear4	Noise prior $a_0 = 10$, $b_0 = 10$. Ridge prior $\lambda = 0.25$. Based on RMS2 net. Trained only for $t_s = 20$.
RMS	Greedy NN approach, fixed learning rate ($\gamma = 0.01$).
RMS2	Learning rate decays, and it is reset every training period.
RMS2b	Similar to RMS2, but training for longer ($t_s = 800$).
RMS3	Learning rate decays, and it is not reset at all. Starts at $\gamma = 1$.
SGFS	Burning $= 500$, learning rate $\gamma = 0.014$, EMA decay $= 0.9$, noise $\sigma = 0.75$.
SparseGP	Maximum number of points is 20000.
constSGD	Burning $= 500$, EMA decay $= 0.9$, noise $\sigma = 0.5$.
epsGreedyRMS1	Initial $\epsilon = 0.01$. Multiplied by 0.999 after every context. Based on RMS1 net.
epsGreedyRMS2	Initial $\epsilon = 0.01$. Multiplied by 0.999 after every context. Based on RMS2 net.
epsGreedyRMS3	Initial $\epsilon = 0.01$. Multiplied by 0.999 after every context. Based on RMS3 net.
linDiagPost	Σ in Eq. 1 is diagonalized. Ridge prior $\lambda = 0.25$. Assumed noise level $\sigma^2 = 0.25$.
linDiagPrecPost	Σ^{-1} in Eq. 1 is diagonalized. Ridge prior $\lambda = 0.25$. Assumed noise level $\sigma^2 = 0.25$.
linGreedy	Takes action with highest predicted reward for Ridge regression, $\lambda = 0.25$. Noise level $\sigma^2 = 0.25$.
linGreedyEps0-01	linGreedy that selects action uniformly at random with prob $p = 0.01$.
linGreedyEps0-05	linGreedy that selects action uniformly at random with prob $p = 0.05$.
linPost	Ridge prior $\lambda = 0.25$. Assumed noise level $\sigma^2 = 0.25$.
linfullDiagPost	Σ in Eq. 1 is diagonalized. Noise prior $a_0 = 6$, $b_0 = 6$. Ridge prior $\lambda = 0.25$.
linfullDiagPrecPost	Σ^{-1} in Eq. 1 is diagonalized. Noise prior $a_0 = 6$, $b_0 = 6$. Ridge prior $\lambda = 0.25$.
linfullPost	Noise prior $a_0 = 6$, $b_0 = 6$. Ridge prior $\lambda = 0.25$.
pNoise	Initial noise $\sigma = 0.01$, and level $\epsilon = 0.01$. Based on RMS3 net.
pNoise2	Initial noise $\sigma = 0.01$, and level $\epsilon = 0.01$. Based on RMS3 net. Trained for longer: $t_s = 800$.
unif	Takes each action at random with equal probability.

Table 4: Cumulative regret incurred by *linear* models using algorithms in Section 3 on the bandits described in Section A. Values reported are the mean over 50 independent trials with standard error of the mean.

	Mushroom	Statlog	Covertype	Financial	Jester	Adult	Song	Census
Cumulative regret								
Alpha Divergences	29051.02 ± 238.56	3213.57 ± 8.88	47594.06 ± 30.86	119.92 ± 1.70	60401.66 ± 455.68	34774.94 ± 18.70	134944.04 ± 15.02	69479.09 ± 41.17
Alpha Divergences (1)	65203.06 ± 204.46	3636.06 ± 19.61	47904.47 ± 27.89	118.67 ± 1.86	61896.47 ± 380.77	35043.71 ± 15.02	134950.96 ± 19.57	71224.80 ± 51.79
Alpha Divergences (2)	67612.86 ± 195.67	5644.49 ± 24.17	50401.92 ± 35.04	131.40 ± 2.05	69341.27 ± 587.65	35925.67 ± 12.82	134936.35 ± 21.05	75907.28 ± 43.19
Alpha Divergences (3)	77061.53 ± 221.74	3670.65 ± 17.56	47973.43 ± 35.15	164.44 ± 2.89	62070.93 ± 452.55	35193.04 ± 16.59	134969.47 ± 18.76	71749.81 ± 50.83
BBBN	19352.55 ± 2315.29	**2611.59 ± 9.19**	**43684.69 ± 59.12**	132.12 ± 11.53	**56384.55 ± 477.64**	**31650.61 ± 15.92**	133776.71 ± 21.46	43048.44 ± 28.41
BBBN2	24885.00 ± 4784.46	2820.29 ± 7.58	44603.71 ± 21.78	232.86 ± 2.07	**56465.72 ± 498.79**	33540.20 ± 15.02	133856.10 ± 24.15	48105.98 ± 32.19
BBBN3	43362.04 ± 7452.98	3065.43 ± 11.70	45880.12 ± 21.76	361.13 ± 2.62	57055.81 ± 492.60	35011.00 ± 16.26	133879.53 ± 18.29	52947.71 ± 28.81
BBBN4	61581.02 ± 8644.66	3365.49 ± 15.22	47324.12 ± 21.01	496.80 ± 3.84	**55826.75 ± 525.71**	36290.27 ± 16.80	134006.96 ± 19.50	58207.12 ± 30.64
Bootstrapped NN	15207.55 ± 2223.28	2801.43 ± 58.40	48399.53 ± 276.16	149.15 ± 12.35	57922.32 ± 470.78	31740.18 ± 35.07	120264.57 ± 21.99	
Bootstrapped NN2	30388.98 ± 3002.15	2777.61 ± 85.16	49919.61 ± 421.32	124.47 ± 8.93	56886.28 ± 522.05	31760.43 ± 53.26		
Bootstrapped NN3	22681.53 ± 2445.45	**2574.35 ± 38.98**		**108.00 ± 6.96**	56693.99 ± 497.70	31845.12 ± 63.36		
Dropout (RMS3)	24079.80 ± 2905.35	3504.33 ± 233.49	49382.00 ± 336.47	**118.64 ± 6.63**	**56600.80 ± 562.76**	32133.98 ± 67.08	120167.76 ± 26.96	55489.62 ± 623.32
Dropout (RMS2)	30470.00 ± 3256.06	3594.18 ± 224.22	49555.14 ± 354.48	127.23 ± 8.85	57321.68 ± 496.09	32132.51 ± 81.59	120176.47 ± 34.39	54965.00 ± 550.83
RMS1	19626.63 ± 2706.22	3147.37 ± 12.28	49518.84 ± 234.92	126.73 ± 9.38	58098.59 ± 546.44	33514.08 ± 20.16	135004.90 ± 20.29	74938.92 ± 161.56
RMS2	24920.71 ± 3629.18	3511.47 ± 237.12	49475.98 ± 355.82	129.84 ± 9.76	**55619.60 ± 476.35**	32183.02 ± 71.02	120198.61 ± 37.30	55862.26 ± 502.10
RMS2b	35375.51 ± 3941.93	4128.22 ± 273.86	49554.74 ± 461.20	128.02 ± 10.80	**55941.69 ± 554.92**	32094.82 ± 78.15		
RMS3	24408.57 ± 2876.77	3288.41 ± 198.61	50056.27 ± 392.93	**109.53 ± 7.92**	56661.44 ± 495.01	32087.24 ± 65.81	120158.53 ± 30.15	55497.49 ± 642.54
SGFS	129586.43 ± 82.79	10811.00 ± 186.70	79162.47 ± 375.83	4747.22 ± 64.59	67961.22 ± 1393.66	40784.20 ± 87.79	131348.35 ± 196.77	92249.10 ± 310.83
ConstSGD	129737.86 ± 85.36	9405.92 ± 0.33	79492.84 ± 927.87	4749.99 ± 72.03	65572.88 ± 1112.01	39450.41 ± 33.53	130140.61 ± 334.40	95594.83 ± 2181.46
EpsGreedy (RMS1)	8732.76 ± 516.32	3283.57 ± 11.93	48605.16 ± 125.63	143.89 ± 4.31	56961.95 ± 444.26	33569.51 ± 19.84	134967.37 ± 18.90	75233.08 ± 140.98
EpsGreedy (RMS2)	23687.76 ± 2166.78	3009.10 ± 124.54	48617.06 ± 331.37	146.49 ± 5.56	**55855.45 ± 457.55**	32021.98 ± 66.44	120172.06 ± 22.07	53505.02 ± 494.23
EpsGreedy (RMS3)	23637.96 ± 1847.91	2938.84 ± 80.86	48509.29 ± 273.31	145.17 ± 5.62	**56072.51 ± 479.59**	32026.18 ± 67.21	120212.90 ± 29.71	54322.20 ± 462.86
LinDiagPost	42696.84 ± 276.34	19118.47 ± 12.30	122826.24 ± 24.59	459.02 ± 2.95	**56568.08 ± 527.28**	41858.82 ± 9.56	127053.92 ± 30.11	
LinDiagPrecPost	19523.67 ± 2404.15	2805.69 ± 9.76	44253.76 ± 50.98	217.08 ± 2.17	**56227.81 ± 463.11**	33521.02 ± 15.75	117499.96 ± 25.33	
LinGreedy	26835.20 ± 3759.77	3897.45 ± 215.39	45446.98 ± 166.07	**105.48 ± 6.73**	**55651.56 ± 438.66**	34558.63 ± 315.75	**114664.76 ± 27.56**	**40941.36 ± 49.68**
LinGreedy (eps=0.01)	8522.86 ± 537.80	3775.86 ± 143.74	44407.24 ± 81.43	129.20 ± 3.84	57082.85 ± 538.96	32214.82 ± 114.46	114830.94 ± 30.07	41812.63 ± 29.27
LinGreedy (eps=0.05)	14489.69 ± 145.06	4814.65 ± 66.76	47485.59 ± 42.17	308.34 ± 4.74	58201.11 ± 460.96	31813.22 ± 19.33	115743.71 ± 24.81	45414.69 ± 42.06
LinPost	16251.22 ± 1680.93	2849.12 ± 6.62	44176.49 ± 22.51	347.11 ± 2.69	56677.18 ± 550.34	33243.59 ± 16.12	117629.16 ± 24.31	
LinfullDiagPost	212029.08 ± 387.36	10546.86 ± 9.82	94875.53 ± 34.06	336.08 ± 2.70	59981.66 ± 388.85	40372.00 ± 12.03	123684.71 ± 20.84	
LinfullDiagPrecPost	10485.82 ± 1081.24	2738.04 ± 7.74	**43734.08 ± 26.14**	189.89 ± 2.13	57738.18 ± 428.36	32329.84 ± 12.74	116112.57 ± 26.44	
LinfullPost	**5299.90 ± 532.93**	2746.51 ± 6.46	**43728.67 ± 25.12**	267.23 ± 2.33	59037.26 ± 463.65	32233.51 ± 14.20	116281.65 ± 23.42	
Param-Noise	27804.49 ± 3389.40	2888.02 ± 61.41	49353.51 ± 364.93	142.05 ± 13.22	57399.91 ± 543.27	32091.20 ± 80.79	120321.43 ± 27.15	54919.50 ± 442.45
Param-Noise2	22882.35 ± 2682.16	2853.94 ± 57.96	49426.47 ± 326.31	**107.94 ± 5.45**	57418.49 ± 530.30	32090.24 ± 66.27		
Uniform	244892.76 ± 412.03	37288.98 ± 8.71	128543.61 ± 18.59	4672.92 ± 57.66	95646.52 ± 1145.95	41989.37 ± 8.38	135007.24 ± 20.51	133309.71 ± 18.18

Table 5: Simple regret incurred by *linear* models using algorithms in Section 3 on the bandits described in Section A. Simple regret was approximated by averaging the regret over the final 500 steps. Values reported are the mean over 50 independent trials with standard error of the mean.

	Mushroom	Statlog	Covertype	Financial	Jester	Adult	Song	Census
Simple regret								
Alpha Divergences	0.68 ± 0.04	0.07 ± 0.00	0.31 ± 0.00	$\mathbf{0.00 \pm 0.00}$	2.91 ± 0.04	0.75 ± 0.00	0.90 ± 0.00	0.45 ± 0.00
Alpha Divergences (1)	1.50 ± 0.05	0.08 ± 0.00	0.31 ± 0.00	$\mathbf{0.00 \pm 0.00}$	2.98 ± 0.03	0.75 ± 0.00	0.90 ± 0.00	0.45 ± 0.00
Alpha Divergences (2)	1.51 ± 0.05	0.13 ± 0.00	0.32 ± 0.00	0.00 ± 0.00	3.42 ± 0.05	0.77 ± 0.00	0.90 ± 0.00	0.47 ± 0.00
Alpha Divergences (3)	1.50 ± 0.05	0.08 ± 0.00	0.31 ± 0.00	$\mathbf{0.00 \pm 0.00}$	2.97 ± 0.03	0.75 ± 0.00	0.90 ± 0.00	0.46 ± 0.00
BBBN	0.35 ± 0.05	0.06 ± 0.00	$\mathbf{0.28 \pm 0.00}$	0.01 ± 0.00	$\mathbf{2.78 \pm 0.04}$	$\mathbf{0.67 \pm 0.00}$	0.89 ± 0.00	0.27 ± 0.00
BBBN2	0.45 ± 0.09	0.06 ± 0.00	$\mathbf{0.28 \pm 0.00}$	0.01 ± 0.00	$\mathbf{2.77 \pm 0.04}$	0.70 ± 0.00	0.89 ± 0.00	0.28 ± 0.00
BBBN3	0.81 ± 0.15	0.06 ± 0.00	0.29 ± 0.00	0.03 ± 0.00	$\mathbf{2.78 \pm 0.04}$	0.72 ± 0.00	0.89 ± 0.00	0.30 ± 0.00
BBBN4	1.13 ± 0.18	0.06 ± 0.00	0.29 ± 0.00	0.05 ± 0.00	$\mathbf{2.77 \pm 0.04}$	0.75 ± 0.00	0.90 ± 0.00	0.31 ± 0.00
Bootstrapped NN	0.22 ± 0.04	$\mathbf{0.05 \pm 0.00}$	0.31 ± 0.00	0.01 ± 0.00	2.82 ± 0.04	$\mathbf{0.68 \pm 0.00}$	0.77 ± 0.00	
Bootstrapped NN2	0.51 ± 0.06	$\mathbf{0.05 \pm 0.00}$	0.33 ± 0.00	0.01 ± 0.00	$\mathbf{2.76 \pm 0.03}$	$\mathbf{0.68 \pm 0.00}$		
Bootstrapped NN3	0.36 ± 0.05	$\mathbf{0.05 \pm 0.00}$		0.01 ± 0.00	2.82 ± 0.04	0.68 ± 0.00		
Dropout (RMS3)	0.41 ± 0.06	0.07 ± 0.01	0.32 ± 0.00	0.01 ± 0.00	$\mathbf{2.78 \pm 0.03}$	0.69 ± 0.00	0.78 ± 0.00	0.35 ± 0.01
Dropout (RMS2)	0.56 ± 0.06	0.07 ± 0.01	0.32 ± 0.00	0.01 ± 0.00	2.81 ± 0.03	0.68 ± 0.00	0.77 ± 0.00	0.34 ± 0.01
RMS1	0.33 ± 0.05	0.07 ± 0.00	0.32 ± 0.00	0.01 ± 0.00	2.90 ± 0.04	0.73 ± 0.00	0.90 ± 0.00	0.51 ± 0.01
RMS2	0.42 ± 0.07	0.07 ± 0.01	0.33 ± 0.00	0.01 ± 0.00	$\mathbf{2.72 \pm 0.03}$	0.69 ± 0.00	0.77 ± 0.00	0.35 ± 0.01
RMS2b	0.61 ± 0.08	0.08 ± 0.01	0.32 ± 0.01	0.01 ± 0.00	$\mathbf{2.75 \pm 0.03}$	0.69 ± 0.00		
RMS3	0.39 ± 0.05	0.06 ± 0.00	0.33 ± 0.00	0.01 ± 0.00	2.84 ± 0.04	0.68 ± 0.00	0.77 ± 0.00	0.35 ± 0.01
SGFS	2.59 ± 0.02	0.21 ± 0.00	0.53 ± 0.01	1.28 ± 0.02	3.55 ± 0.08	0.90 ± 0.01	0.87 ± 0.00	0.63 ± 0.02
ConstSGD	2.58 ± 0.01	0.21 ± 0.00	0.53 ± 0.01	1.28 ± 0.02	3.43 ± 0.07	0.87 ± 0.00	0.87 ± 0.00	0.64 ± 0.01
EpsGreedy (RMS1)	$\mathbf{0.08 \pm 0.01}$	0.07 ± 0.00	0.32 ± 0.00	0.01 ± 0.00	2.80 ± 0.04	0.73 ± 0.00	0.90 ± 0.00	0.50 ± 0.01
EpsGreedy (RMS2)	0.32 ± 0.04	0.05 ± 0.00	0.32 ± 0.00	0.01 ± 0.00	$\mathbf{2.78 \pm 0.03}$	0.69 ± 0.00	0.77 ± 0.00	0.33 ± 0.00
EpsGreedy (RMS3)	0.32 ± 0.03	$\mathbf{0.05 \pm 0.00}$	0.32 ± 0.00	0.02 ± 0.00	$\mathbf{2.78 \pm 0.04}$	0.68 ± 0.00	0.77 ± 0.00	0.34 ± 0.01
LinDiagPost	0.80 ± 0.02	0.32 ± 0.00	0.82 ± 0.00	0.03 ± 0.00	$\mathbf{2.75 \pm 0.04}$	0.92 ± 0.00	0.82 ± 0.00	
LinDiagPrecPost	0.37 ± 0.05	0.05 ± 0.00	$\mathbf{0.28 \pm 0.00}$	0.01 ± 0.00	$\mathbf{2.77 \pm 0.04}$	0.69 ± 0.00	0.76 ± 0.00	
LinGreedy	0.51 ± 0.08	0.07 ± 0.01	0.30 ± 0.00	0.01 ± 0.00	$\mathbf{2.73 \pm 0.03}$	0.74 ± 0.01	$\mathbf{0.75 \pm 0.00}$	$\mathbf{0.26 \pm 0.00}$
LinGreedy (eps=0.01)	$\mathbf{0.07 \pm 0.01}$	0.07 ± 0.00	0.29 ± 0.00	0.01 ± 0.00	2.80 ± 0.03	$\mathbf{0.67 \pm 0.00}$	$\mathbf{0.75 \pm 0.00}$	$\mathbf{0.26 \pm 0.00}$
LinGreedy (eps=0.05)	0.24 ± 0.02	0.10 ± 0.00	0.31 ± 0.00	0.06 ± 0.00	2.86 ± 0.03	0.68 ± 0.00	0.76 ± 0.00	0.29 ± 0.00
LinPost	0.29 ± 0.03	0.06 ± 0.00	$\mathbf{0.28 \pm 0.00}$	0.01 ± 0.00	$\mathbf{2.74 \pm 0.04}$	0.69 ± 0.00	0.77 ± 0.00	
LinfullDiagPost	4.10 ± 0.07	0.18 ± 0.00	0.63 ± 0.00	0.00 ± 0.00	2.86 ± 0.03	0.89 ± 0.00	0.80 ± 0.00	
LinfullDiagPrecPost	0.19 ± 0.02	0.05 ± 0.00	$\mathbf{0.28 \pm 0.00}$	0.00 ± 0.00	2.82 ± 0.03	$\mathbf{0.67 \pm 0.00}$	0.76 ± 0.00	
LinfullPost	$\mathbf{0.08 \pm 0.01}$	0.05 ± 0.00	$\mathbf{0.28 \pm 0.00}$	0.00 ± 0.00	2.86 ± 0.03	$\mathbf{0.67 \pm 0.00}$	$\mathbf{0.76 \pm 0.00}$	
Param-Noise	0.49 ± 0.07	$\mathbf{0.05 \pm 0.00}$	0.32 ± 0.00	0.01 ± 0.00	2.87 ± 0.04	0.69 ± 0.00	0.77 ± 0.00	0.34 ± 0.00
Param-Noise2	0.36 ± 0.05	$\mathbf{0.05 \pm 0.00}$	0.33 ± 0.00	0.01 ± 0.00	2.83 ± 0.04	0.69 ± 0.00		
Uniform	4.88 ± 0.07	0.86 ± 0.00	0.86 ± 0.00	1.25 ± 0.02	5.03 ± 0.07	0.93 ± 0.00	0.91 ± 0.00	0.89 ± 0.00

Table 6: Cumulative regret incurred by models using algorithms in Section 3 on the bandits described in Section A. Values reported are the mean over 50 independent trials with standard error of the mean.

	Mushroom	Statlog	Covertype	Financial	Jester	Adult	Song	Census
Cumulative regret								
Alpha Divergences	132716.63 ± 77.81	7194.51 ± 314.41	56986.69 ± 993.22	2266.20 ± 42.44	68451.38 ± 688.37	39525.59 ± 11.23	130934.36 ± 285.05	85197.81 ± 1616.78
Alpha Divergences (1)	133003.78 ± 87.58	7215.88 ± 643.04	50695.49 ± 644.20	1902.36 ± 32.63	69533.59 ± 516.27	39613.94 ± 11.12	131828.64 ± 323.62	90435.32 ± 387.93
Alpha Divergences (2)	133348.67 ± 72.30	7249.18 ± 710.67	88053.18 ± 1006.41	1985.49 ± 22.67	73667.43 ± 503.06	41032.53 ± 34.43	132133.14 ± 339.83	91103.59 ± 395.98
Alpha Divergences (3)	135117.24 ± 234.55	16046.47 ± 494.44	77401.61 ± 42.26	4211.99 ± 115.26	65479.19 ± 827.17	39565.92 ± 14.92	129647.09 ± 36.91	89316.76 ± 28.49
BBBN	8742.14 ± 499.99	4699.53 ± 569.92	74836.94 ± 2777.46	1069.68 ± 60.32	68050.69 ± 640.33	39481.53 ± 249.02	131424.96 ± 497.53	87954.58 ± 3727.46
BBBN2	14235.41 ± 2840.14	9409.80 ± 0.51	80975.86 ± 1076.50	1842.43 ± 133.21	64656.89 ± 762.38	39830.90 ± 164.23	131252.06 ± 462.30	96704.73 ± 2222.35
BBBN3	10528.67 ± 2598.62	9409.69 ± 0.53	80854.45 ± 1068.09	2130.46 ± 119.20	63544.37 ± 746.37	39695.86 ± 135.63	131104.18 ± 431.86	95463.66 ± 2089.41
BBBN4	25671.63 ± 6639.45	9409.43 ± 0.52	79617.37 ± 873.59	2737.35 ± 170.66	62290.74 ± 783.10	39818.61 ± 151.73	130941.76 ± 434.18	97504.47 ± 2386.58
Bootstrapped NN	18936.84 ± 2767.90	**251.57 ± 9.55**	34291.52 ± 139.85	444.17 ± 15.63	70077.99 ± 614.25	34936.57 ± 106.35	122917.00 ± 522.95	
Bootstrapped NN2	14533.57 ± 2937.69	**236.88 ± 7.45**	33286.15 ± 72.30	509.49 ± 26.04	68655.84 ± 534.28	34688.16 ± 98.02		
Bootstrapped NN3	13713.06 ± 1462.60	**240.98 ± 7.76**	34002.60 ± 307.92	436.97 ± 20.76	69905.34 ± 589.11	34707.67 ± 117.20		
Dropout (RMS3)	13167.65 ± 2547.61	1179.27 ± 314.76	39911.88 ± 651.30	880.08 ± 33.95	63370.26 ± 713.73	35335.22 ± 199.43	125518.61 ± 423.34	53555.06 ± 513.43
Dropout (RMS2)	13636.43 ± 2506.73	877.18 ± 218.15	39415.18 ± 296.53	832.84 ± 31.94	63106.73 ± 709.62	35429.27 ± 186.26	125761.31 ± 484.77	53168.28 ± 310.08
GP	29571.92 ± 3222.91	1699.53 ± 552.93		150.91 ± 3.70	71401.09 ± 890.50	38167.29 ± 294.07		
Neural Linear	**5607.86 ± 222.26**	395.96 ± 2.71	38658.86 ± 177.74	617.07 ± 4.98	72167.76 ± 485.55	36544.67 ± 118.55	128459.02 ± 191.44	76138.53 ± 811.20
Neural Linear (2)	**5450.00 ± 201.86**	340.90 ± 2.28	38462.20 ± 212.75	542.66 ± 5.23	71859.58 ± 393.22	36660.35 ± 115.09	128548.11 ± 219.35	73789.41 ± 561.62
Neural Linear (3)	**5605.41 ± 226.80**	405.55 ± 2.70	39780.43 ± 274.96	637.68 ± 4.70	70999.02 ± 447.50	36767.88 ± 84.79	128757.95 ± 269.56	79708.10 ± 822.44
Neural Linear (4)	**5485.10 ± 238.67**	439.39 ± 2.33	38559.20 ± 245.55	704.39 ± 4.85	71488.45 ± 504.08	36806.55 ± 121.40	128530.46 ± 182.33	75623.86 ± 642.13
RMS1	21194.49 ± 1018.18	441.08 ± 135.07	32325.94 ± 210.50	920.59 ± 31.73	71710.79 ± 732.01	37829.04 ± 76.22	131099.00 ± 453.18	66968.00 ± 578.39
RMS2	16623.16 ± 3926.97	898.92 ± 238.13	35498.71 ± 206.39	575.03 ± 37.29	67751.88 ± 604.91	35033.96 ± 81.33	123471.63 ± 382.81	51224.78 ± 179.79
RMS2b	32264.69 ± 5763.25	767.35 ± 211.74	36703.72 ± 317.67	444.64 ± 24.71	65863.08 ± 563.91	36035.96 ± 72.98		
RMS3	16364.80 ± 3717.34	1060.86 ± 272.69	35678.06 ± 227.20	604.13 ± 34.46	66618.89 ± 537.08	34888.08 ± 98.85	123605.57 ± 428.16	51439.69 ± 253.06
SGFS	14673.67 ± 2502.17	1424.88 ± 167.70	47029.63 ± 1034.16	1376.07 ± 24.96	64794.65 ± 601.08	36837.51 ± 172.55	132796.18 ± 393.51	54068.98 ± 602.50
Sparse GP				1521.71 ± 26.21				
ConstSGD	64607.96 ± 7698.43	2532.43 ± 527.98	**28904.94 ± 1002.51**	4182.68 ± 132.90	66774.08 ± 765.29	37480.53 ± 121.52	130741.45 ± 412.70	68232.27 ± 2324.51
EpsGreedy (RMS1)	23069.80 ± 790.68	454.92 ± 23.44	32430.86 ± 205.42	966.63 ± 28.16	72257.09 ± 711.29	37895.27 ± 86.70	129029.82 ± 122.93	66022.16 ± 660.48
EpsGreedy (RMS2)	16492.65 ± 3489.04	705.92 ± 100.06	35289.80 ± 185.12	518.30 ± 20.91	66233.05 ± 570.17	35081.47 ± 84.54	123011.41 ± 336.39	51323.53 ± 200.49
EpsGreedy (RMS3)	12176.63 ± 2547.23	792.39 ± 125.40	35264.33 ± 170.17	586.45 ± 22.30	66351.72 ± 662.56	34989.65 ± 96.29	123017.86 ± 316.29	51402.10 ± 240.35
LinDiagPost	43292.76 ± 438.74	19122.59 ± 12.08	122805.88 ± 22.07	455.22 ± 2.54	**55835.64 ± 466.27**	41871.10 ± 8.60	127089.98 ± 30.67	
LinDiagPrecPost	23212.86 ± 3907.39	2806.59 ± 7.06	44242.10 ± 28.66	217.17 ± 2.12	**55806.69 ± 571.80**	33543.33 ± 12.10	117498.69 ± 25.12	
LinGreedy	34979.49 ± 4867.12	4220.14 ± 234.25	45386.84 ± 136.39	**103.50 ± 6.72**	**56863.58 ± 568.70**	34863.59 ± 295.98	**114639.69 ± 24.22**	**40969.00 ± 28.41**
LinGreedy (eps=0.01)	8291.63 ± 433.26	3883.98 ± 144.74	44486.39 ± 96.87	139.57 ± 5.67	57107.81 ± 551.95	32187.45 ± 104.34	114865.63 ± 23.87	41833.76 ± 28.42
LinGreedy (eps=0.05)	14425.82 ± 141.00	4752.49 ± 58.30	47587.06 ± 42.11	311.86 ± 5.22	58697.32 ± 412.85	**31807.94 ± 20.09**	115774.14 ± 24.33	45412.80 ± 26.24
LinPost	15004.08 ± 1650.29	2848.94 ± 6.49	44243.69 ± 25.41	344.33 ± 2.51	**56337.76 ± 473.46**	33222.84 ± 12.77	117684.73 ± 23.17	
LinfullDiagPost	212654.18 ± 326.39	10549.12 ± 9.23	94939.69 ± 41.94	330.37 ± 2.69	60228.70 ± 577.95	40379.73 ± 9.54	123671.67 ± 17.40	129505.00 ± nan
LinfullDiagPrecPost	8511.22 ± 880.59	2737.57 ± 10.18	43776.00 ± 27.06	191.53 ± 2.15	57756.09 ± 419.60	32329.76 ± 13.60	116112.80 ± 21.56	
LinfullPost	**5796.84 ± 601.37**	2735.80 ± 7.36	43729.78 ± 28.05	268.58 ± 2.01	58939.40 ± 419.12	32246.82 ± 12.64	116292.67 ± 18.89	
Param-Noise	6797.04 ± 375.54	546.69 ± 61.54	34475.14 ± 133.06	903.59 ± 36.97	65651.99 ± 506.69	36761.45 ± 36.26	128842.67 ± 90.27	52261.29 ± 97.10
Param-Noise2	7568.47 ± 352.51	573.45 ± 72.30	35857.58 ± 250.08	883.92 ± 26.07	67341.83 ± 554.78	37340.63 ± 42.61		
Uniform	245231.94 ± 326.81	37293.59 ± 8.28	128580.61 ± 21.71	4709.38 ± 63.27	96931.64 ± 1084.94	41997.86 ± 7.69	135017.10 ± 17.39	133321.10 ± 16.62

Table 7: Simple regret incurred by models using algorithms in Section 3 on the bandits described in Section A. Simple regret was approximated by averaging the regret over the final 500 steps. Values reported are the mean over 50 independent trials with standard error of the mean.

	Mushroom	Statlog	Covertype	Financial	Jester	Adult	Song	Census
Simple regret								
Alpha Divergences	2.59 ± 0.02	0.11 ± 0.01	0.29 ± 0.01	0.34 ± 0.02	3.38 ± 0.05	0.88 ± 0.00	0.87 ± 0.00	0.55 ± 0.02
Alpha Divergences (1)	2.61 ± 0.02	0.15 ± 0.01	0.26 ± 0.01	0.25 ± 0.01	3.43 ± 0.05	0.88 ± 0.00	0.88 ± 0.00	0.61 ± 0.01
Alpha Divergences (2)	2.58 ± 0.02	0.15 ± 0.02	0.60 ± 0.01	0.30 ± 0.01	3.63 ± 0.04	0.90 ± 0.00	0.88 ± 0.00	0.60 ± 0.01
Alpha Divergences (3)	2.64 ± 0.02	0.25 ± 0.01	0.51 ± 0.00	1.04 ± 0.05	3.37 ± 0.05	0.88 ± 0.00	0.86 ± 0.00	0.59 ± 0.00
BBBN	$\mathbf{0.03 \pm 0.01}$	0.11 ± 0.01	0.49 ± 0.02	0.24 ± 0.02	3.35 ± 0.04	0.87 ± 0.01	0.88 ± 0.00	0.57 ± 0.03
BBBN2	0.11 ± 0.05	0.22 ± 0.00	0.54 ± 0.01	0.44 ± 0.04	3.22 ± 0.05	0.88 ± 0.00	0.87 ± 0.00	0.64 ± 0.02
BBBN3	0.13 ± 0.06	0.22 ± 0.00	0.53 ± 0.01	0.48 ± 0.03	3.16 ± 0.05	0.88 ± 0.00	0.87 ± 0.00	0.62 ± 0.01
BBBN4	0.43 ± 0.13	0.22 ± 0.00	0.53 ± 0.01	0.62 ± 0.05	3.20 ± 0.05	0.88 ± 0.00	0.87 ± 0.00	0.64 ± 0.02
Bootstrapped NN	0.18 ± 0.05	$\mathbf{0.00 \pm 0.00}$	0.20 ± 0.00	0.05 ± 0.00	3.51 ± 0.04	0.73 ± 0.00	0.78 ± 0.01	
Bootstrapped NN2	0.15 ± 0.05	$\mathbf{0.00 \pm 0.00}$	0.19 ± 0.00	0.06 ± 0.01	3.42 ± 0.04	0.73 ± 0.00		
Bootstrapped NN3	0.12 ± 0.02	$\mathbf{0.00 \pm 0.00}$	0.19 ± 0.01	0.04 ± 0.00	3.53 ± 0.04	0.73 ± 0.00		
Dropout (RMS3)	0.16 ± 0.05	0.02 ± 0.01	0.24 ± 0.01	0.16 ± 0.01	3.22 ± 0.04	0.76 ± 0.00	0.82 ± 0.00	0.33 ± 0.00
Dropout (RMS2)	0.13 ± 0.05	0.01 ± 0.00	0.24 ± 0.00	0.15 ± 0.01	3.19 ± 0.04	0.76 ± 0.01	0.82 ± 0.00	0.32 ± 0.00
GP	0.54 ± 0.07	0.04 ± 0.01		$\mathbf{0.00 \pm 0.00}$	3.74 ± 0.06	0.84 ± 0.01		
Neural Linear	$\mathbf{0.03 \pm 0.01}$	$\mathbf{0.00 \pm 0.00}$	0.21 ± 0.00	0.04 ± 0.00	3.57 ± 0.03	0.77 ± 0.00	0.85 ± 0.00	0.47 ± 0.01
Neural Linear (2)	$\mathbf{0.02 \pm 0.01}$	$\mathbf{0.00 \pm 0.00}$	0.22 ± 0.00	0.03 ± 0.00	3.55 ± 0.03	0.77 ± 0.00	0.85 ± 0.00	0.45 ± 0.01
Neural Linear (3)	0.04 ± 0.01	$\mathbf{0.00 \pm 0.00}$	0.22 ± 0.00	0.04 ± 0.00	3.50 ± 0.04	0.78 ± 0.00	0.85 ± 0.00	0.50 ± 0.01
Neural Linear (4)	$\mathbf{0.02 \pm 0.01}$	$\mathbf{0.00 \pm 0.00}$	0.21 ± 0.00	0.04 ± 0.00	3.57 ± 0.04	0.79 ± 0.00	0.85 ± 0.00	0.46 ± 0.01
RMS1	0.27 ± 0.03	0.01 ± 0.00	0.17 ± 0.00	0.19 ± 0.01	3.72 ± 0.05	0.82 ± 0.00	0.87 ± 0.00	0.41 ± 0.00
RMS2	0.26 ± 0.08	0.02 ± 0.01	0.20 ± 0.00	0.09 ± 0.01	3.40 ± 0.04	0.74 ± 0.00	0.79 ± 0.00	0.31 ± 0.00
RMS2b	0.56 ± 0.12	0.01 ± 0.00	0.21 ± 0.00	0.06 ± 0.01	3.28 ± 0.04	0.76 ± 0.00		
RMS3	0.25 ± 0.07	0.02 ± 0.01	0.20 ± 0.00	0.09 ± 0.01	3.30 ± 0.04	0.74 ± 0.00	0.80 ± 0.00	0.31 ± 0.00
SGFS	0.21 ± 0.04	0.02 ± 0.00	0.29 ± 0.01	0.21 ± 0.01	3.25 ± 0.04	0.77 ± 0.01	0.89 ± 0.00	0.32 ± 0.01
Sparse GP				0.29 ± 0.01				
ConstSGD	0.99 ± 0.17	0.05 ± 0.01	$\mathbf{0.13 \pm 0.01}$	1.11 ± 0.05	3.44 ± 0.05	0.82 ± 0.00	0.87 ± 0.00	0.43 ± 0.02
EpsGreedy (RMS1)	0.29 ± 0.02	0.01 ± 0.00	0.17 ± 0.00	0.19 ± 0.01	3.71 ± 0.05	0.82 ± 0.00	0.85 ± 0.00	0.40 ± 0.01
EpsGreedy (RMS2)	0.22 ± 0.07	0.00 ± 0.00	0.20 ± 0.00	0.06 ± 0.00	3.30 ± 0.03	0.74 ± 0.00	0.79 ± 0.00	0.31 ± 0.00
EpsGreedy (RMS3)	0.13 ± 0.06	0.00 ± 0.00	0.20 ± 0.00	0.07 ± 0.01	3.29 ± 0.04	0.74 ± 0.00	0.79 ± 0.00	0.31 ± 0.00
LinDiagPost	0.84 ± 0.02	0.32 ± 0.00	0.82 ± 0.00	0.03 ± 0.00	$\mathbf{2.70 \pm 0.03}$	0.93 ± 0.00	0.82 ± 0.00	
LinDiagPrecPost	0.43 ± 0.08	0.06 ± 0.00	0.28 ± 0.00	0.01 ± 0.00	$\mathbf{2.73 \pm 0.04}$	0.70 ± 0.00	0.77 ± 0.00	
LinGreedy	0.69 ± 0.10	0.08 ± 0.01	0.30 ± 0.00	0.01 ± 0.00	2.78 ± 0.04	0.75 ± 0.01	$\mathbf{0.75 \pm 0.00}$	$\mathbf{0.26 \pm 0.00}$
LinGreedy (eps=0.01)	0.09 ± 0.01	0.07 ± 0.00	0.29 ± 0.00	0.01 ± 0.00	2.81 ± 0.04	$\mathbf{0.68 \pm 0.00}$	$\mathbf{0.75 \pm 0.00}$	0.26 ± 0.00
LinGreedy (eps=0.05)	0.24 ± 0.02	0.10 ± 0.00	0.31 ± 0.00	0.07 ± 0.00	2.88 ± 0.03	0.69 ± 0.00	0.76 ± 0.00	0.29 ± 0.00
LinPost	0.27 ± 0.03	0.06 ± 0.00	0.28 ± 0.00	0.01 ± 0.00	$\mathbf{2.76 \pm 0.03}$	0.69 ± 0.00	0.77 ± 0.00	
LinfullDiagPost	4.25 ± 0.07	0.18 ± 0.00	0.63 ± 0.00	0.00 ± 0.00	2.88 ± 0.04	0.89 ± 0.00	0.80 ± 0.00	$0.82 \pm nan$
LinfullDiagPrecPost	0.14 ± 0.02	0.06 ± 0.00	0.28 ± 0.00	$\mathbf{0.00 \pm 0.00}$	2.84 ± 0.03	$\mathbf{0.68 \pm 0.00}$	0.76 ± 0.00	
LinfullPost	0.09 ± 0.01	0.05 ± 0.00	0.28 ± 0.00	$\mathbf{0.00 \pm 0.00}$	2.88 ± 0.04	$\mathbf{0.68 \pm 0.00}$	0.76 ± 0.00	
Param-Noise	0.07 ± 0.01	0.01 ± 0.00	0.19 ± 0.00	0.20 ± 0.01	3.26 ± 0.04	0.79 ± 0.00	0.84 ± 0.00	0.32 ± 0.00
Param-Noise2	0.07 ± 0.01	0.01 ± 0.00	0.19 ± 0.00	0.19 ± 0.01	3.35 ± 0.04	0.80 ± 0.00		
Uniform	4.89 ± 0.08	0.86 ± 0.00	0.86 ± 0.00	1.27 ± 0.02	5.06 ± 0.06	0.93 ± 0.00	0.90 ± 0.00	0.88 ± 0.00

Table 8: Elapsed time for algorithms in Section 3 on the bandits described in Section A. Values reported are the mean over 50 independent trials with standard error of the mean.

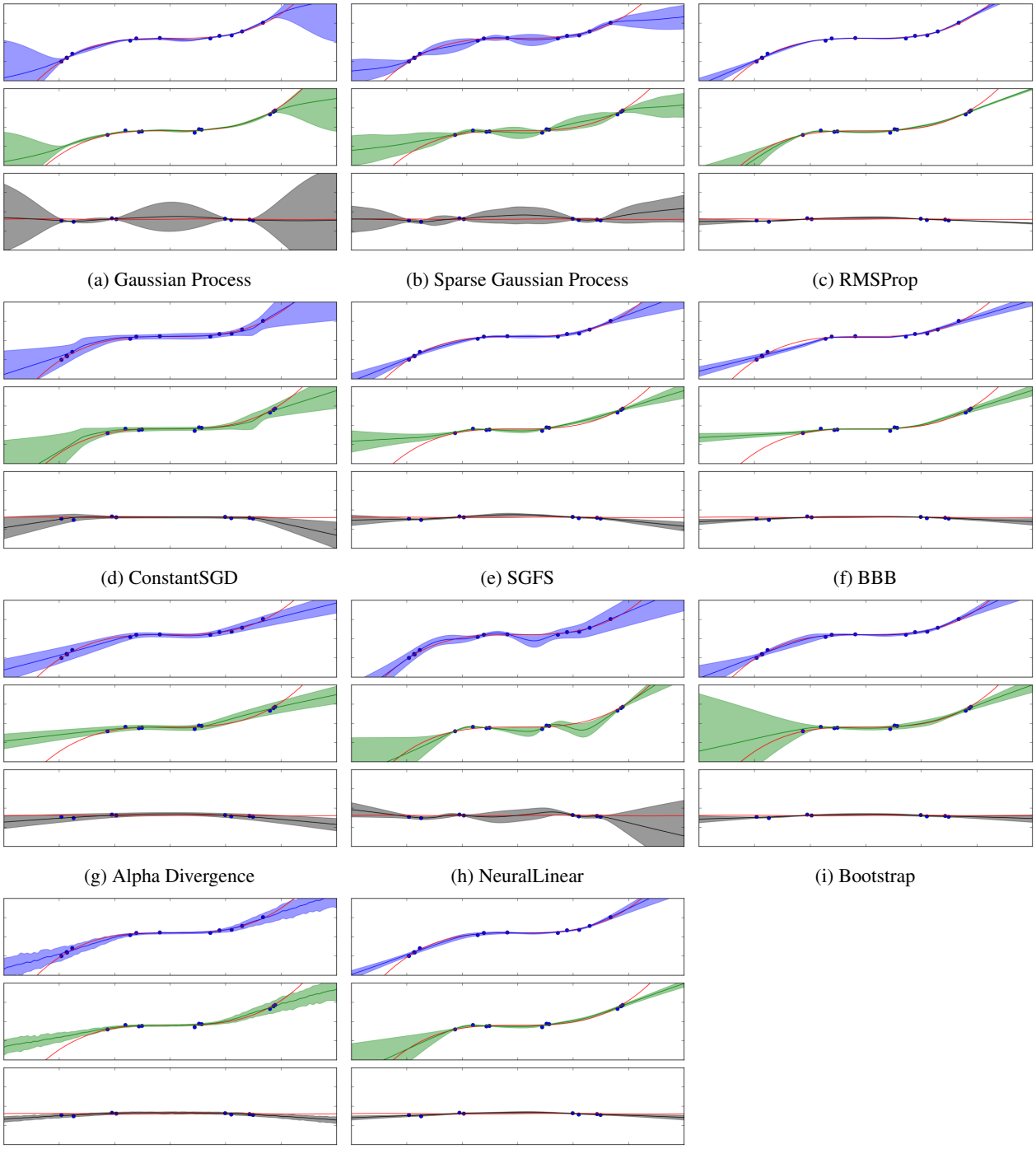
	Mushroom	Statlog	Covertype	Financial	Jester	Adult	Song	Census
Time								
Alpha Divergences	13367.14 ± 891.27	11757.35 ± 910.81	41736.29 ± 2361.46	6973.55 ± 575.13	8686.43 ± 761.34	14110.39 ± 1289.62	50138.43 ± 2833.96	72146.26 ± 9119.93
Alpha Divergences (1)	13669.35 ± 1140.05	11576.81 ± 874.22	41721.42 ± 2418.27	6894.85 ± 554.49	8690.00 ± 751.55	14527.61 ± 1078.94	49983.58 ± 3294.11	73477.73 ± 4622.28
Alpha Divergences (2)	13387.38 ± 1126.83	11693.04 ± 787.83	42018.41 ± 2446.65	6961.08 ± 616.10	8937.71 ± 845.46	14362.37 ± 1024.97	50141.78 ± 3322.18	69776.50 ± 3567.19
Alpha Divergences (3)	13780.62 ± 1089.39	11766.47 ± 1036.78	43113.45 ± 2794.67	6864.50 ± 573.95	8827.69 ± 935.92	14816.87 ± 1102.47	49938.50 ± 3976.92	69659.96 ± 5566.70
BBBN	6457.48 ± 278.77	5152.88 ± 309.05	23684.49 ± 1653.45	3061.64 ± 186.65	4175.06 ± 374.11	6649.55 ± 411.72	29480.70 ± 2105.30	45865.29 ± 2169.67
BBBN2	6498.33 ± 304.68	5256.36 ± 321.92	24121.98 ± 1663.14	3112.36 ± 186.61	3934.73 ± 284.94	6700.65 ± 482.04	29354.28 ± 1334.66	46015.00 ± 1584.11
BBBN3	6650.46 ± 286.71	5168.49 ± 331.62	23548.79 ± 1346.59	3113.82 ± 172.59	3925.66 ± 297.07	6502.71 ± 382.48	30049.20 ± 3928.53	45445.03 ± 1242.57
BBBN4	6558.96 ± 280.58	5325.81 ± 425.05	23656.59 ± 1539.29	3094.35 ± 202.55	3924.93 ± 329.01	6603.83 ± 407.91	29580.81 ± 1606.40	45662.91 ± 5373.47
Bootstrapped NN	10588.83 ± 522.54	7201.76 ± 409.87	77536.48 ± 5086.24	317.34 ± 31.44	2301.51 ± 190.28	9740.34 ± 703.28	83769.83 ± 2572.40	
Bootstrapped NN2	12680.24 ± 519.90	8051.58 ± 519.94	88045.64 ± 5342.62	328.00 ± 31.00	2478.81 ± 216.01	10719.69 ± 604.24		
Bootstrapped NN3	25194.89 ± 1489.80	16281.40 ± 1002.46	166989.56 ± 2886.58	652.59 ± 51.89	5088.23 ± 395.98	21334.57 ± 1168.38		
Dropout (RMS3)	2578.84 ± 136.64	1782.88 ± 146.39	17452.36 ± 1448.07	80.46 ± 8.31	615.02 ± 73.32	2285.24 ± 167.91	24435.55 ± 3223.85	29513.20 ± 711.09
Dropout (RMS2)	3558.95 ± 172.63	2353.51 ± 158.63	27896.84 ± 2075.69	88.92 ± 7.11	686.18 ± 35.49	2915.46 ± 175.25	32450.92 ± 1537.29	55323.67 ± 13304.08
GP	100171.59 ± 3198.56	86988.82 ± 2216.96		5510.14 ± 395.27	38414.09 ± 2548.57	94272.16 ± 4754.11		
Neural Linear	3343.31 ± 138.18	3248.11 ± 222.03	34583.14 ± 2618.35	139.75 ± 10.00	1012.81 ± 102.40	4508.93 ± 358.67	52589.62 ± 8068.16	45019.37 ± 1402.27
Neural Linear (2)	3348.43 ± 137.51	3261.88 ± 229.23	34435.65 ± 2315.51	142.44 ± 10.71	1031.93 ± 88.97	4502.67 ± 315.09	51615.50 ± 7119.53	44433.45 ± 1265.02
Neural Linear (3)	5391.12 ± 301.39	4820.27 ± 335.30	56795.16 ± 2436.51	191.32 ± 9.69	1475.32 ± 156.95	6389.41 ± 502.72	72080.60 ± 4076.48	58931.75 ± 1509.85
Neural Linear (4)	3327.70 ± 151.06	3194.69 ± 246.87	34856.04 ± 3037.70	141.06 ± 9.29	1045.63 ± 94.86	4553.73 ± 343.62	52081.19 ± 7632.04	44901.67 ± 1105.74
RMS1	2338.24 ± 95.61	1468.39 ± 81.29	16557.72 ± 889.75	57.68 ± 3.63	455.06 ± 26.54	1946.99 ± 77.55	21105.69 ± 1513.89	36738.58 ± 1234.41
RMS2	3314.91 ± 148.87	2195.91 ± 151.30	25907.61 ± 2186.73	74.08 ± 5.71	646.93 ± 57.87	2807.81 ± 215.13	30909.99 ± 2585.80	36615.44 ± 1577.72
RMS2b	17848.54 ± 965.86	13417.36 ± 825.72	135638.61 ± 7214.45	535.69 ± 32.86	4122.23 ± 347.13	16156.73 ± 1030.07		
RMS3	2440.07 ± 129.60	1575.04 ± 107.26	16155.28 ± 1097.64	67.01 ± 5.58	522.39 ± 64.11	2111.96 ± 170.05	22085.67 ± 2508.38	28822.81 ± 429.23
SGFS	3473.28 ± 135.51	2412.09 ± 132.75	20521.07 ± 1899.33	129.77 ± 6.79	788.42 ± 44.83	2869.21 ± 156.73	25936.71 ± 4537.24	41267.86 ± 2540.52
Sparse GP				27920.75 ± 2112.72				
ConstSGD	2707.37 ± 182.81	1720.12 ± 117.90	17341.78 ± 1246.59	78.18 ± 8.34	582.95 ± 47.02	2317.05 ± 191.97	22812.89 ± 2020.08	29449.44 ± 783.85
EpsGreedy (RMS1)	2410.84 ± 104.88	1564.61 ± 109.10	16459.43 ± 1271.37	65.79 ± 6.57	502.46 ± 50.37	2064.02 ± 172.90	22052.83 ± 2015.59	28580.43 ± 737.22
EpsGreedy (RMS2)	3359.89 ± 190.40	2256.25 ± 217.52	26608.50 ± 2045.96	73.65 ± 6.20	649.86 ± 69.78	2888.56 ± 214.73	34077.35 ± 6629.02	35980.86 ± 510.18
EpsGreedy (RMS3)	2438.01 ± 98.74	1585.12 ± 96.43	16275.39 ± 1145.56	60.07 ± 3.53	511.70 ± 63.72	2072.89 ± 167.94	22492.14 ± 2626.13	28847.94 ± 650.94
LinDiagPost	1984.53 ± 79.28	336.40 ± 26.26	6914.52 ± 441.53	8.00 ± 0.62	117.35 ± 13.80	1133.31 ± 72.10	12912.27 ± 2064.63	
LinDiagPrecPost	2085.55 ± 231.96	397.56 ± 31.10	9579.13 ± 812.35	8.69 ± 0.90	120.08 ± 14.13	1250.88 ± 97.25	15455.07 ± 3045.85	
LinGreedy	1924.57 ± 271.59	337.49 ± 22.36	9050.14 ± 634.18	2.36 ± 0.31	72.91 ± 8.42	816.23 ± 125.82	14559.73 ± 2169.27	45132.85 ± 606.83
LinGreedy (eps=0.01)	1794.61 ± 82.75	335.72 ± 27.93	8922.09 ± 740.74	2.44 ± 0.28	71.77 ± 9.61	675.06 ± 69.60	14143.25 ± 1788.65	44611.27 ± 907.35
LinGreedy (eps=0.05)	1803.65 ± 72.65	315.03 ± 25.87	8507.67 ± 571.55	2.35 ± 0.22	68.39 ± 9.18	674.39 ± 64.34	13875.89 ± 2289.28	43741.15 ± 583.62
LinPost	2282.28 ± 87.45	408.38 ± 41.71	10480.69 ± 811.83	15.54 ± 1.03	199.55 ± 19.03	2946.89 ± 186.56	18286.39 ± 1598.92	
LinfullDiagPost	1963.57 ± 80.02	400.96 ± 40.41	7814.68 ± 509.26	10.29 ± 0.95	126.29 ± 13.49	1207.94 ± 102.73	12940.26 ± 720.02	$80020.21 \pm nan$
LinfullDiagPrecPost	1970.25 ± 73.96	420.90 ± 31.67	9429.35 ± 654.40	10.17 ± 0.98	117.83 ± 11.31	1293.37 ± 96.94	18509.99 ± 3862.95	
LinfullPost	2303.59 ± 90.18	428.79 ± 29.66	10504.58 ± 652.50	17.42 ± 1.37	210.17 ± 18.07	2988.98 ± 198.08	19814.67 ± 3279.84	
Param-Noise	3545.93 ± 193.72	2685.68 ± 211.07	20877.39 ± 2031.67	153.19 ± 13.88	969.21 ± 117.95	3172.61 ± 243.87	26490.95 ± 2713.47	32049.97 ± 769.50
Param-Noise2	25966.38 ± 2006.88	20496.04 ± 1567.80	155691.19 ± 4691.00	1163.84 ± 96.36	7816.45 ± 638.04	24046.79 ± 1520.70		
Uniform	$\mathbf{4.63 \pm 0.38}$	$\mathbf{4.23 \pm 0.86}$	$\mathbf{36.87 \pm 2.68}$	$\mathbf{0.14 \pm 0.03}$	$\mathbf{1.09 \pm 0.15}$	$\mathbf{4.01 \pm 0.47}$	$\mathbf{44.01 \pm 4.99}$	$\mathbf{36.37 \pm 2.48}$

Table 9: Cumulative regret incurred on the Wheel Bandit problem with increasing values of δ. Values reported are the mean over 50 independent trials with standard error of the mean.

	$\delta = 0.5$	$\delta = 0.7$	$\delta = 0.9$	$\delta = 0.95$	$\delta = 0.99$
Cumulative regret					
Alpha Divergences	728697.89 ± 418.01	496275.47 ± 513.11	181321.48 ± 1738.12	93743.66 ± 633.58	19411.53 ± 140.67
Alpha Divergences (1)	730006.13 ± 481.11	496047.12 ± 520.96	183339.89 ± 893.35	91956.53 ± 912.62	19309.46 ± 169.21
Alpha Divergences (2)	727932.43 ± 2081.80	496328.45 ± 492.93	180447.41 ± 1525.17	91970.51 ± 864.57	18867.71 ± 176.24
Alpha Divergences (3)	103953.93 ± 7481.23	83903.77 ± 6191.86	77213.63 ± 5320.09	56494.01 ± 2052.91	17441.69 ± 232.37
BBBN	49250.99 ± 21154.52	78253.58 ± 21520.15	113341.63 ± 9447.34	81693.05 ± 2632.31	19248.39 ± 138.59
BBBN2	76893.21 ± 20123.60	85567.60 ± 22756.20	135533.12 ± 8887.75	83438.85 ± 2742.78	19104.30 ± 163.23
BBBN3	45801.74 ± 16279.37	80239.77 ± 19720.11	126083.45 ± 9522.96	82848.16 ± 2473.31	19161.02 ± 189.91
BBBN4	63175.17 ± 19101.01	97109.72 ± 23477.52	127792.97 ± 9015.56	81388.83 ± 3224.03	19439.77 ± 177.22
Bootstrapped NN	47925.12 ± 16500.71	149129.32 ± 25862.48	137230.10 ± 8134.48	87557.63 ± 1947.46	18937.61 ± 219.79
Bootstrapped NN2	70490.90 ± 21918.31	172866.51 ± 25198.32	139574.75 ± 7461.54	85185.45 ± 2430.34	19150.15 ± 201.04
Bootstrapped NN3	34567.75 ± 15620.36	114212.95 ± 21215.73	137551.25 ± 7687.96	81889.31 ± 2731.27	19044.73 ± 179.29
Dropout (RMS3)	261219.62 ± 26580.84	234244.72 ± 21127.44	148387.58 ± 5501.94	86991.92 ± 2482.39	18821.18 ± 242.30
Dropout (RMS2)	258164.63 ± 27779.93	243186.25 ± 19010.21	153491.17 ± 5724.94	86370.65 ± 1940.96	19510.63 ± 141.64
GP	20834.05 ± 3830.98	19886.13 ± 4204.80	54732.75 ± 5363.75	51701.96 ± 1919.92	18007.77 ± 275.56
Neural Linear	**6348.28 ± 109.69**	**7201.55 ± 123.80**	7145.15 ± 241.10	7912.27 ± 400.38	9356.57 ± 292.69
Neural Linear (2)	**6452.93 ± 107.47**	**7112.65 ± 117.77**	**6564.38 ± 161.69**	9111.86 ± 774.19	9851.37 ± 382.17
Neural Linear (3)	**6506.27 ± 111.03**	**6936.71 ± 145.10**	**6663.85 ± 125.21**	**7548.97 ± 488.17**	9065.57 ± 314.77
Neural Linear (4)	**6318.70 ± 96.69**	**6998.10 ± 95.96**	**6571.24 ± 171.49**	**7217.88 ± 309.82**	8932.78 ± 250.55
RMS1	147114.59 ± 33931.87	128248.19 ± 23505.00	117918.47 ± 10441.85	74625.63 ± 4037.23	19388.24 ± 165.82
RMS2	128681.52 ± 24897.82	180326.40 ± 26718.82	137985.99 ± 7455.84	82817.58 ± 2327.20	18843.66 ± 265.74
RMS2b	117569.87 ± 24828.02	154039.90 ± 20285.34	125651.13 ± 7650.26	80251.13 ± 2836.20	18920.44 ± 294.55
RMS3	147850.35 ± 27856.41	180080.38 ± 24836.24	136381.67 ± 6481.16	83589.82 ± 2547.23	19507.71 ± 200.10
SGFS	44735.83 ± 6493.79	41720.83 ± 5700.98	29315.24 ± 2433.11	20397.15 ± 1224.44	8524.28 ± 279.44
Sparse GP					
ConstSGD	131024.19 ± 30098.30	134391.98 ± 22606.75	130556.06 ± 8761.83	88890.23 ± 1741.28	19525.55 ± 146.53
EpsGreedy (RMS1)	38218.87 ± 11704.53	50017.20 ± 13128.10	66488.62 ± 8556.89	57069.28 ± 4364.23	18722.99 ± 388.63
EpsGreedy (RMS2)	31915.72 ± 11650.04	64550.77 ± 13317.88	111535.56 ± 6622.91	74039.10 ± 3036.65	19379.29 ± 127.48
EpsGreedy (RMS3)	30508.48 ± 5030.47	109643.72 ± 19877.01	111261.27 ± 6775.62	68193.40 ± 3227.37	18982.18 ± 271.50
LinDiagPost	**6579.92 ± 196.19**	**7227.36 ± 306.32**	7680.92 ± 216.25	**7173.93 ± 265.48**	**7068.75 ± 407.09**
LinDiagPrecPost	7077.44 ± 240.39	7439.27 ± 280.24	8246.02 ± 293.44	7850.89 ± 263.80	**7665.37 ± 420.59**
LinGreedy	388131.58 ± 28837.23	287658.08 ± 17286.23	165242.55 ± 4710.93	82052.79 ± 2444.88	19614.05 ± 169.73
LinGreedy (eps=0.01)	68769.65 ± 6062.76	69302.76 ± 5180.20	70043.77 ± 3596.73	55884.88 ± 2811.55	18068.70 ± 294.55
LinGreedy (eps=0.05)	46304.06 ± 1615.87	38427.00 ± 1392.45	29472.56 ± 1182.94	26390.11 ± 1641.24	13880.51 ± 305.02
LinPost	**6493.44 ± 240.24**	**6959.34 ± 221.55**	7516.29 ± 265.58	8226.80 ± 391.62	7970.30 ± 383.41
LinfullDiagPost	14432.68 ± 218.58	11358.35 ± 183.75	9379.56 ± 188.83	9143.24 ± 312.57	10036.79 ± 500.10
LinfullDiagPrecPost	9124.25 ± 191.97	8268.69 ± 259.71	8173.78 ± 269.69	7848.91 ± 311.39	10746.86 ± 462.03
LinfullPost	9624.54 ± 180.72	8457.70 ± 205.00	8020.44 ± 247.03	8497.51 ± 454.72	10827.97 ± 531.20
Param-Noise	131868.30 ± 19237.44	193430.53 ± 19047.81	143549.40 ± 6581.36	87885.43 ± 1970.97	19307.56 ± 165.96
Param-Noise2	168839.33 ± 21919.88	211118.05 ± 18205.53	146716.95 ± 5375.25	89052.58 ± 1974.69	19144.09 ± 141.81
Uniform	588169.99 ± 479.83	400928.29 ± 546.20	151106.74 ± 293.57	79029.43 ± 210.70	19152.74 ± 118.83

Table 10: Simple regret incurred on the Wheel Bandit problem with increasing values of δ. Simple regret was approximated by averaging the regret over the final 500 steps. Values reported are the mean over 50 independent trials with standard error of the mean.

	$\delta = 0.5$	$\delta = 0.7$	$\delta = 0.9$	$\delta = 0.95$	$\delta = 0.99$
Simple regret					
Alpha Divergences	36.63 ± 0.13	24.90 ± 0.19	8.76 ± 0.16	4.67 ± 0.10	0.99 ± 0.05
Alpha Divergences (1)	36.35 ± 0.11	24.67 ± 0.15	9.06 ± 0.15	4.56 ± 0.12	1.02 ± 0.04
Alpha Divergences (2)	36.20 ± 0.23	24.92 ± 0.15	8.95 ± 0.15	4.60 ± 0.12	1.01 ± 0.04
Alpha Divergences (3)	2.38 ± 0.95	2.12 ± 0.59	3.03 ± 0.33	2.44 ± 0.15	0.84 ± 0.04
BBBN	1.87 ± 1.06	3.03 ± 1.11	5.11 ± 0.55	4.08 ± 0.17	0.91 ± 0.04
BBBN2	2.47 ± 0.95	3.51 ± 1.17	6.17 ± 0.53	3.97 ± 0.20	0.91 ± 0.04
BBBN3	0.83 ± 0.74	3.02 ± 0.98	5.63 ± 0.55	4.03 ± 0.17	0.94 ± 0.05
BBBN4	2.27 ± 0.93	4.17 ± 1.23	5.61 ± 0.57	3.86 ± 0.19	0.97 ± 0.05
Bootstrapped NN	1.69 ± 0.80	6.70 ± 1.31	6.73 ± 0.42	4.36 ± 0.13	0.92 ± 0.05
Bootstrapped NN2	2.97 ± 1.13	8.18 ± 1.32	6.87 ± 0.40	4.21 ± 0.17	0.92 ± 0.04
Bootstrapped NN3	1.26 ± 0.75	4.78 ± 1.07	6.61 ± 0.44	4.02 ± 0.16	0.94 ± 0.04
Dropout (RMS3)	12.96 ± 1.38	11.64 ± 1.08	7.35 ± 0.29	4.42 ± 0.15	0.97 ± 0.05
Dropout (RMS2)	12.75 ± 1.41	11.95 ± 0.95	7.62 ± 0.29	4.39 ± 0.14	0.96 ± 0.04
GP	0.84 ± 0.18	0.80 ± 0.22	2.57 ± 0.28	2.55 ± 0.13	0.87 ± 0.04
Neural Linear	$\mathbf{0.08 \pm 0.01}$	$\mathbf{0.14 \pm 0.02}$	$\mathbf{0.15 \pm 0.01}$	$\mathbf{0.17 \pm 0.01}$	0.31 ± 0.03
Neural Linear (2)	$\mathbf{0.09 \pm 0.01}$	$\mathbf{0.13 \pm 0.01}$	$\mathbf{0.16 \pm 0.01}$	0.21 ± 0.03	0.41 ± 0.03
Neural Linear (3)	$\mathbf{0.09 \pm 0.01}$	$\mathbf{0.14 \pm 0.01}$	$\mathbf{0.16 \pm 0.01}$	$\mathbf{0.17 \pm 0.01}$	$\mathbf{0.26 \pm 0.02}$
Neural Linear (4)	$\mathbf{0.09 \pm 0.01}$	$\mathbf{0.13 \pm 0.01}$	$\mathbf{0.16 \pm 0.01}$	$\mathbf{0.17 \pm 0.01}$	$\mathbf{0.27 \pm 0.02}$
RMS1	6.92 ± 1.75	5.59 ± 1.23	5.62 ± 0.56	3.57 ± 0.25	1.00 ± 0.05
RMS2	5.69 ± 1.19	8.71 ± 1.37	6.77 ± 0.39	4.10 ± 0.13	0.91 ± 0.04
RMS2b	5.47 ± 1.24	7.42 ± 1.06	6.22 ± 0.43	4.01 ± 0.17	0.96 ± 0.04
RMS3	6.80 ± 1.38	8.61 ± 1.29	6.60 ± 0.36	4.22 ± 0.17	0.94 ± 0.04
SGFS	1.35 ± 0.28	1.41 ± 0.27	0.99 ± 0.13	0.69 ± 0.06	0.31 ± 0.02
Sparse GP					
ConstSGD	6.04 ± 1.53	6.01 ± 1.17	6.49 ± 0.47	4.40 ± 0.12	0.98 ± 0.04
EpsGreedy (RMS1)	0.80 ± 0.57	1.54 ± 0.67	2.49 ± 0.47	2.43 ± 0.26	0.92 ± 0.05
EpsGreedy (RMS2)	0.75 ± 0.52	1.76 ± 0.66	4.88 ± 0.40	3.49 ± 0.19	0.96 ± 0.05
EpsGreedy (RMS3)	0.47 ± 0.20	4.04 ± 1.03	4.79 ± 0.41	3.16 ± 0.21	0.93 ± 0.05
LinDiagPost	0.15 ± 0.01	0.21 ± 0.02	0.25 ± 0.02	0.21 ± 0.02	$\mathbf{0.24 \pm 0.03}$
LinDiagPrecPost	0.12 ± 0.01	0.17 ± 0.02	0.27 ± 0.02	0.20 ± 0.01	$\mathbf{0.26 \pm 0.03}$
LinGreedy	19.71 ± 1.49	14.43 ± 0.90	8.07 ± 0.27	4.05 ± 0.14	0.94 ± 0.04
LinGreedy (eps=0.01)	0.70 ± 0.06	0.50 ± 0.04	1.20 ± 0.17	1.77 ± 0.18	0.79 ± 0.04
LinGreedy (eps=0.05)	1.64 ± 0.06	1.20 ± 0.05	0.63 ± 0.03	0.49 ± 0.04	0.51 ± 0.03
LinPost	0.12 ± 0.01	0.19 ± 0.02	0.22 ± 0.02	0.21 ± 0.02	$\mathbf{0.29 \pm 0.03}$
LinfullDiagPost	0.33 ± 0.02	0.31 ± 0.02	0.23 ± 0.02	0.22 ± 0.02	0.35 ± 0.03
LinfullDiagPrecPost	0.21 ± 0.02	0.18 ± 0.02	0.26 ± 0.02	$\mathbf{0.18 \pm 0.02}$	0.40 ± 0.03
LinfullPost	0.19 ± 0.02	0.22 ± 0.02	0.26 ± 0.02	$\mathbf{0.17 \pm 0.01}$	0.40 ± 0.04
Param-Noise	5.55 ± 0.92	9.33 ± 0.98	6.86 ± 0.37	4.26 ± 0.14	0.99 ± 0.05
Param-Noise2	7.67 ± 1.10	10.00 ± 0.94	7.30 ± 0.31	4.33 ± 0.13	0.91 ± 0.04
Uniform	29.33 ± 0.16	19.81 ± 0.15	7.62 ± 0.09	3.84 ± 0.06	0.95 ± 0.04



(a) Gaussian Process (b) Sparse Gaussian Process (c) RMSProp

(d) ConstantSGD (e) SGFS (f) BBB

(g) Alpha Divergence (h) NeuralLinear (i) Bootstrap

(j) Dropout (k) Parameter Noise

Figure 5: We qualitatively compare plots of the sample distribution from various methods, similarly to Hernández-Lobato et al. (2016). We plot the mean and standard deviation of 100 samples drawn from each method conditioned on a small set of observations with three outputs (two are from the same underlying function and thus strongly correlated while the third (bottom) is independent). The true underlying functions are plotted in red.

A REAL-WORLD DATASETS

Mushroom. The Mushroom Dataset (Schlimmer, 1981) contains 22 attributes per mushroom, and two classes: poisonous and safe. As in Blundell et al. (2015), we create a bandit problem where the agent must decide whether to eat or not a given mushroom. Eating a safe mushroom provides reward +5. Eating a poisonous mushroom delivers reward +5 with probability 1/2 and reward -35 otherwise. If the agent does not eat a mushroom, then the reward is 0. We set $n = 50000$.

Statlog. The Shuttle Statlog Dataset (Asuncion & Newman, 2007) provides the value of $d = 9$ indicators during a space shuttle flight, and the goal is to predict the state of the radiator subsystem of the shuttle. There are $k = 7$ possible states, and if the agent selects the right state, then reward 1 is generated. Otherwise, the agent obtains no reward ($r = 0$). The most interesting aspect of the dataset is that one action is the optimal one in 80% of the cases, and some algorithms may commit to this action instead of further exploring. In this case, $n = 43500$.

Covertype. The Covertype Dataset (Asuncion & Newman, 2007) classifies the cover type of northern Colorado forest areas in $k = 7$ classes, based on $d = 54$ features, including elevation, slope, aspect, and soil type. Again, the agent obtains reward 1 if the correct class is selected, and 0 otherwise. We run the bandit for $n = 150000$.

Financial. We created the Financial Dataset by pulling the stock prices of $d = 21$ publicly traded companies in NYSE and Nasdaq, for the last 14 years $(n = 3713)$. For each day, the context was the price difference between the beginning and end of the session for each stock. We synthetically created the arms, to be a linear combination of the contexts, representing $k = 8$ different potential portfolios. By far, this was the smallest dataset, and many algorithms over-explored at the beginning with no time to amortize their investment (Thompson Sampling does not account for the horizon).

Jester. We create a recommendation system bandit problem as follows. The Jester Dataset (Goldberg et al., 2001) provides continuous ratings in $[-10, 10]$ for 100 jokes from 73421 users. We find a complete subset of $n = 19181$ users rating all 40 jokes. Following Riquelme et al. (2017), we take $d = 32$ of the ratings as the context of the user, and $k = 8$ as the arms. The agent recommends one joke, and obtains the reward corresponding to the rating of the user for the selected joke.

Adult. The Adult Dataset (Kohavi, 1996; Asuncion & Newman, 2007) comprises personal information from the US Census Bureau database, and the standard prediction task is to determine if a person makes over $50K a year or not. However, we consider the $k = 14$ different occupations as feasible actions, based on $d = 94$ covariates (many of them binarized). As in previous datasets, the agent obtains reward 1 for making the right prediction, and 0 otherwise. We set $n = 45222$.

Census. The US Census (1990) Dataset (Asuncion & Newman, 2007) contains a number of personal features (age, native language, education...) which we summarize in $d = 389$ covariates, including binary dummy variables for categorical features. Our goal again is to predict the occupation of the individual among $k = 9$ classes. The agent obtains reward 1 for making the right prediction, and 0 otherwise, for each of the $n = 250000$ randomly selected data points.

Song. The YearPredictionMSD Dataset is a subset of the Million Song Dataset (Bertin-Mahieux et al., 2011). The goal is to predict the year a given song was released (1922-2011) based on $d = 90$ technical audio features. We divided the years in $k = 10$ contiguous year buckets containing the same number of songs, and provided decreasing Gaussian rewards as a function of the distance between the interval chosen by the agent and the one containing the year the song was actually released. We initially selected $n = 250000$ songs at random from the training set.

The Statlog, Covertype, Adult, and Census datasets were tested in Elmachtoub et al. (2017).