DEEP BAYESIAN BANDITS SHOWDOWN

AN EMPIRICAL COMPARISON OF BAYESIAN DEEP NETWORKS FOR THOMPSON SAMPLING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent interest in decision making with deep neural networks has led to a wide development of practical methods that trade off exploration and exploitation. Bayesian approaches to deep learning are especially appealing for this purpose as they can provide accurate uncertainty estimates as input for reinforcement learning algorithms. However, these methods are rarely compared on benchmarks that evaluate the impact of their approximations in terms of decision-making performance, and their empirical effectiveness seems poorly understood. In this paper, we compare a variety of well-established and recent methods under the lens of Thompson Sampling over a series of contextual bandit problems.

1 INTRODUCTION

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

Unfortunately, computing the exact Bayesian posterior is generally intractable for such highly parameterized 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., 2013; Ahn et al., 2012; Mandt et al., 2016). 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 methods for exploration using deep neural networks. We compare a variety of well-established and recent methods under the lens of Thompson sampling for contextual bandits, a classical task in sequential 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 epistemic uncertainty. Furthermore, exploratory actions should be coordinated throughout the entire decision making process, known as deep exploration, rather than performed independently at each state.

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

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 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.

As expected, no single algorithm beats all the others in every bandit problem. We observe, however, some general trends. We find that simple heuristics, including bootstrapping, dropout, and injecting random noise, do not significantly and systematically improve performance, while they may still sometimes help. 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.

In Section 2 we discuss Thompson Sampling, and present the contextual bandit problem. The different algorithmic principles to 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.

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 could 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? There are two main possibilities. In some 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?

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).

3 ALGORITHMS

In this section, we describe the main algorithmic design principles behind the specific implementations used to approximately sample from the posterior in our simulations of Section 5. In Figure 24, we visualize the posteriors of the nonlinear algorithms on a synthetic one dimensional problem.

Linear Methods We can 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 lead to the right posterior *only* when the data was generated according to $Y = X^T\beta + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. 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, leading, in general, 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 $a_0 = b_0 = \alpha > 1$.

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 observed 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 $\text{KL}(\mathcal{N}(\mu_t, \hat{\Sigma}) \mid\mid \mathcal{N}(\mu_t, \Sigma_t))$, 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 $\text{KL}(\mathcal{N}(\mu_t, \Sigma_t) \mid\mid \mathcal{N}(\mu_t, \bar{\Sigma}))$ instead, similarly to expectation-propagation algorithms. 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. Somewhat surprisingly, this did not help.

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). Typically (and in our experiments), the posterior is approximated by a mean-field or factorized distribution. 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.

Dropout (Srivastava et al., 2014) can be seen as optimizing a variational objective (Kingma et al., 2015; Gal & Ghahramani, 2016), which leads to a straight-forward posterior approximation and application to Thompson sampling.

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. (2013) show that a preconditioner based on the RMSprop algorithm performs well on deep neural networks. Patterson & Teh (2013) suggesed using the Fisher information matrix as a preconditioner in SGLD. Unfortunately theory requires that 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 constantSGD 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 and 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) or variational inference (Titsias, 2009). However, these are outside the scope of this comparison. [1] 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 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:

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

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

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

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.

There are a variety of other methods, including expectation propagation (Hernández-Lobato & Adams, 2015) and α-divergence minimization of which variational inference is a special case (Hernández-Lobato et al., 2016). We leave evaluating these approaches to future work.

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: $\tilde{\pi} \approx \pi = \mathbf{P}(\theta \mid \mathbf{D})$. Let the distance $d(\tilde{\pi}, \pi)$ represent the quality of our approximation.

[1] We invite others to apply these and new methods upon the open source release of this benchmark.

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 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 red) 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.

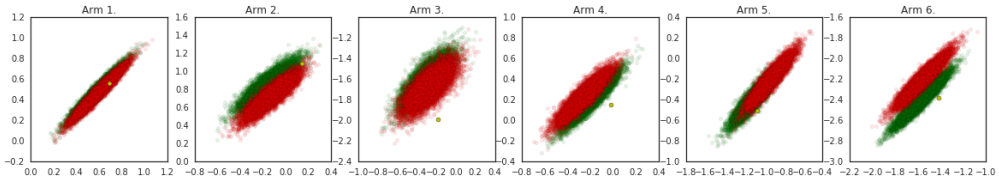
5 Empirical Evaluation

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.

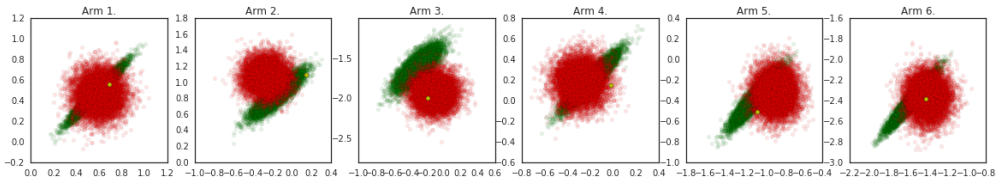
5.1 The Experimental Framework

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.

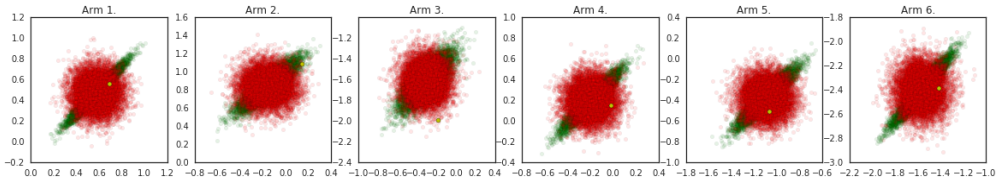
Neural Network Architectures. All algorithms based on neural networks as function approximators share the same architecture. In particular, we fit a simple fully-connected feedforward network



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

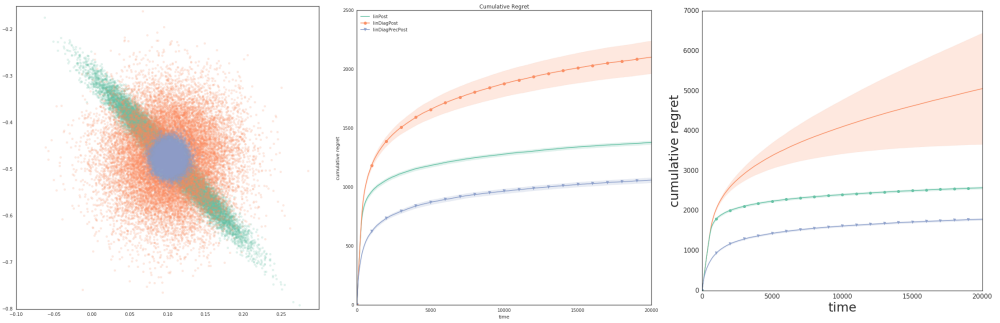


(b) Thompson Sampling with the true linear posterior (green), and the diagonalized version (red).



(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.



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

Figure 2: In The impact on regret of different approximated posteriors. We show (green) the actual linear posterior, (red) 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.

with two hidden layers with 100 units each and softplus activations.[2] 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.

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 $t_s = 20$ mini-batches every $t_f = 20$ timesteps [3]. The size of each mini-batch was 512. We experimented with increasing values of t_s, and it proved essential for some algorithms like variational inference approaches.

[2]We also tried relu activations, which led to similar results.

[3]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; Meire Fortunato, 2017).

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)).

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; for some algorithms they were chosen through careful tuning over cumulative regret on a small subset of linear bandits (Dropout, SGFS, RMS, BBBN, pNoise) and some (ConstSGD, Neural Linear and all linear methods) required very little tuning.

Buffer After some experimentation, we decided not to use a data buffer as some evidence of catastrophic forgetting was observed. 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 test the algorithms in a broad range of bandit problems created using real-world data. In particular, we test on the Mushroom, Statlog, Covertype, Financial, Jester, Adult, Census, and Song datasets. A detailed description of each dataset and its bandit problem is provided in the Appendix, Section A. 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 datasets are directly presented in the Appendix. The Statlog, Covertype, Adult, and Census datasets were originally tested in Elmachtoub et al. (2017). The results are displayed in Table 1 (cumulative regret) and Table 5 (simple regret).

5.3 Real-World Data Problems with Linear Models

As the algorithms from Section 3 can be implemented for any model architecture, here we use linear models as a baseline comparison across algorithms. The datasets are the same as in the previous subsection. The results are displayed in Table 2 (cumulative regret) and Table 5 (simple regret).

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 26). Set $d = 2$, and $\delta \in (0, 1)$, the exploration parameter. Contexts are sampled uniformly at random in the unit circle, $X \sim U(\mathbf{D})$. 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. For contexts inside the blue circle, i.e. $\|X\| \leq \delta$, 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 one of the actions $a_1, \ldots, a_4$ is optimal depending on the sign of X_1 and 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 $r \sim \mathcal{N}(\mu_2, \sigma^2)$, 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 falling in the high-reward region is $1 - \delta^2$.

The difficulty of the problem increases with δ, and we expect algorithms to get stuck repeatedly selecting action a_1. The problem can be easily generalized for $d > 2$. Results are shown in Table 6, and Figures 3, 4, 5, 6, and 7.

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 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 consistently beat the RMS neural network it is based on, suggesting its exploration mechanisms add concrete value. We believe its main strength relies on the fact 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. 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.

Variational Inference. Overall, BBB performed poorly, even when the model was linear. Variational methods are known to underestimate uncertainty (Bishop, 2006), so we expected that might lead to poor exploration, however, this does not explain the poor performance entirely. When the model is linear, LinDiagPrecPost computes the exact solution to the minimization problem BBB optimizes with stochastic variational inference. The poor performance of BBB relative to LinDiagPrecPost suggests that the problem is more than underexploration. Figure 25 shows that performance improves as we increase the number of training steps, suggesting that it is not sufficient to partially optimize the variational parameters when the uncertainty estimates directly affect the data being collected. Optimizing to convergence at every step significantly reduces regret, but at major computational cost.

Monte Carlo. ConstantSGD 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 doesn't 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 improvement, if any, provided by bootstrapping neural networks over the base model is surprisingly small. Moreover, it might not justify the important computational overhead of the method. The randomness from SGD may be enough for exploration in many bandits problems.

Direct Noise Injection. Parameter-Noise was based on the RMS2 implementation. Results suggest that it may only offer modest improvements over its base learner, while we found the algorithm hard to tune, and quite sensitive to the heuristic controlling the injected noise-level. In addition, developing an intuition for the heuristic is not straightforward as it lacks transparency, and may require previous and repeated access to the decision-making process.

On the other hand, 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. 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. The methods we found to perform best 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

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

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.

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.

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.

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

Granmo, Ole-Christoffer. Solving two-armed 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.

Hernández-Lobato, José Miguel and Adams, Ryan P. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, 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.

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, Ke, Swersly, Kevin, Ryan, and Zemel, Richard S. Efficient feature learning using perturb-and-map. *NIPS Workshop on Perturbations, Optimization, and Statistics*, 2013.

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

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

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.

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.

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*.

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.

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

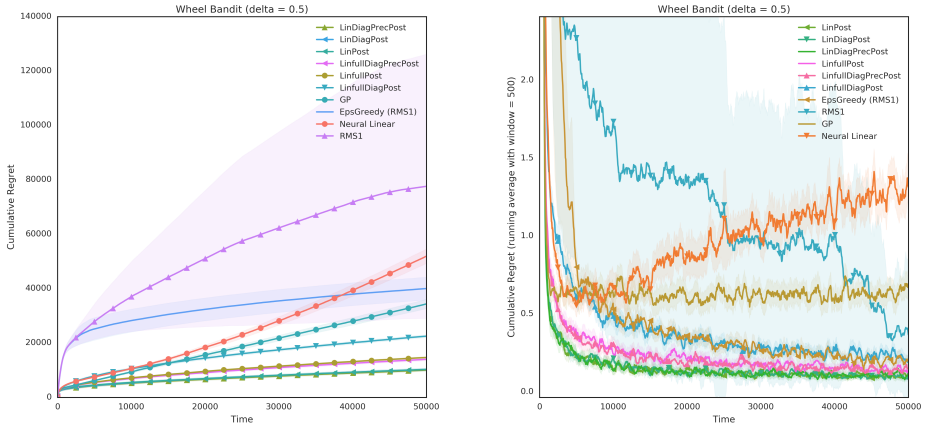


Figure 3: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Wheel Bandit (delta = 0.5) dataset.

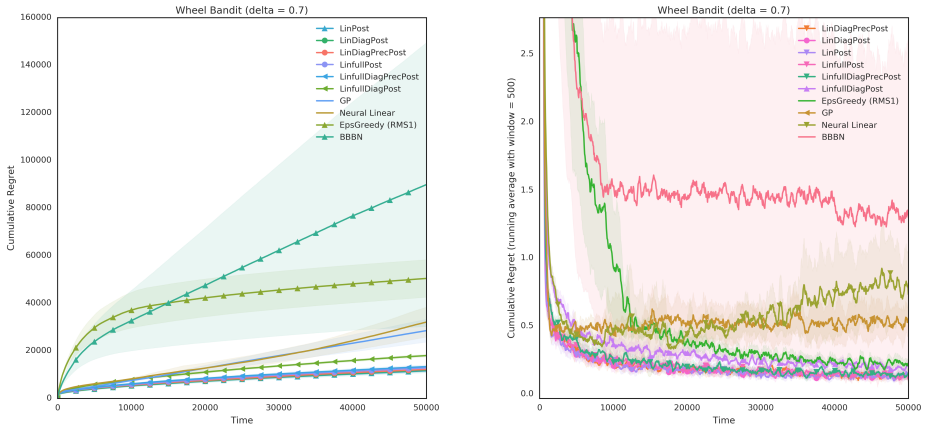


Figure 4: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Wheel Bandit (delta = 0.7) dataset.

APPENDIX

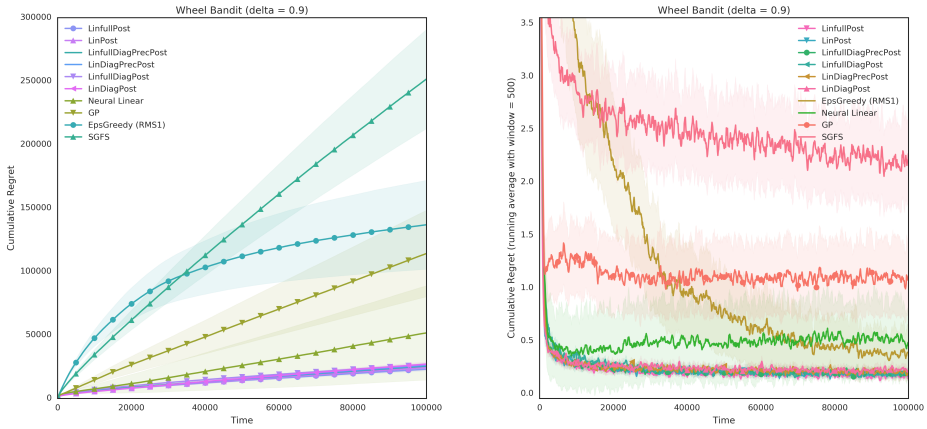


Figure 5: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Wheel Bandit (delta = 0.9) dataset.

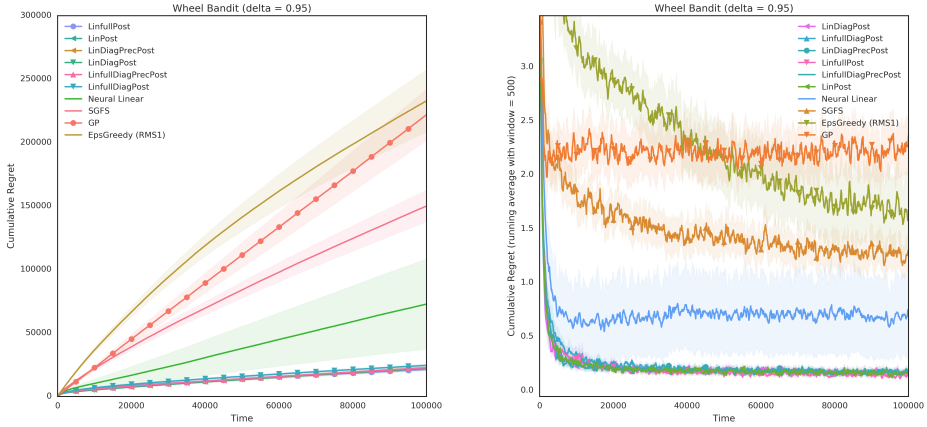


Figure 6: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Wheel Bandit (delta = 0.95) dataset.

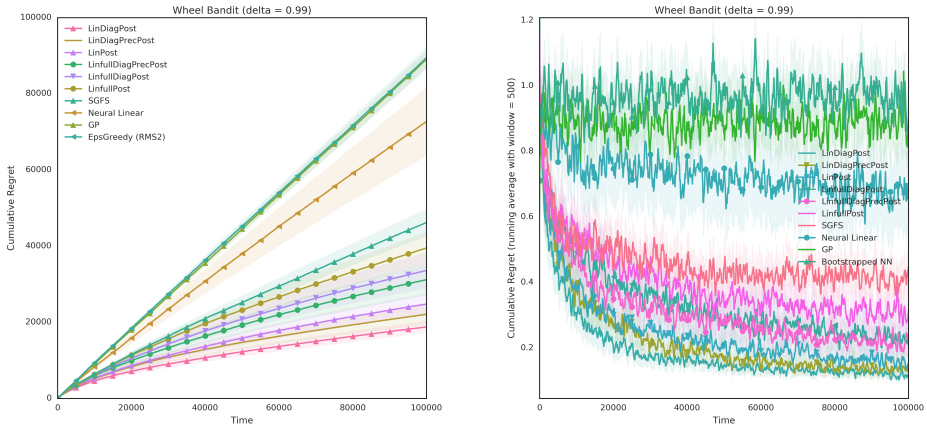


Figure 7: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Wheel Bandit (delta = 0.99) dataset.

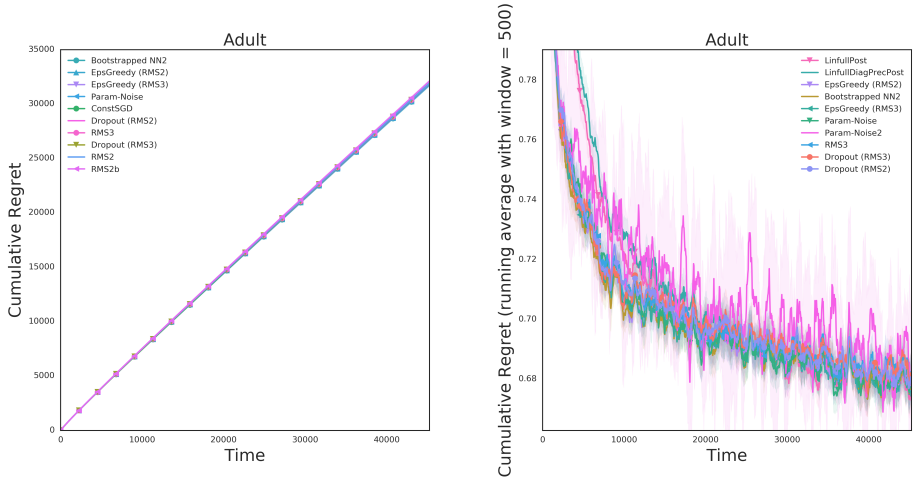


Figure 8: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Adult dataset where all methods use linear models.

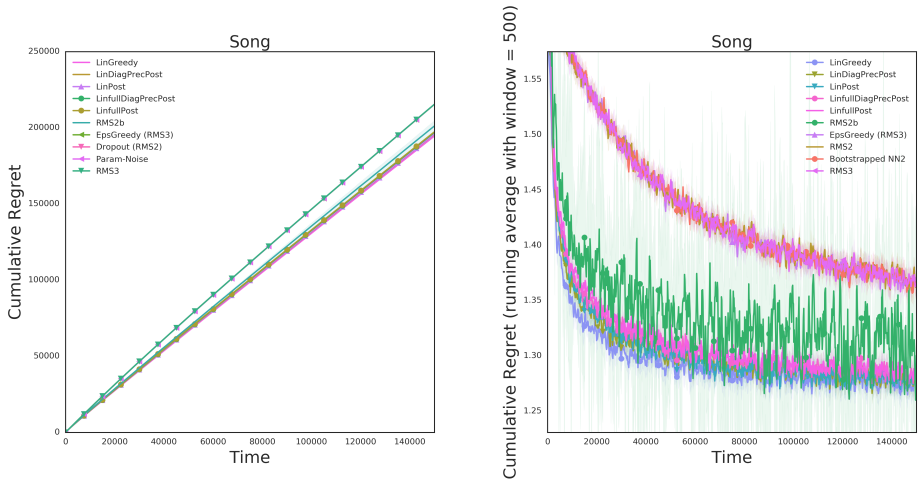


Figure 9: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Song dataset where all methods use linear models.

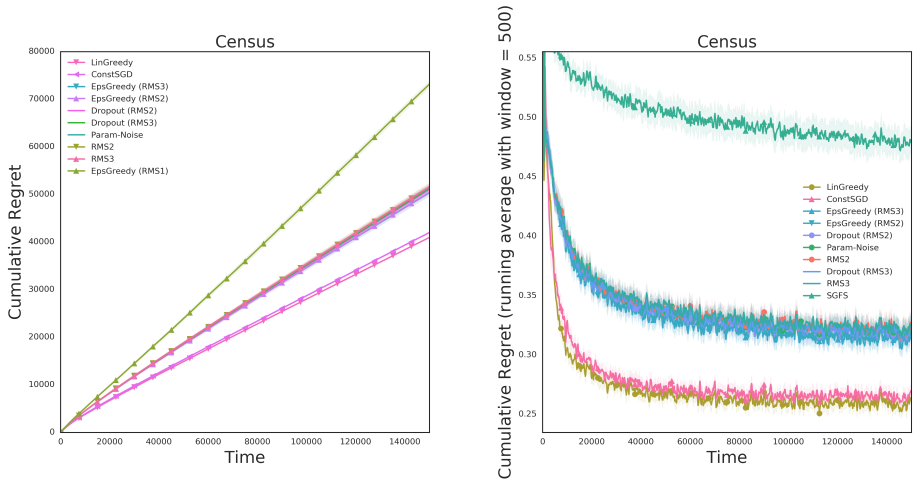


Figure 10: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Census dataset where all methods use linear models.

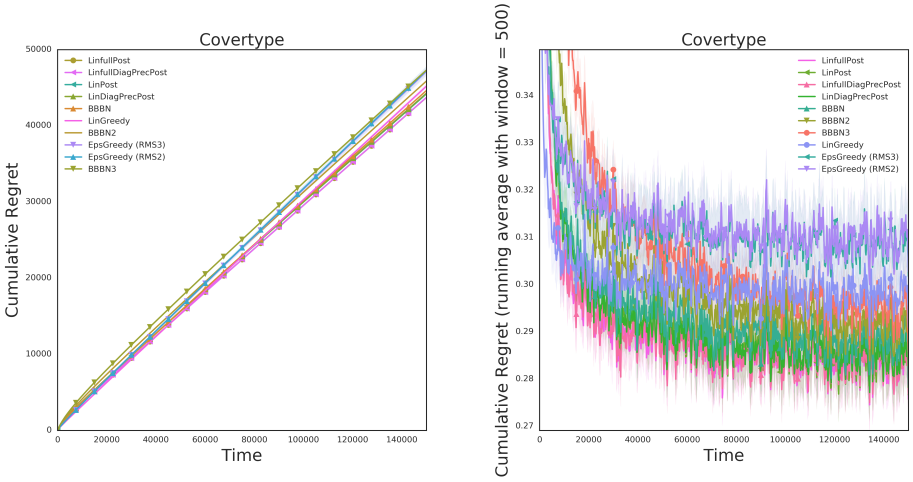


Figure 11: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Covertype dataset where all methods use linear models.

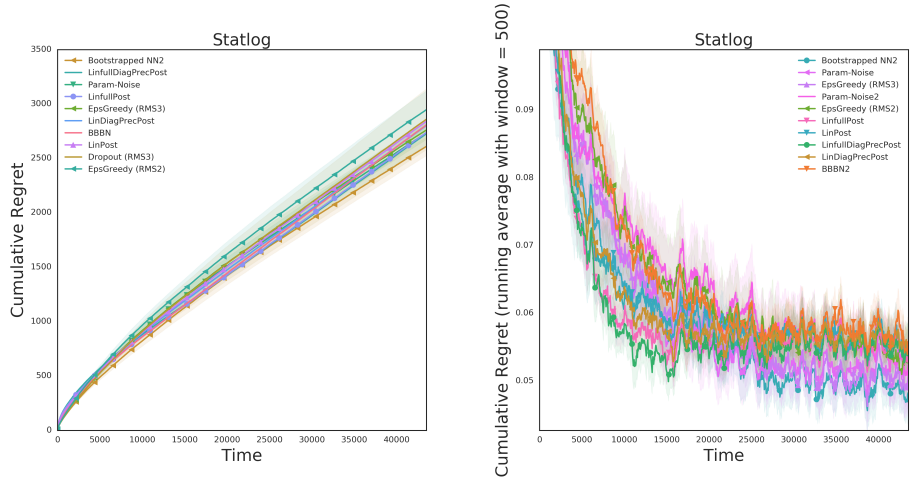


Figure 12: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Statlog dataset where all methods use linear models.

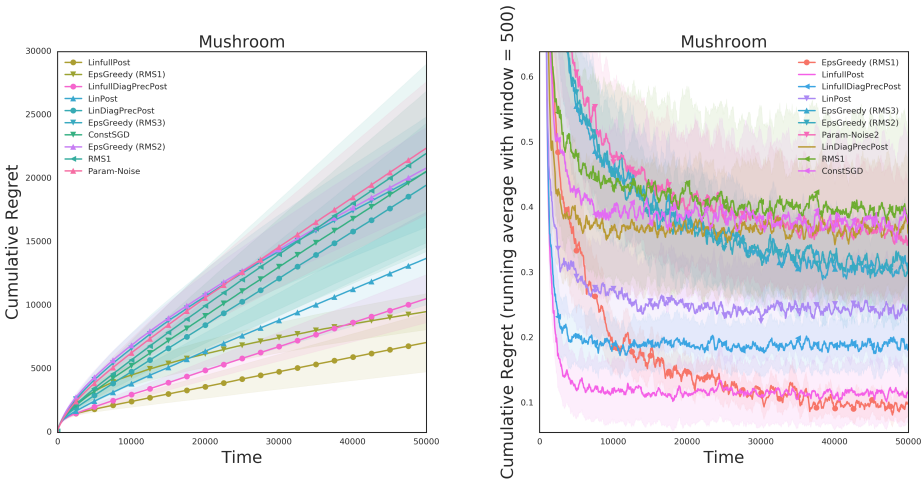


Figure 13: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Mushroom dataset where all methods use linear models.

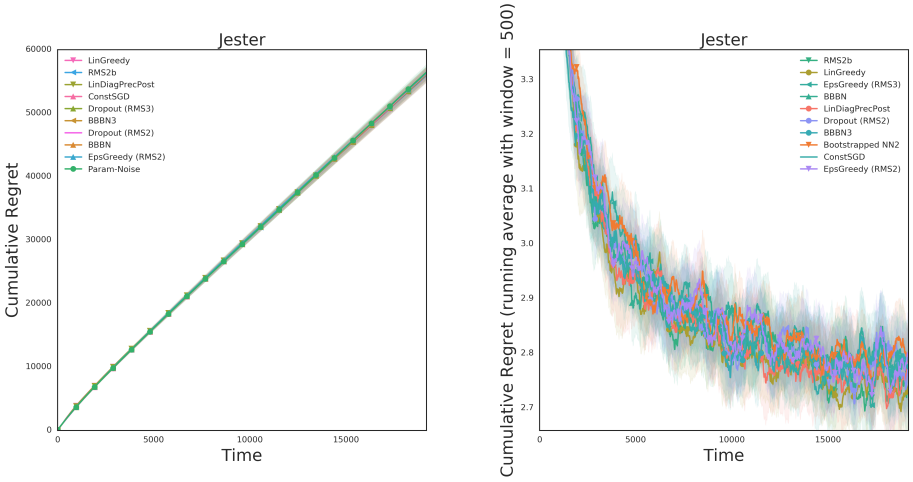


Figure 14: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Jester dataset where all methods use linear models.

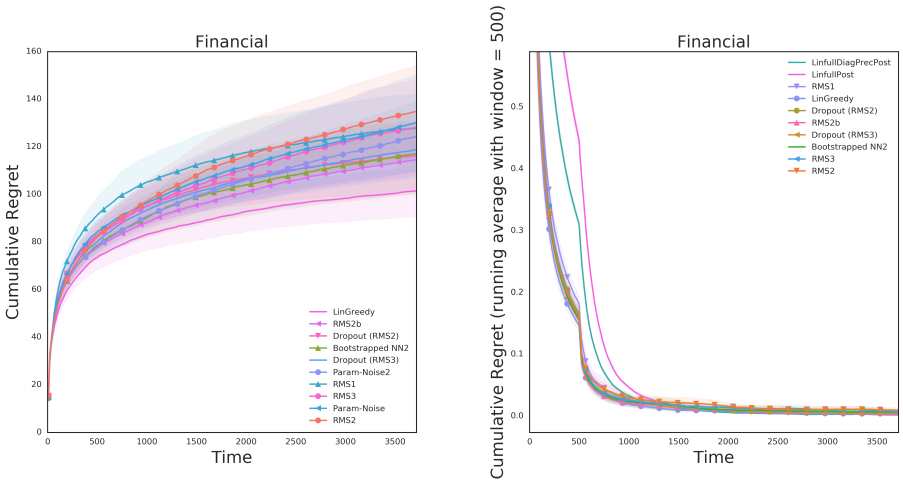


Figure 15: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Financial dataset where all methods use linear models.

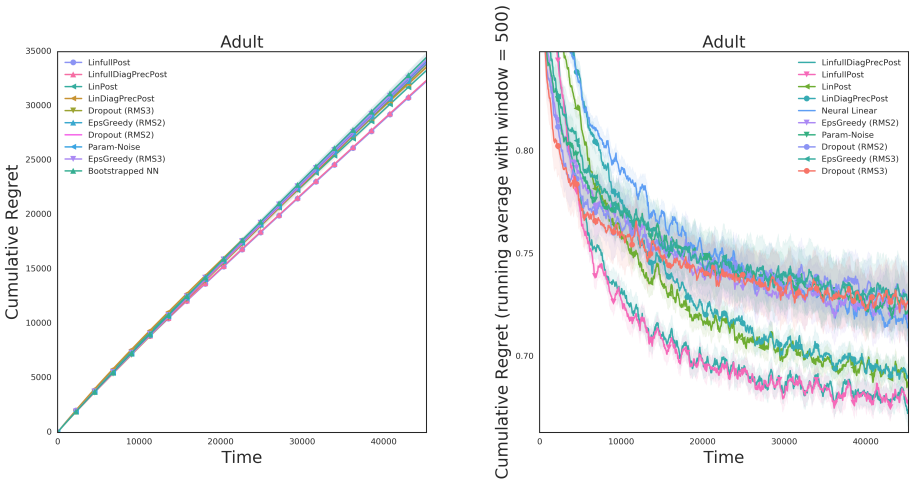


Figure 16: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Adult dataset.

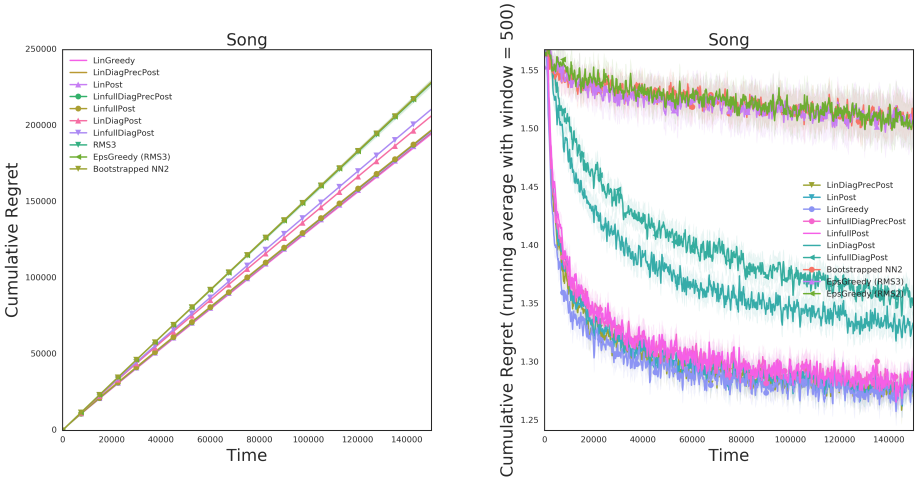


Figure 17: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Song dataset.

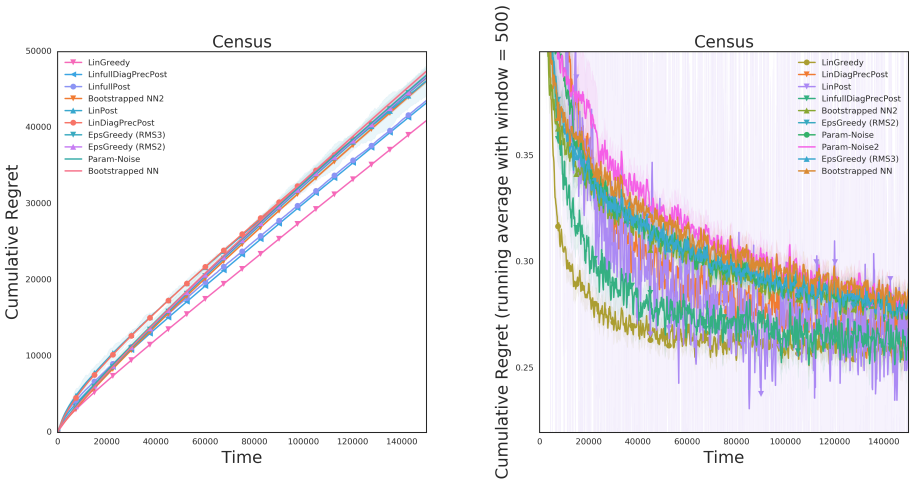


Figure 18: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Census dataset.

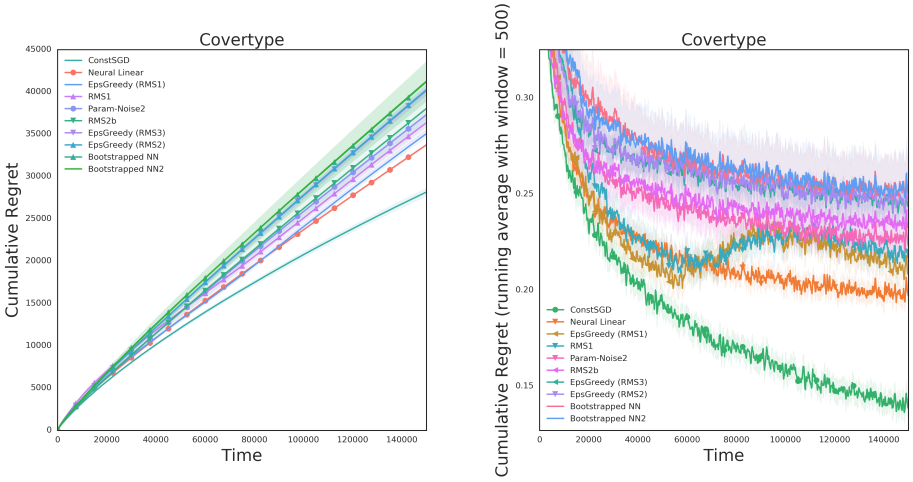


Figure 19: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Covertype dataset.

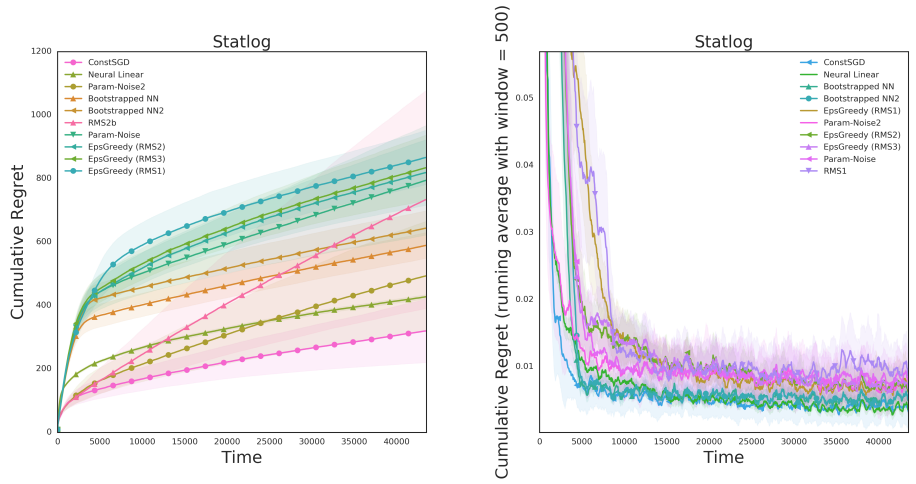


Figure 20: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Statlog dataset.

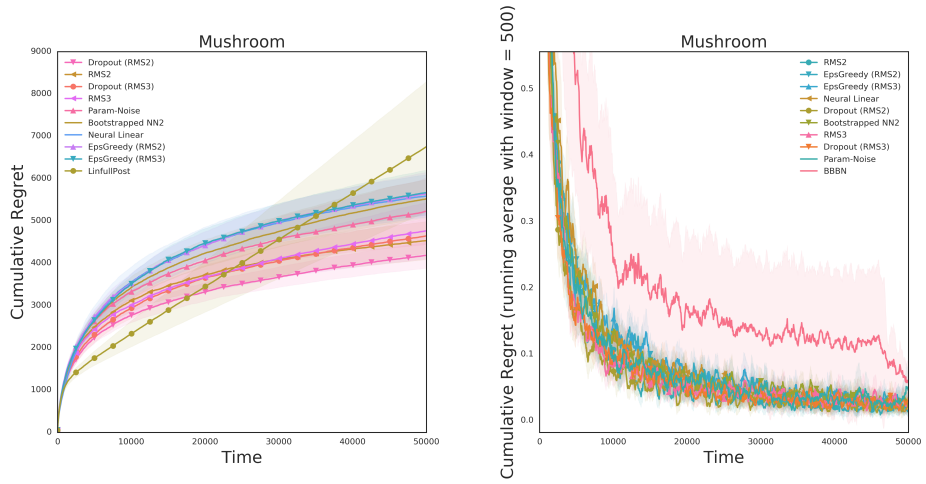


Figure 21: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Mushroom dataset.

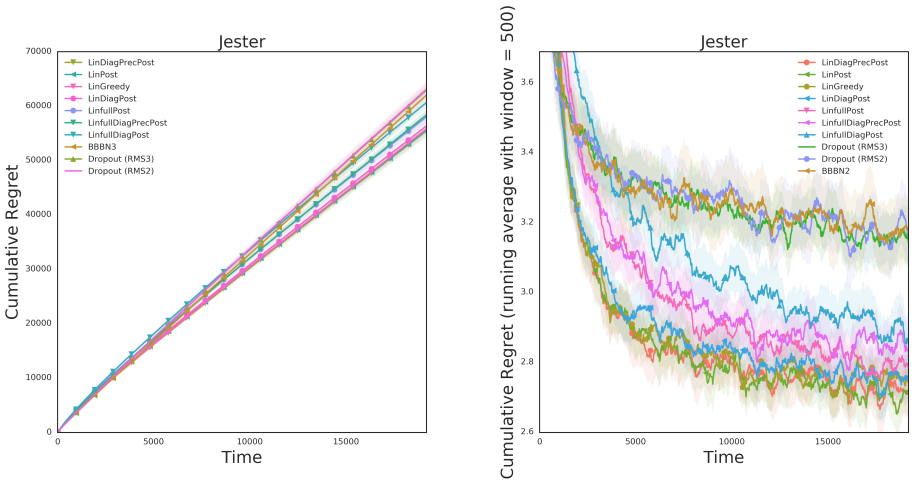


Figure 22: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Jester dataset.

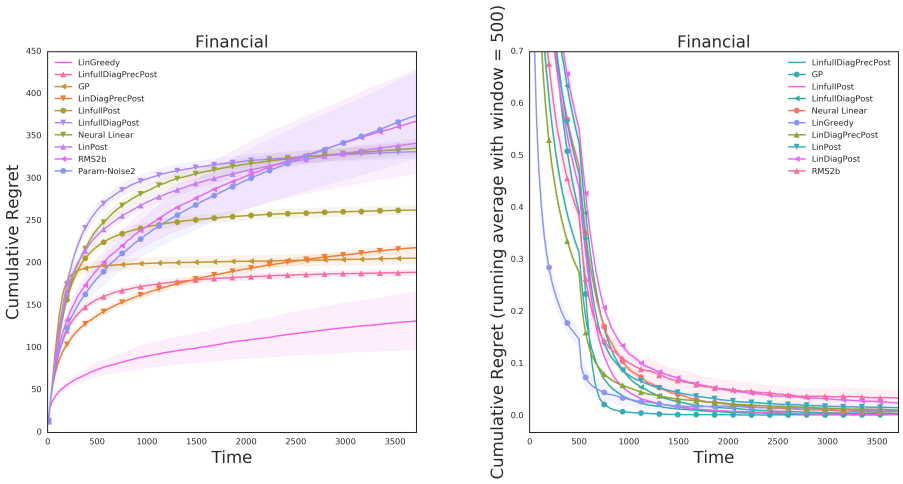
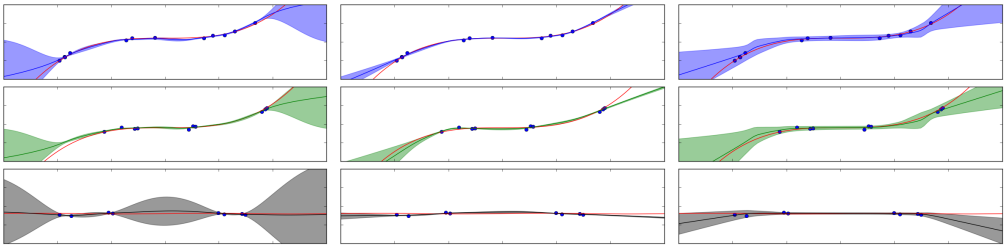
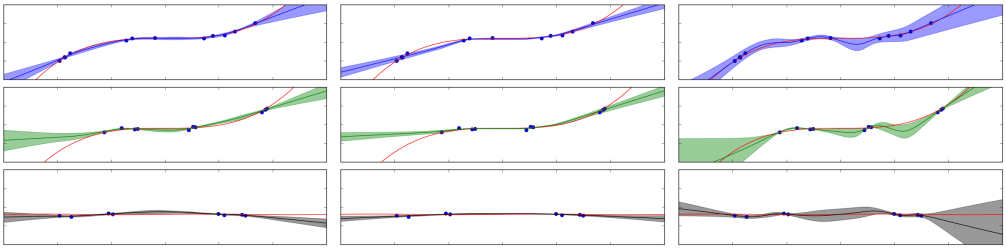


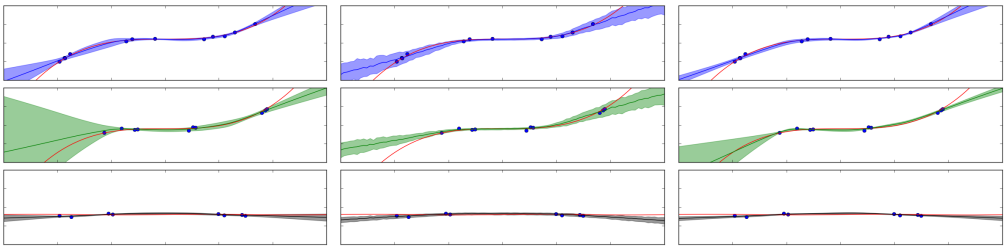
Figure 23: Cumulative regret and average regret (over a moving window of 500 timesteps) for the Financial dataset.



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



(d) SGFS (e) BBB (f) NeuralLinear



(g) Bootstrap (h) Dropout (i) Parameter Noise

Figure 24: 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.

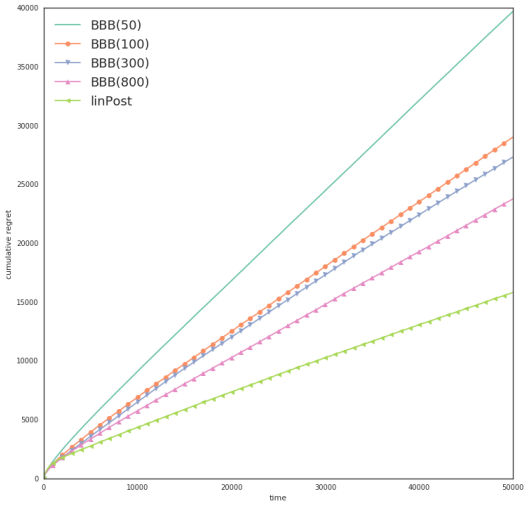
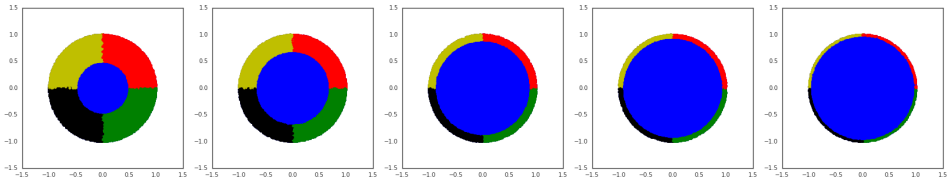


Figure 25: Performance of BBB algorithms with linear models in Mushroom Dataset for increasing number of training steps every 20 time-steps (in parenthesis next to the name).



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

Figure 26: Chrome bandits for increasing values of $\delta \in (0, 1)$. Optimal action for yellow, red, black, green, and blue regions, are actions 1, 2, 3, 4, and 5, respectively.

Table 1: Cumulative regret incurred by the algorithms in Section 3 on the bandits described in Section A. All methods use linear models. Results are relative to the cumulative regret of the Uniform algorithm.

	Mushroom	Statlog	Covertype	Financial	Jester
BBBN	9.60 ± 1.96	7.52 ± 0.02	34.70 ± 0.02	4.96 ± 0.07	58.98 ± 0.47
BBBN2	19.80 ± 3.27	8.19 ± 0.03	35.67 ± 0.02	7.53 ± 0.05	59.17 ± 0.45
BBBN3	25.79 ± 3.49	8.93 ± 0.03	36.79 ± 0.02	10.49 ± 0.09	58.89 ± 0.55
Bootstrapped NN					
Bootstrapped NN2	10.91 ± 1.22	7.00 ± 0.11	37.81 ± 0.26	2.47 ± 0.18	59.30 ± 0.45
Dropout (RMS3)	12.16 ± 1.35	7.67 ± 0.37	37.25 ± 0.26	2.51 ± 0.12	58.74 ± 0.59
Dropout (RMS2)	13.07 ± 1.60	8.11 ± 0.47	37.45 ± 0.27	2.45 ± 0.16	58.91 ± 0.47
GP					
Neural Linear					
RMS1	8.94 ± 1.43	8.64 ± 0.08	37.87 ± 0.13	2.69 ± 0.15	60.95 ± 0.53
RMS2	11.24 ± 1.41	8.35 ± 0.52	37.35 ± 0.26	2.84 ± 0.20	59.68 ± 0.52
RMS2b	11.90 ± 1.32	10.14 ± 0.69	38.83 ± 0.27	2.42 ± 0.15	58.58 ± 0.53
RMS3	12.28 ± 1.77	8.13 ± 0.47	37.22 ± 0.22	2.70 ± 0.21	59.79 ± 0.71
SGFS	20.01 ± 1.73	16.07 ± 0.52	50.97 ± 0.41	7.96 ± 0.53	74.50 ± 0.57
Oracle	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
ConstSGD	8.37 ± 1.28	10.01 ± 0.58	36.87 ± 0.26	3.53 ± 0.18	58.72 ± 0.55
EpsGreedy (RMS1)	3.87 ± 0.29	8.94 ± 0.05	37.47 ± 0.09	3.43 ± 0.12	60.50 ± 0.48
EpsGreedy (RMS2)	8.47 ± 0.65	7.90 ± 0.23	36.74 ± 0.21	3.20 ± 0.14	59.03 ± 0.55
EpsGreedy (RMS3)	8.35 ± 0.88	7.40 ± 0.12	36.64 ± 0.22	3.17 ± 0.10	59.22 ± 0.58
LinDiagPost					
LinDiagPrecPost	7.92 ± 0.99	7.51 ± 0.02	34.42 ± 0.03	4.65 ± 0.04	58.71 ± 0.65
LinGreedy	12.86 ± 1.61	11.93 ± 0.64	35.15 ± 0.11	2.14 ± 0.11	58.54 ± 0.55
LinPost	5.58 ± 0.74	7.61 ± 0.02	34.36 ± 0.02	7.30 ± 0.06	60.09 ± 0.48
LinfullDiagPost					
LinfullDiagPrecPost	4.28 ± 0.38	7.31 ± 0.02	34.00 ± 0.03	3.94 ± 0.04	61.80 ± 0.52
LinfullPost	2.88 ± 0.47	7.34 ± 0.02	33.99 ± 0.02	5.61 ± 0.06	61.58 ± 0.48
Param-Noise	9.11 ± 1.06	7.33 ± 0.15	37.15 ± 0.23	2.74 ± 0.21	59.15 ± 0.63
Param-Noise2	9.28 ± 1.03	7.91 ± 0.21		2.62 ± 0.15	60.75 ± 0.58
Uniform	100.00 ± 0.14	100.00 ± 0.02	100.00 ± 0.02	100.00 ± 1.55	100.00 ± 1.09

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.

	Mushroom	Statlog	Covertype	Financial	Jester
BBBN	9.60 ± 1.96	7.52 ± 0.02	34.70 ± 0.02	4.96 ± 0.07	58.98 ± 0.47
BBBN2	19.80 ± 3.27	8.19 ± 0.03	35.67 ± 0.02	7.53 ± 0.05	59.17 ± 0.45
BBBN3	25.79 ± 3.49	8.93 ± 0.03	36.79 ± 0.02	10.49 ± 0.09	58.89 ± 0.55
Bootstrapped NN					
Bootstrapped NN2	10.91 ± 1.22	7.00 ± 0.11	37.81 ± 0.26	2.47 ± 0.18	59.30 ± 0.45
Dropout (RMS3)	12.16 ± 1.35	7.67 ± 0.37	37.25 ± 0.26	2.51 ± 0.12	58.74 ± 0.59
Dropout (RMS2)	13.07 ± 1.60	8.11 ± 0.47	37.45 ± 0.27	2.45 ± 0.16	58.91 ± 0.47
GP					
Neural Linear					
RMS1	8.94 ± 1.43	8.64 ± 0.08	37.87 ± 0.13	2.69 ± 0.15	60.95 ± 0.53
RMS2	11.24 ± 1.41	8.35 ± 0.52	37.35 ± 0.26	2.84 ± 0.20	59.68 ± 0.52
RMS2b	11.90 ± 1.32	10.14 ± 0.69	38.83 ± 0.27	2.42 ± 0.15	58.58 ± 0.53
RMS3	12.28 ± 1.77	8.13 ± 0.47	37.22 ± 0.22	2.70 ± 0.21	59.79 ± 0.71
SGFS	20.01 ± 1.73	16.07 ± 0.52	50.97 ± 0.41	7.96 ± 0.53	74.50 ± 0.57
Oracle	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
ConstSGD	8.37 ± 1.28	10.01 ± 0.58	36.87 ± 0.26	3.53 ± 0.18	58.72 ± 0.55
EpsGreedy (RMS1)	3.87 ± 0.29	8.94 ± 0.05	37.47 ± 0.09	3.43 ± 0.12	60.50 ± 0.48
EpsGreedy (RMS2)	8.47 ± 0.65	7.90 ± 0.23	36.74 ± 0.21	3.20 ± 0.14	59.03 ± 0.55
EpsGreedy (RMS3)	8.35 ± 0.88	7.40 ± 0.12	36.64 ± 0.22	3.17 ± 0.10	59.22 ± 0.58
LinDiagPost					
LinDiagPrecPost	7.92 ± 0.99	7.51 ± 0.02	34.42 ± 0.03	4.65 ± 0.04	58.71 ± 0.65
LinGreedy	12.86 ± 1.61	11.93 ± 0.64	35.15 ± 0.11	2.14 ± 0.11	58.54 ± 0.55
LinPost	5.58 ± 0.74	7.61 ± 0.02	34.36 ± 0.02	7.30 ± 0.06	60.09 ± 0.48
LinfullDiagPost					
LinfullDiagPrecPost	4.28 ± 0.38	7.31 ± 0.02	34.00 ± 0.03	3.94 ± 0.04	61.80 ± 0.52
LinfullPost	2.88 ± 0.47	7.34 ± 0.02	33.99 ± 0.02	5.61 ± 0.06	61.58 ± 0.48
Param-Noise	9.11 ± 1.06	7.33 ± 0.15	37.15 ± 0.23	2.74 ± 0.21	59.15 ± 0.63
Param-Noise2	9.28 ± 1.03	7.91 ± 0.21		2.62 ± 0.15	60.75 ± 0.58
Uniform	100.00 ± 0.14	100.00 ± 0.02	100.00 ± 0.02	100.00 ± 1.55	100.00 ± 1.09

Table 3: Cumulative regret incurred by the algorithms in Section 3 on the bandits described in Section A. All methods use linear models. Results are relative to the cumulative regret of the Uniform algorithm.

	Adult	Song	Census
BBBN	94.12 ± 0.23	95.55 ± 0.17	67.64 ± 0.59
BBBN2	94.12 ± 0.23	95.40 ± 0.16	67.40 ± 0.54
BBBN3	94.20 ± 0.26	95.47 ± 0.18	66.82 ± 0.08
Bootstrapped NN	82.08 ± 0.54	94.43 ± 0.22	35.57 ± 0.10
Bootstrapped NN2	82.60 ± 0.83	94.26 ± 0.26	34.57 ± 0.09
Dropout (RMS3)	80.54 ± 0.69	95.84 ± 0.28	39.86 ± 1.86
Dropout (RMS2)	81.00 ± 0.64	95.01 ± 0.30	36.97 ± 1.11
GP	92.06 ± 0.32	97.20 ± 0.17	45.40 ± 0.29
Neural Linear	82.26 ± 0.10	96.51 ± 0.47	59.77 ± 2.88
RMS1	87.68 ± 0.40	95.43 ± 0.20	40.44 ± 0.47
RMS2	82.35 ± 0.85	95.02 ± 0.34	37.74 ± 1.81
RMS2b	83.71 ± 0.66	94.98 ± 0.44	38.60 ± 1.96
RMS3	82.92 ± 0.92	94.01 ± 0.47	38.35 ± 2.04
SGFS	95.33 ± 0.31	96.58 ± 0.18	50.07 ± 0.17
Oracle	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
ConstSGD	88.76 ± 0.34	95.10 ± 0.10	40.01 ± 0.37
EpsGreedy (RMS1)	87.35 ± 0.27	95.49 ± 0.08	39.60 ± 0.25
EpsGreedy (RMS2)	80.77 ± 0.63	94.36 ± 0.32	35.04 ± 0.19
EpsGreedy (RMS3)	81.47 ± 0.73	94.13 ± 0.44	34.95 ± 0.18
LinDiagPost	99.67 ± 0.02	85.11 ± 0.02	99.44 ± 0.01
LinDiagPrecPost	79.93 ± 0.04	80.64 ± 0.02	34.91 ± 0.05
LinGreedy	83.29 ± 0.68	80.26 ± 0.05	30.71 ± 0.02
LinPost	79.20 ± 0.03	80.80 ± 0.02	34.61 ± 0.12
LinfullDiagPost	96.18 ± 0.03	86.92 ± 0.02	97.12 ± 0.02
LinfullDiagPrecPost	77.05 ± 0.03	81.19 ± 0.02	32.45 ± 0.02
LinfullPost	76.84 ± 0.03	81.28 ± 0.02	$32.69 \pm nan$
Param-Noise	81.17 ± 0.57	94.98 ± 0.37	35.18 ± 0.38
Param-Noise2	83.21 ± 0.57	95.56 ± 0.28	36.20 ± 0.34
Uniform	100.00 ± 0.02	100.00 ± 0.01	100.00 ± 0.01

Table 4: 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.

	Adult	Song	Census
BBBN	94.12 ± 0.23	95.55 ± 0.17	67.64 ± 0.59
BBBN2	94.12 ± 0.23	95.40 ± 0.16	67.40 ± 0.54
BBBN3	94.20 ± 0.26	95.47 ± 0.18	66.82 ± 0.08
Bootstrapped NN	82.08 ± 0.54	94.43 ± 0.22	35.57 ± 0.10
Bootstrapped NN2	82.60 ± 0.83	94.26 ± 0.26	34.57 ± 0.09
Dropout (RMS3)	80.54 ± 0.69	95.84 ± 0.28	39.86 ± 1.86
Dropout (RMS2)	81.00 ± 0.64	95.01 ± 0.30	36.97 ± 1.11
GP	92.06 ± 0.32	97.20 ± 0.17	45.40 ± 0.29
Neural Linear	82.26 ± 0.10	96.51 ± 0.47	59.77 ± 2.88
RMS1	87.68 ± 0.40	95.43 ± 0.20	40.44 ± 0.47
RMS2	82.35 ± 0.85	95.02 ± 0.34	37.74 ± 1.81
RMS2b	83.71 ± 0.66	94.98 ± 0.44	38.60 ± 1.96
RMS3	82.92 ± 0.92	94.01 ± 0.47	38.35 ± 2.04
SGFS	95.33 ± 0.31	96.58 ± 0.18	50.07 ± 0.17
Oracle	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
ConstSGD	88.76 ± 0.34	95.10 ± 0.10	40.01 ± 0.37
EpsGreedy (RMS1)	87.35 ± 0.27	95.49 ± 0.08	39.60 ± 0.25
EpsGreedy (RMS2)	80.77 ± 0.63	94.36 ± 0.32	35.04 ± 0.19
EpsGreedy (RMS3)	81.47 ± 0.73	94.13 ± 0.44	34.95 ± 0.18
LinDiagPost	99.67 ± 0.02	85.11 ± 0.02	99.44 ± 0.01
LinDiagPrecPost	79.93 ± 0.04	80.64 ± 0.02	34.91 ± 0.05
LinGreedy	83.29 ± 0.68	80.26 ± 0.05	30.71 ± 0.02
LinPost	79.20 ± 0.03	80.80 ± 0.02	34.61 ± 0.12
LinfullDiagPost	96.18 ± 0.03	86.92 ± 0.02	97.12 ± 0.02
LinfullDiagPrecPost	77.05 ± 0.03	81.19 ± 0.02	32.45 ± 0.02
LinfullPost	76.84 ± 0.03	81.28 ± 0.02	$32.69 \pm nan$
Param-Noise	81.17 ± 0.57	94.98 ± 0.37	35.18 ± 0.38
Param-Noise2	83.21 ± 0.57	95.56 ± 0.28	36.20 ± 0.34
Uniform	100.00 ± 0.02	100.00 ± 0.01	100.00 ± 0.01

Table 5: Simple regret incurred by the algorithms in Section 3 on the bandits described in Section A.

	Mushroom	Statlog	Covertype	Financial	Jester	Adult	Song	Census

Table 6: Cumulative regret for the Wheel Bandit with increasing values of δ.

	$\delta = 0.5$	$\delta = 0.7$	$\delta = 0.9$	$\delta = 0.95$	$\delta = 0.99$
BBBN	11.65 ± 4.44	8.96 ± 2.96	69.91 ± 7.61	84.01 ± 6.56	103.27 ± 0.63
BBBN2	8.85 ± 4.27	13.49 ± 4.89	78.22 ± 7.71	90.77 ± 6.08	102.29 ± 0.85
BBBN3	5.74 ± 2.73	17.14 ± 5.38	59.62 ± 8.10	104.87 ± 4.95	101.81 ± 1.05
Bootstrapped NN	40.41 ± 4.08	66.97 ± 3.94	94.75 ± 3.10	105.75 ± 2.54	103.53 ± 0.69
Bootstrapped NN2	45.65 ± 4.20	71.45 ± 3.23	89.37 ± 3.56	103.60 ± 2.56	100.86 ± 1.05
Dropout (RMS3)	50.91 ± 4.98	73.49 ± 4.09	87.99 ± 4.51	107.75 ± 2.88	102.53 ± 0.67
Dropout (RMS2)	60.32 ± 3.93	64.44 ± 4.33	98.55 ± 3.75	110.42 ± 2.63	102.59 ± 0.63
GP	2.33 ± 0.06	2.84 ± 0.24	15.07 ± 2.24	56.03 ± 2.55	95.27 ± 1.18
Neural Linear	3.53 ± 0.09	3.20 ± 0.31	6.81 ± 2.44	18.36 ± 4.47	77.80 ± 4.71
RMS1	5.27 ± 1.64	14.84 ± 3.83	63.75 ± 6.19	103.18 ± 2.92	103.25 ± 0.55
RMS2	52.83 ± 3.90	79.47 ± 4.06	89.93 ± 3.44	109.57 ± 2.61	103.84 ± 0.46
RMS2b	43.98 ± 4.97	61.20 ± 4.69	100.58 ± 3.32	109.03 ± 1.90	102.96 ± 0.61
RMS3	50.95 ± 3.93	67.98 ± 3.46	97.76 ± 3.38	105.98 ± 3.03	101.71 ± 0.97
SGFS	13.33 ± 1.93	19.15 ± 2.75	33.23 ± 2.57	37.86 ± 1.57	49.45 ± 1.84
Oracle	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
ConstSGD	23.49 ± 3.80	30.13 ± 5.61	64.76 ± 6.56	90.09 ± 5.90	103.13 ± 0.38
EpsGreedy (RMS1)	2.72 ± 0.14	5.03 ± 0.39	18.07 ± 2.30	58.76 ± 3.08	96.51 ± 1.11
EpsGreedy (RMS2)	14.64 ± 1.91	47.17 ± 3.28	72.71 ± 1.67	85.18 ± 1.94	95.79 ± 1.47
EpsGreedy (RMS3)	14.14 ± 1.56	53.38 ± 2.93	70.58 ± 1.41	84.06 ± 1.64	98.22 ± 1.39
LinDiagPost	0.70 ± 0.02	1.18 ± 0.04	3.57 ± 0.12	5.63 ± 0.16	20.07 ± 0.78
LinDiagPrecPost	0.68 ± 0.02	1.21 ± 0.05	3.44 ± 0.11	5.60 ± 0.21	23.64 ± 1.38
LinGreedy	57.14 ± 4.88	73.25 ± 3.96	104.57 ± 3.67	112.27 ± 1.87	100.47 ± 1.22
LinPost	0.70 ± 0.02	1.15 ± 0.04	3.20 ± 0.11	5.53 ± 0.15	26.51 ± 1.46
LinfullDiagPost	1.53 ± 0.02	1.80 ± 0.03	3.53 ± 0.07	6.18 ± 0.14	35.96 ± 2.46
LinfullDiagPrecPost	0.95 ± 0.02	1.33 ± 0.04	3.35 ± 0.08	5.71 ± 0.16	33.38 ± 2.15
LinfullPost	1.00 ± 0.01	1.28 ± 0.03	3.03 ± 0.08	5.35 ± 0.18	42.26 ± 2.18
Param-Noise	49.38 ± 5.39	63.81 ± 3.27	90.70 ± 3.71	106.05 ± 2.56	102.86 ± 0.93
Param-Noise2	41.14 ± 5.11	69.27 ± 4.92	96.17 ± 3.69	108.82 ± 2.16	101.99 ± 0.78
Uniform	100.00 ± 0.06	100.00 ± 0.08	100.00 ± 0.11	100.00 ± 0.13	100.00 ± 0.27

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 (asu) 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 (asu) 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; asu) 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 (asu) 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).