Batched Neural Bandits

QUANQUAN GU, University of California, Los Angeles, USA
AMIN KARBASI, Yale University, USA
KHASHAYAR KHOSRAVI, Google Research NYC, USA
VAHAB MIRROKNI, Google Research NYC, USA
DONGRUO ZHOU, University of California, Los Angeles, USA

In many sequential decision-making problems, the individuals are split into several batches and the decision-maker is only allowed to change her policy at the end of batches. These batch problems have a large number of applications, ranging from clinical trials to crowdsourcing. Motivated by this, we study the stochastic contextual bandit problem for general reward distributions under the batched setting. We propose the BatchNeuralUCB algorithm which combines neural networks with optimism to address the exploration-exploitation tradeoff while keeping the total number of batches limited. We study BatchNeuralUCB under both fixed and adaptive batch size settings and prove that it achieves the same regret as the fully sequential version while reducing the number of policy updates considerably. We confirm our theoretical results via simulations on both synthetic and real-world datasets.

1 INTRODUCTION

In the stochastic contextual bandit problem, a learner sequentially picks actions over T rounds (the *horizon*). At each round, the learner observes K actions, each associated with a d-dimensional feature vector. After selecting an action, she receives stochastic reward. Her goal is to maximize the cumulative reward attained over the horizon. Very often, the learner also receives some form of side information, so called context, about the arms. Contextual bandits problems have been extensively studied in the literature [7, 25, 26] and have a vast number of applications such as personalized news recommendation [28] and healthcare (see Bouneffouf and Rish [6] and references therein).

There has been a large body of work, aiming at designing efficient policies with low regrets when contexts enjoy extra structures such as linear models [1–3, 12, 13, 28], generalized linear models [18, 29], and kernel-based models [36, 37]. Recently, neural network models that allow for a more powerful approximation of the underlying reward functions have been proposed [32, 38, 40, 41]. Notably, the NeuralUCB algorithm [41] can achieve near-optimal regret bounds while only requiring a mild boundedness assumption on the rewards. However, a major shortcoming of NeuralUCB is that it requires updating the neural network parameters in every round, as well as optimizing the loss function over observed rewards and contexts. Due to the large number of parameters of the neural networks, NeuralUCB is considerably slow for large-scale applications, especially for the case where the online decision needs to be made quickly [9].

A widely used approach to alleviate such computational burden is to update the model parameters *in batches*. Batch learning has been studied in many applications that require limited adaptivity. Examples include multi-stage clinical trials [31], crowdsourcing platforms [23, 24], active learning [11, 17], and running time-consuming simulations for reinforcement learning [27]. Specifically, for simpler models, such as multi-armed bandits and linear contextual bandits, recent works have studied *batched* algorithms for both settings [16, 19, 20, 22, 31, 33] where the parameters are updated at pre-fixed time periods, and *rarely switching* setting [8, 10, 14, 33, 34] where the parameters are updated adaptively and can depend on the previous contexts and reward observations. While these papers provide a complete characterization of the optimal number of policy switches in stochastic contextual bandit with linear rewards, the extension of results to more general reward functions remains unstudied.

In this paper, we propose a BatchNeuralUCB (BNUCB) algorithm that uses neural networks for estimating rewards while keeping the total number of policy updates to be small . BatchNeuralUCB addresses both limitations described above: (1) it reduces the computational complexity of NeuralUCB, allowing its usage in large-scale applications, and (2) it limits the number of policy updates, making it a natural choice for settings that require limited adaptivity. Figure 1 shows the effectiveness of our proposed BNUCB algorithm in achieving these two points on the Mushroom dataset from the UCI repository. [1] It is worth noting that while the idea of limiting the number of updates for neural networks has been used in Riquelme et al. [32], Xu et al. [38] and the experiments of Zhou et al. [41], no formal results on the number of required batches nor the optimal batch selection scheme have been shown. Our paper takes the first step and provides a rigorous treatment of these empirical results. Specifically, we study how the batches should be designed to guarantee good regret performance, similar to those depicted in Figure 1. Our main contributions are as follows:

- We propose BatchNeuralUCB which, in sharp contrast to NeuralUCB, only updates its network parameters at most B times, where B is the number of batches. We propose two update schemes: the *fixed* batch scheme where the batch grid is pre-fixed, and the *adaptive* batch scheme where the selection of batch grid can depend on previous contexts and observed rewards. When $B = T$, BatchNeuralUCB degenerates to NeuralUCB.
- We prove that for BatchNeuralUCB with a fixed batch scheme, the regret is bounded by $\widetilde{O}(\widetilde{d}\sqrt{T} + \widetilde{d}T/B)$, where $\widetilde{d}$ is the effective dimension (See Definition 5.6). For adaptive batch scheme, for any choice of q, the regret is bounded by $\widetilde{O}(\sqrt{\max\{q, (1+TK)^{\widetilde{d}}/q^B\}}\widetilde{d}\sqrt{T})$, where q is the parameter that determines the adaptivity of our algorithm (See Algorithm 1 for details), and K is the number of arms. Therefore, to obtain the same regret as in the fully sequential counterpart, BatchNeuralUCB only requires $\Omega(\sqrt{T})$ for the fixed and $\Omega(\log T)$ for adaptive batch schemes. These bounds match the lower bounds presented in the batched linear bandits [20] and rarely switching linear bandits [33], respectively.
- We carry out numerical experiments over synthetic and real datasets to confirm our theoretical findings. These experiments demonstrate that in most configurations with fixed and adaptive schemes, the regret of the proposed BatchNeuralUCB remains close to the regret of the fully sequential benchmarks, while the number of policy updates and the running time are reduced significantly.

Technical challenges and innovations To derive our batch-dependent neural bandit regret bound, we first utilize the recent breakthrough in the Neural Tangent Kernel (NTK) theory [21] which approximates the overparameterized neural network with a high dimensional linear function with random feature mappings. Later on, we utilize results from batched linear bandits [20, 33]. The main challenge is that directly applying batched linear bandit results on NTK would yield a neural network width-dependent regret bound, which is unacceptable in the overparameterized setting. To address this issue, we provide a more refined analysis and derive a regret bound that *only* depends on the effective dimension.

Notations We use lower case letters to denote scalars, lower and upper case bold letters to denote vectors and matrices. We use $\|\cdot\|$ to indicate Euclidean norm, and for a semi-positive definite matrix Σ and any vector $\mathbf{x}$, $\|\mathbf{x}\|_\Sigma := \|\Sigma^{1/2}\mathbf{x}\| = \sqrt{\mathbf{x}^\top\Sigma\mathbf{x}}$. We also use the standard O and Ω notations. We say $a_n = O(b_n)$ if and only if $\exists C > 0, N > 0, \forall n > N, a_n \le Cb_n$; $a_n = \Omega(b_n)$ if $a_n \ge Cb_n$. The notation $\widetilde{O}$ is used to hide logarithmic factors. Finally, we use the shorthand that $[n]$ to denote the set of integers $\{1, ..., n\}$.

[1]This dataset can be found at https://archive.ics.uci.edu/ml/datasets/mushroom

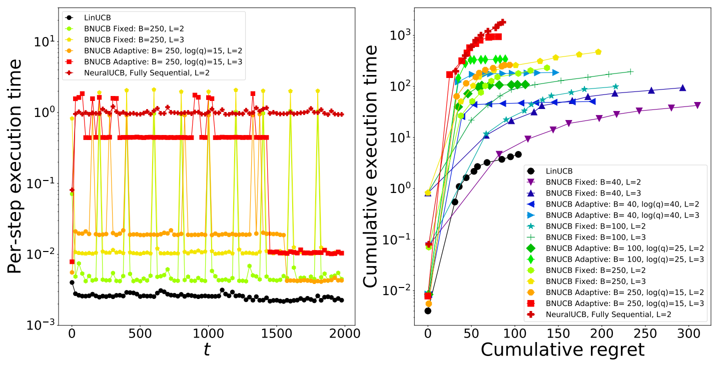


Fig. 1. Comparison of regret and execution time of LinUCB [28] and NeuralUCB [41] with the proposed BatchNeuralUCB (BNUCB) algorithm on the Mushroom dataset. Left: per-step execution time of algorithms and, right: cumulative execution time vs cumulative regret. In both figures B is the number of batches, L is the number of layers of the fully connected neural network and q is a tuning parameter of adaptive BNUCB.

2 RELATED WORK

The literature on the contextual multi-armed problem is vast. Due to the space limitations, we only review the existing work on batched bandits and bandits with function approximations here and refer the interested reader to recent monographs by Slivkins et al. [35] and Lattimore and Szepesvári [26] for a thorough overview.

Batched Bandits. The design of batched multi-armed bandit models can be traced back to UCB2 [4] and Improved-UCB [5] algorithms originally for the fully sequential setting. Perchet et al. [31] provided the first systematic analysis of the batched stochastic multi-armed bandit problem and established near-optimal gap-dependent and gap-independent regrets for the case of two arms ($K = 2$). Gao et al. [19] extended this analysis to the general setting of $K > 2$. They proved regret bounds for both adaptive and non-adaptive grids. Esfandiari et al. [16] improved the gap-dependent regret bound for the stochastic case and provided lower and upper bound regret guarantees for the adversarial case. They also establish regret bounds for the batched stochastic linear bandits.

Our work in the batched setting is mostly related to Han et al. [20], Ruan et al. [33]. In particular, Han et al. [20] studied the batched stochastic linear contextual bandit problem for both adversarial and stochastic contexts. For the case of adversarial contexts, they show that the number of batches B should be at least $\Omega(\sqrt{dT})$. Ruan et al. [33] studied the batched contextual bandit problem using distributional optimal designs and extended the result of [20]. They also studied the minimum adaptivity needed for the rarely switching contextual bandit problems in both adversarial and stochastic context settings. In particular, for adversarial contexts, they proved a lower bound of $\Omega((d \log T)/\log(d \log T))$. Our work, however, is different from Han et al. [20], Ruan et al. [33] as we do not require any assumption on the linearity of the reward functions; similar to NeuralUCB [41], our regret analysis only requires the rewards to be bounded.

Bandits with Function Approximation. Given the fact that Deep Neural Network (DNN) models enable the learner to make use of nonlinear models with less domain knowledge, Riquelme et al. [32], Zahavy and Mannor [39] studied *neural-linear bandits*. In particular, they used all but the last layers of a DNN as a feature map, which transforms contexts from the raw input space to a low-dimensional space, usually with better representation and less frequent updates. Then they learned a linear exploration policy on top of the last hidden layer of the DNN with more frequent updates. Even though these attempts have achieved great empirical success, they do not provide any regret guarantees. Zhou et al. [41] proposed NeuralUCB algorithm that uses neural networks to estimate reward functions while addressing the exploration-exploitation tradeoff using the upper confidence bound technique. Zhang et al. [40] extended their analysis to Thompson Sampling. Xu et al. [38] proposed Neural-LinUCB which shares the same spirit as *neural-linear bandits* and proved $\widetilde{O}(\sqrt{T})$ regret bound.

3 PROBLEM SETTING

In this section, we present the technical details of our model and our problem setting.

Model. We consider the stochastic K-armed contextual bandit problem, where the total number of rounds T is known. At round $t \in [T]$, the learner observes the context consisting of K feature vectors: $\{\mathbf{x}_{t,a} \in \mathbb{R}^d \mid a \in [K]\}$. For brevity, we denote the collection of all contexts $\{\mathbf{x}_{1,1}, \mathbf{x}_{1,2}, \dots, \mathbf{x}_{T,K}\}$ by $\{\mathbf{x}^i\}_{i=1}^{TK}$.

Reward. Upon selecting an action a_t, she receives a stochastic reward r_{t,a_t}. In this work, we make the following assumption about reward generation: for any round t,

$$r_{t,a_t} = h(\mathbf{x}_{t,a_t}) + \xi_t, \tag{3.1}$$

where h is an unknown function satisfying $0 \le h(\mathbf{x}) \le 1$ for any $\mathbf{x}$, and ξ_t is ν-sub-Gaussian noise conditioned on $\mathbf{x}_{1,a_1}, \dots, \mathbf{x}_{t-1,a_{t-1}}$ satisfying $\mathbb{E}[\xi_t | \mathbf{x}_{1,a_1}, \dots, \mathbf{x}_{t-1,a_{t-1}}] = 0$.

Goal. The learner wishes to maximize the following *pseudo regret* (or *regret* for short): $R_T = \sum_{t=1}^{T} [h(\mathbf{x}_{t,a_t^*}) - h(\mathbf{x}_{t,a_t})]$, where $a_t^* = \mathrm{argmax}_{a \in [K]}\, h(\mathbf{x}_{t,a})$ is the optimal action at round t that maximizes the expected reward.

Reward Estimation. In order to learn the reward function h in Eq. (3.1), we propose to use a fully connected neural networks with depth $L \ge 2$:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \sqrt{m}\mathbf{W}_L \sigma\Big(\mathbf{W}_{L-1}\sigma\big(\cdots \sigma(\mathbf{W}_1 \mathbf{x})\big)\Big), \tag{3.2}$$

where $\sigma(x) = \max\{x, 0\}$ is the rectified linear unit (ReLU) activation function, $\mathbf{W}_1 \in \mathbb{R}^{m \times d}, \mathbf{W}_i \in \mathbb{R}^{m \times m}, 2 \le i \le L-1, \mathbf{W}_L \in \mathbb{R}^{m \times 1}$, and $\boldsymbol{\theta} = [\mathrm{vec}(\mathbf{W}_1)^\top, \dots, \mathrm{vec}(\mathbf{W}_L)^\top]^\top \in \mathbb{R}^p$ with $p = m + md + m^2(L-1)$. Without loss of generality, we assume that the width of each hidden layer is the same (i.e., m) for convenience in the analysis. We denote the gradient of the neural network function by $\mathbf{g}(\mathbf{x}; \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\mathbf{x}; \boldsymbol{\theta}) \in \mathbb{R}^p$.

Batch Setting. In this work, we consider the *batch bandits* setting in which the entire horizon of T rounds is divided into B batches. Formally, we define a grid $\mathcal{T} = \{t_0, t_1, \cdots, t_B\}$, where $1 = t_0 < t_1 < t_2 < \cdots < t_B = T + 1$ are the start and end rounds of the batches. Here, the interval $[t_{b-1}, t_b)$ indicates the rounds belonging to batch $b \in [B]$. The learner selects her policy at the beginning of each batch and executes it for the entire batch. She observes all collected rewards at the end of this batch and then updates her policy for the next batch. The batch model consists of two specific schemes. In the *fixed batch size scheme*, the points in the grid $\mathcal{T}$ are pre-fixed and cannot be altered during the execution of the algorithm. In the *adaptive batch size scheme*, however, the beginning and the end rounds of each batch are decided dynamically by the algorithm.

4 ALGORITHMS

We propose our algorithm BatchNeuralUCB in this section. In essence, BatchNeuralUCB uses a neural network $f(\mathbf{x}; \boldsymbol{\theta})$ to predict the reward of the context $\mathbf{x}$ and upper confidence bounds computed from the network to guide the exploration [3]. Note that BatchNeuralUCB does not

Algorithm 1 BatchNeuralUCB

```
Require: A neural network $f(\mathbf{x};\boldsymbol{\theta})$ initialized with parameter $\boldsymbol{\theta}_0$, batch number $B$, ratio parameter
    $q$ (only needed for adaptive batch size), regularization parameter $\lambda$, step size $\eta$, number of
    gradient descent steps $J$
 1: $\mathbf{Z}_1 = \lambda\mathbf{I}$, $b = 0$, $t_0 = 1$
 2: for $t = 1, \dots, T$ do
 3:   if
          Fixed Batch Size Scheme:, $t = b \cdot \lfloor T/B \rfloor + 1$                                   (4.1)
          Adaptive Batch Size Scheme:, $\det(\mathbf{Z}_t) > q \cdot \det(\mathbf{Z}_{t_b})$ and $b \le B - 2$,     (4.2)
       then
 4:     $b \leftarrow b + 1$, $t_b \leftarrow t$
 5:     Observe rewards $\{r_{i,a_i}\}_{i=t_{b-1}}^{t_b-1}$ corresponding to contexts $\{\mathbf{x}_{i,a_i}\}_{i=t_{b-1}}^{t_b-1}$
 6:     $\boldsymbol{\theta}_b \leftarrow \text{TrainNN}(\lambda, \eta, J, m, \{\mathbf{x}_{i,a_i}\}_{i=1}^{t_b-1}, \{r_{i,a_i}\}_{i=1}^{t_b-1}, \boldsymbol{\theta}_0)$
 7:     $\widehat{f}_b(\cdot) \leftarrow f(\cdot;\boldsymbol{\theta}_b) + \beta_{t_b}\sqrt{\mathbf{g}(\cdot;\boldsymbol{\theta}_b)^\top \mathbf{Z}_{t_b}^{-1}\mathbf{g}(\cdot;\boldsymbol{\theta}_b)/m}$,
 8:   end if
 9:   Receive $\{\mathbf{x}_{t,a}\}_{a=1}^K$
10:   Select $a_t \leftarrow \text{argmax}_{a\in[K]} \widehat{f}_b(\mathbf{x}_{t,a})$
11:   Set $\mathbf{Z}_{t+1} \leftarrow \mathbf{Z}_t + \mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_b)\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_b)^\top/m$
12: end for
13: $t_{b+1} = T + 1$
```

Algorithm 2 TrainNN [41]

```
Require: Regularization parameter $\lambda$, step size $\eta$, number of gradient descent steps $J$, network
    width $m$, actions $\{\mathbf{x}_t\}$, rewards $\{r_t\}$, initial parameter $\boldsymbol{\theta}^{(0)}$.
 1: Define $\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^t (f(\mathbf{x}_i;\boldsymbol{\theta}) - r_i)^2/2 + m\lambda\|\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}\|_2^2/2$.
 2: for $j = 0, \dots, J-1$ do
 3:   $\boldsymbol{\theta}^{(j+1)} \leftarrow \boldsymbol{\theta}^{(j)} - \eta\nabla\mathcal{L}(\boldsymbol{\theta}^{(j)})$
 4: end for
Ensure: $\boldsymbol{\theta}^{(J)}$.
```

update its parameter $\boldsymbol{\theta}$ at each round. Instead, BatchNeuralUCB specifies either a fixed or adaptive batch grid $\mathcal{T} = \{t_0, t_1, \dots, t_B\}$. At the beginning of the b-th batch, the algorithm updates the parameter $\boldsymbol{\theta}$ of the neural network to $\boldsymbol{\theta}_b$ by performing gradient descent with step size η for J steps over a regularized square loss trained on all observed contexts and rewards, starting from the initial parameter $\boldsymbol{\theta}_0$. Then Algorithm 2 takes the J-th iterate $\boldsymbol{\theta}^{(J)}$ generated by gradient descent as its output to BatchNeuralUCB. The training procedure is described in Algorithm 2.

Meanwhile, within each batch, BatchNeuralUCB maintains the covariance matrix $\mathbf{Z}_{t_b}$ which is calculated over the gradients of the observed contexts, each taken with respect to the estimated parameter of the neural network at the beginning of that contexts' corresponding batch. Based on $\boldsymbol{\theta}_b$ and $\mathbf{Z}_{t_b}$, BatchNeuralUCB calculates the UCB estimate of reward $\widehat{f}_b(\cdot)$, as Line 7 in Algorithm 1 suggests. The function $\widehat{f}_b(\cdot)$ is used to select actions during the b-th batch. In particular, at round t, BatchNeuralUCB receives contexts $\{\mathbf{x}_{t,a}\}_{a=1}^K$ and picks a_t which maximizes the optimistic reward $\widehat{f}_b(x_{t,a})$ (see Line 10). Once this batch finishes, the rewards r_{t,a_t} collected during this batch are observed (Line 5), and the process continues.

We compare BatchNeuralUCB with several related algorithms. BatchNeuralUCB is similar to NeuralUCB [41], except that BatchNeuralUCB only updates its parameter in batch and NeuralUCB updates its parameter per round. The batch update scheme greatly reduces the total computational complexity of BatchNeuralUCB. Meanwhile, it is worth noting that arm elimination-based algorithms for linear bandits algorithm [16] also updates its parameter in batch. However, their proposed algorithm works only for fixed action sets and linear rewards. In contrast, our BatchNeuralUCB deals with the general case where the action sets can change over time and when the interaction between the contexts and rewards are nonlinear.

4.1 Fixed Batch Size Scheme

For the fixed batch scheme, BatchNeuralUCB predefines the batch grid $\mathcal{T} = \{t_0, t_1, \dots t_B\}$ as a deterministic set depending on the time horizon T and number of batches B.

In particular, BatchNeuralUCB selects the simple uniform batch grid, with $t_b = b \cdot \lfloor T/B \rfloor + 1$, as suggested in Eq. (4.1). It is easy to see that when $B = T$, BatchNeuralUCB updates the network parameters at each round, reducing to NeuralUCB. [20] also studied the fixed batch size scheme, but for the linear reward.

4.2 Adaptive Batch Size Scheme

Unlike the fixed batch size scheme, in the adaptive batch size scheme, BatchNeuralUCB does not predefine the batch grid. Instead, it dynamically selects the batch grids based on the previous observations. Specifically, at any time t, the algorithm calculates the determinant of the covariance matrix and keeps track of its ratio to the determinant of the covariance matrix calculated at the end of the previous batch. If this ratio is larger than a hyperparameter q and the number of utilized batches is less than the budget B, then BatchNeuralUCB starts a new batch. This idea used in the adaptive batch size scheme is similar to the *rarely switching* updating rule introduced in Abbasi-Yadkori et al. [1] for linear bandits. The difference is that while Abbasi-Yadkori et al. [1] applies this idea directly to the contexts $\{\mathbf{x}^i\}_i$, Algorithm 1 applies it to the gradient mapping of contexts.

5 MAIN RESULTS

In this section, we propose our main theoretical results about Algorithm 1. First, we propose the definition of the neural tangent kernel (NTK) matrix [21]. Similar definition has also been made in Zhou et al. [41].

Definition 5.1. Let $\{\mathbf{x}^i\}_{i=1}^{TK}$ be a set of contexts. Define

$$\begin{aligned}
\widetilde{\mathbf{H}}^{(1)}_{i,j} &= \Sigma^{(1)}_{i,j} = \langle \mathbf{x}^i, \mathbf{x}^j \rangle, \ \mathbf{A}^{(l)}_{i,j} = \begin{pmatrix} \Sigma^{(l)}_{i,i} & \Sigma^{(l)}_{i,j} \\ \Sigma^{(l)}_{i,j} & \Sigma^{(l)}_{j,j} \end{pmatrix}, \\
\Sigma^{(l+1)}_{i,j} &= 2\mathbb{E}_{(u,v)\sim N(\mathbf{0}, \mathbf{A}^{(l)}_{i,j})} \left[\sigma(u)\sigma(v)\right], \\
\widetilde{\mathbf{H}}^{(l+1)}_{i,j} &= 2\widetilde{\mathbf{H}}^{(l)}_{i,j} \mathbb{E}_{(u,v)\sim N(\mathbf{0}, \mathbf{A}^{(l)}_{i,j})} \left[\sigma'(u)\sigma'(v)\right] + \Sigma^{(l+1)}_{i,j}.
\end{aligned}$$

Then, $\mathbf{H} = (\widetilde{\mathbf{H}}^{(L)} + \Sigma^{(L)})/2$ is called the *neural tangent kernel (NTK)* matrix on the context set $\{\mathbf{x}^i\}_i$. For simplicity, let $\mathbf{h} \in \mathbb{R}^{TK}$ denote the vector $(h(\mathbf{x}^i))_{i=1}^{TK}$.

In general, the NTK gram matrix $\mathbf{H}$ describes the relationship between each pair of contexts $\mathbf{x}^i, \mathbf{x}^j$ which we will face during the decision making process, with the help of a neural network function mapping. Specifically, the (i, j)-th entry of $\mathbf{H}$ denotes the cosine distance between $\mathbf{g}(\mathbf{x}^i), \mathbf{g}(\mathbf{x}^j)$ when the width m goes to infinity, which is, $\mathbf{H}_{i,j} = \lim_{m\to\infty} \langle \mathbf{g}(\mathbf{x}^i), \mathbf{g}(\mathbf{x}^j)\rangle/m$. We need the following assumption over the NTK gram matrix $\mathbf{H}$.

Assumption 5.2. The NTK matrix satisfies $\mathbf{H} \succeq \lambda_0 \mathbf{I}$.
Remark 5.3. Assumption 5.2 suggests that the NTK matrix $\mathbf{H}$ is non-singular. Such a requirement can be guaranteed as long as *no* two contexts in $\{\mathbf{x}^i\}_i$ are parallel [15].

We also need the following assumption over the initialized parameter $\boldsymbol{\theta}_0$ and the contexts $\mathbf{x}^i$.
Assumption 5.4. For any $1 \le i \le TK$, the context $\mathbf{x}^i$ satisfies $\|\mathbf{x}^i\|_2 = 1$ and $[\mathbf{x}^i]_j = [\mathbf{x}^i]_{j+d/2}$. Meanwhile, the initial parameter $\boldsymbol{\theta}_0 = [\text{vec}(\mathbf{W}_1)^\top, \dots, \text{vec}(\mathbf{W}_L)^\top]^\top$ is initialized as follows: for $1 \le l \le L-1$, $\mathbf{W}_l$ is set to $\begin{pmatrix} \mathbf{W} & \mathbf{0} \\ \mathbf{0} & \mathbf{W} \end{pmatrix}$, where each entry of $\mathbf{W}$ is generated independently from $N(0, 4/m)$; $\mathbf{W}_L$ is set to $(\mathbf{w}^\top, -\mathbf{w}^\top)$, where each entry of $\mathbf{w}$ is generated independently from $N(0, 2/m)$.
Remark 5.5. Assumption 5.4 suggests that the context $\mathbf{x}^i$ and the initial parameter $\boldsymbol{\theta}_0$ should be 'symmetric' considering each coordinate. It can be verified that under such an assumption, for any $i \in [TK]$ we have $f(\mathbf{x}^i; \boldsymbol{\theta}_0) = 0$, which is crucial to our analysis. Meanwhile, for any context $\mathbf{x}$ that does not satisfy the assumption, we can always construct a satisfying new context $\mathbf{x}'$ by setting $\mathbf{x}' = [\mathbf{x}^\top, \mathbf{x}^\top]^\top/\sqrt{2}$.

We also need the following definition of the effective dimension, which has been adapted in Srinivas et al. [36], Valko et al. [37], Zhou et al. [41].
Definition 5.6. The effective dimension $\widetilde{d}$ of the neural tangent kernel matrix on contexts $\{\mathbf{x}^i\}_{i=1}^{TK}$ is defined as

$$\widetilde{d} = \frac{\log \det(\mathbf{I} + \mathbf{H}/\lambda)}{\log(1 + TK/\lambda)}.$$

Remark 5.7. The notion of effective dimension $\widetilde{d}$ is similar to the *information gain* introduced in Srinivas et al. [36] and *effective dimension* introduced in Valko et al. [37]. Intuitively, $\widetilde{d}$ measures how quickly the eigenvalues of $\mathbf{H}$ diminish, and it will be upper bounded by the dimension of the RKHS space spanned by $\mathbf{H}$ [41].

The following two theorems characterize the regret bounds of BatchNeuralUCB under two different update schemes. We first show the regret bound of BatchNeuralUCB under the fixed batch size update scheme.
Theorem 5.8. Suppose Assumptions 5.2 and 5.4 hold. Setting $m = \text{poly}(T, L, K, \lambda^{-1}, \lambda_0^{-1}, S^{-1}, \log(1/\delta))$ and $\lambda \ge S^{-2}$, where S is a parameter satisfying $S \ge \sqrt{2\mathbf{h}^\top \mathbf{H}^{-1}\mathbf{h}}$. There exist positive constants C_1, C_2, C_3 such that, if

$$\beta_t = C_1 \Bigg[\Bigg(\nu \sqrt{\log \frac{\det \mathbf{Z}_t}{\det \lambda \mathbf{I}} - 2\log\delta} + \sqrt{\lambda} S \Bigg) + (\lambda + tL)(1 - \eta m \lambda)^{J/2} \sqrt{t/\lambda} \Bigg],$$

$J = 2\log(\lambda S/(\sqrt{T}\lambda + C_2 T^{3/2} L)) TL/\lambda$, $\eta = C_3(mTL + m\lambda)^{-1}$, then with probability at least $1 - \delta$, the regret of Algorithm 1 with fixed batch size scheme is bounded as follows:

$$R_T = \widetilde{O}\Bigg(\Big(\nu \widetilde{d} + \sqrt{\lambda \widetilde{d}} S \Big) \sqrt{T} + \widetilde{d} T/B \Bigg).$$

Here we provide a proof sketch of Theorem 5.8 to show the main technical challenges in deriving the regret bound of batched neural bandits.

Proof sketch of Theorem 5.8. To derive the batch-dependent regret bound R_T, we define the following set C:

$$C = \{ b \in \{0, \cdots, B-1\} : \det(\mathbf{Z}_{t_{b+1}})/\det(\mathbf{Z}_{t_b}) > 2 \}.$$

Then it is easy to verify that for every $b \notin C$ and $t_b \le t < t_{b+1} - 1$, $\det(\mathbf{Z}_t)/\det(\mathbf{Z}_{t_b}) \le 2$. We can bound $|C|$ as follows:

$$\frac{\det(\mathbf{Z}_{T+1})}{\det(\lambda \mathbf{I})} = \prod_{b=0}^{B-1} \frac{\det(\mathbf{Z}_{t_{b+1}})}{\det(\mathbf{Z}_{t_b})} \overset{\text{(a)}}{\ge} \prod_{b\in C} \frac{\det(\mathbf{Z}_{t_{b+1}})}{\det(\mathbf{Z}_{t_b})} \overset{\text{(b)}}{\ge} 2^{|C|},$$

where (a) holds since $\mathbf{Z}_{t_{b+1}} \ge \mathbf{Z}_{t_b}$ and (b) holds due to the definition of C. Based on C, we decompose the regret as follows.

$$R_T = \bigg(\underbrace{\sum_{b\in C}}_{I_1} + \underbrace{\sum_{b\notin C}}_{I_2} \bigg) \sum_{t=t_b}^{t_{b+1}-1} [h(\mathbf{x}_{t,a_t^*}) - h(\mathbf{x}_{t,a_t})],$$

Here, $I_1 = \widetilde{O}(|C|T/B)$ due to the fact that $0 \le h \le 1$, and I_2 can be bounded as

$$I_2 \overset{\text{(a)}}{=} \widetilde{O}\Bigg(\beta_T \sum_{b\notin C} \sum_{t=t_b}^{t_{b+1}-1} \min\Big\{ \|\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_t^{-1}}, 1\Big\}\Bigg) \overset{\text{(b)}}{=} \widetilde{O}\Bigg(\beta_T \sqrt{T \log \frac{\det \mathbf{Z}_{T+1}}{\det \lambda \mathbf{I}}}\Bigg),$$

where (a) holds due to the definition of C and the neural network function approximation lemma (See Lemma 7.1), (b) holds due to the elliptical potential lemma (See Lemma 7.3) and Cauchy-Schwarz inequality. Therefore, we have

$$R_T = \widetilde{O}\Bigg(2 \log \frac{\det(\mathbf{Z}_{T+1})}{\det(\lambda \mathbf{I})} T/B + \beta_T \sqrt{T \log \frac{\det \mathbf{Z}_{T+1}}{\det \lambda \mathbf{I}}}\Bigg).$$

Finally, by the definition of effective dimension, we have $\log(\det \mathbf{Z}_T / \log \lambda \mathbf{I}) = \widetilde{O}(\widetilde{d})$ and $\beta_T = \widetilde{O}(\nu \sqrt{\log(\det \mathbf{Z}_T/\log \lambda \mathbf{I})} + \sqrt{\lambda} S) = \widetilde{O}(\nu\sqrt{\widetilde{d}} + \sqrt{\lambda} S)$, which completes the proof. □

Remark 5.9. Suppose h belongs to the RKHS space of NTK kernel $\mathcal{H}$ with a finite RKHS norm $\|h\|_{\mathcal{H}}$, then $\sqrt{\mathbf{h}^\top \mathbf{H}^{-1} \mathbf{h}} \le \|h\|_{\mathcal{H}}$ (Appendix A.2, Zhou et al. 41). Therefore, by treating ν as a constant and setting $S = \sqrt{2}\|h\|_{\mathcal{H}}$, $\lambda = S^{-2}$, the regret is on the order $\widetilde{O}(\widetilde{d}\sqrt{T} + \widetilde{d}T/B)$. This suggests setting $B = \sqrt{T}$ in order to obtain the standard regret $\widetilde{O}(\widetilde{d}\sqrt{T})$.

Remark 5.10. Han et al. [20] proposed a lower bound on the regret of 2-armed linear bandits with d-dimensional contexts, which suggests that for any algorithm with a fixed B-batch size scheme, the regret is no less than

$$\Omega(\sqrt{dT} + \min\{T\sqrt{d}/B, T/\sqrt{B}\}). \tag{5.1}$$

Eq. (5.1) shows that to obtain an $\widetilde{O}(\sqrt{T})$-regret, at least $\Omega(\sqrt{T})$ number of batches are needed, which implies that our choice of B as $\sqrt{T}$ is tight.

We have the following theorem for Algorithm 1 under the adaptive batch size scheme.

Theorem 5.11. Suppose Assumptions 5.2 and 5.4 hold. Let $S, \lambda, J, \eta, \{\beta_t\}$ be selected as in Theorem 5.8. Then with probability at least $1 - \delta$, the regret of Algorithm 1 with the adaptive batch size scheme can be bounded by

$$R_T = \widetilde{O}\bigg(\sqrt{\max\{q, (1 + TK/\lambda)^{\widetilde{d}}/q^B\}}\Big(\nu \widetilde{d} + \sqrt{\lambda \widetilde{d}} S\Big)\sqrt{T}\bigg).$$

Remark 5.12. By treating ν as a constant and assuming that h belongs to the RKHS space of NTK kernel $\mathcal{H}$ with a finite RKHS norm $\|h\|_{\mathcal{H}}$, and by setting S and λ as Remark 5.9 suggests, the regret is bounded by $\widetilde{O}(\sqrt{\max\{q, (1+TK)^{\widetilde{d}}/q^B\}}\widetilde{d}\sqrt{T})$.

Remark 5.13. To achieve an $\widetilde{O}(\widetilde{d}\sqrt{T})$ regret, here B needs to be chosen as $\Omega(\widetilde{d}\log(1+TK/\lambda))$ and $q = \widetilde{\Theta}((1+TK/\lambda)^{\widetilde{d}/B})$. As a comparison, for the linear bandits case, Ruan et al. [33] has shown that an $O(d\log d\log T)$ number of batches is necessary to achieve a $\widetilde{O}(\sqrt{T})$ regret. Therefore, our choice of B as $\widetilde{d}\log(T)$ is tight up to a $\log\widetilde{d}$ factor.

6 NUMERICAL EXPERIMENTS

In this section, we run numerical experiments on both synthetically generated and real data to validate our theoretical findings.

6.1 Synthetic Data

We compare the performance of our proposed BatchNeuralUCB (BNUCB) algorithm with fixed and adaptive size batches for several values of B and q with two fully sequential benchmarks on synthetic data generated as follows. The fully sequential benchmarks considered are: (1) LinUCB algorithm [28] and (2) NeuralUCB algorithm [41]. We run our experiments on two different reward functions.

- **Cosine Reward.** Consider the cosine reward function given by $r_{t,a} = \cos(3\mathbf{x}_{t,a}^\top\boldsymbol{\theta}^*) + \xi_t$ where $\{\mathbf{x}_{t,1}, \mathbf{x}_{t,2}, \cdots, \mathbf{x}_{t,K}\}$ are contexts generated at time t according to $U[0,1]^d$ independent of each other. The parameter $\boldsymbol{\theta}^*$ is the unknown parameter of model that is generated according to $U[0,1]^d$, normalized to satisfy $\|\boldsymbol{\theta}^*\|_2 = 1$. The noise ξ_t is generated according to $N(0, 0.25)$ independent of all other variables. We choose $d = 10$ and $K = 4$.
- **Quadratic Reward.** Consider the quadratic reward function given by $r_{t,a} = \mathbf{x}_{t,a}^\top\mathbf{A}^\top\mathbf{A}\mathbf{x}_{t,a} + \xi_t$ where $\{\mathbf{x}_{t,1}, \mathbf{x}_{t,2}, \cdots, \mathbf{x}_{t,K}\}$ are contexts generated at time t according to $U[0,1]^d$ independent of each other. Each entry of the matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$ is generated according to $N(0,1)$. The noise ξ_t is generated according to $N(0, 0.25)$ independent of all other variables. We choose $d = 4$ and $K = 10$.

Hyperparameter Tuning. We select parameters as follows. For LinUCB, we search over the regularization parameter $\lambda \in \{0.001, 0.01, 0.1, 1\}$ and exploration parameter $\beta \in \{0.001, 0.01, 0.1, 1\}$ and pick the best hyperparameter configuration. We provide the selected hyperparameters of NeuralUCB and BatchNeuralUCB for cosine reward and quadratic reward respectively as follows.

- **Cosine Reward.** For NeuralUCB and BatchNeuralUCB, we train two-layers neural networks with $m = 200$ hidden layers. For both of these algorithms, we find that choosing parameters $\lambda = 0.01$ and $\beta_t = 0.001$ can have very good performance. Finally, in the iterations where the policy is updated, the parameters of neural networks are updated for $J = 200$ iterations using stochastic gradient descent with $\eta = 0.01$.
- **Quadratic Reward.** For NeuralUCB and BatchNeuralUCB, we train two-layers neural networks with $m = 100$ hidden units. For both of these algorithms, we find that choosing parameters $\lambda = 0.01$ and $\beta_t = 0.01$ can have very good performance. Finally, in the iterations where the policy is updated, the parameters of neural networks are updated for $J = 200$ iterations using stochastic gradient descent with $\eta = 0.005$.

We choose $T = 2000$ and repeat the experiments for 10 times and generate the box plot of the total regret of algorithms together with its standard deviation.

Results. The per-instance regret results for cosine reward and quadratic reward are depicted in Figures 2 and 3 respectively. Figure 4 shows the scatter plot of regret vs execution time in second for 5 instances (selected randomly out of all 10 repeated experiments) for all algorithms.

We can make the following observations. Due to model misspecifications, LinUCB does not perform very well on both rewards. Our proposed BNUCB works pretty well in both fixed and

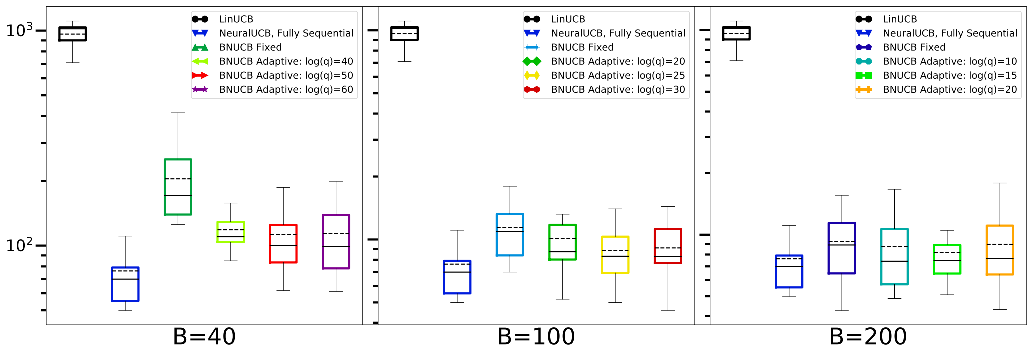


Fig. 2. Distribution of per-instance regret on Synthetic data. The solid and dashed lines indicate the median and the mean respectively. Note that BNUCB stands for BatchNeuralUCB and also that the regrets are plotted on the log scale.

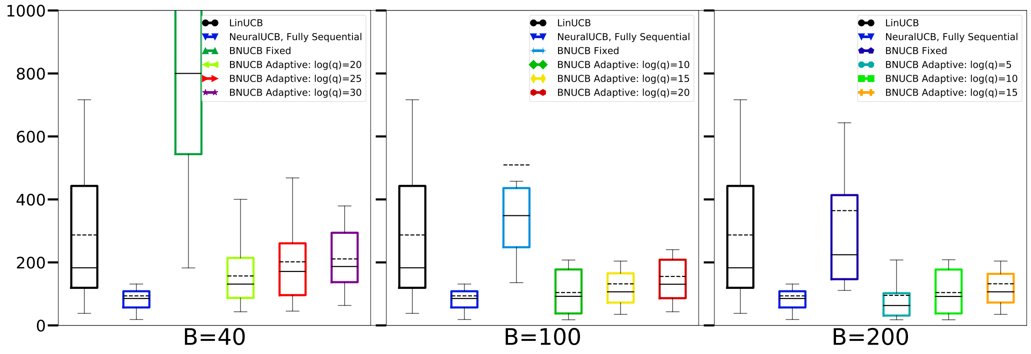


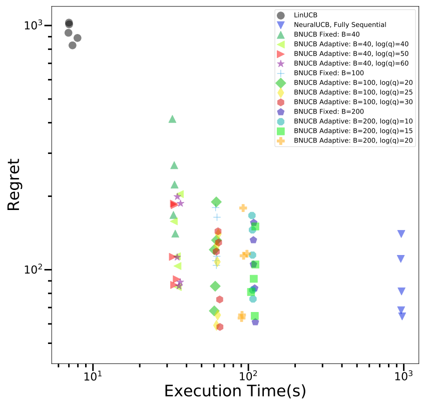
Fig. 3. Distribution of per-instance regret on Synthetic data with quadratic reward. The solid and dashed lines indicate the median and the mean respectively. Note that BNUCB stands for BatchNeuralUCB.

adaptive schemes, while keeping the total number of policy updates and also the running time small. In particular, the adaptive BatchNeuralUCB algorithm with only $B = 40$ batches and all configurations for q, i.e. $\log(q) = 20, 25, 30$, outperforms LinUCB and achieves a very close performance to that of NeuralUCB while enjoying a very fast execution time. The gap in the regret with the fully sequential NeuralUCB algorithm becomes smaller for configurations with $B = 100$ and it becomes almost insignificant for $B = 200$. At the same time, the number of policy updates and also execution times of all configurations of BNUCB for all pairs of (B, q) are almost ten times smaller than the fully sequential version. The last observation is that, for a given batch size B, the adaptive batch scheme configurations have better performance compared to the fixed ones.

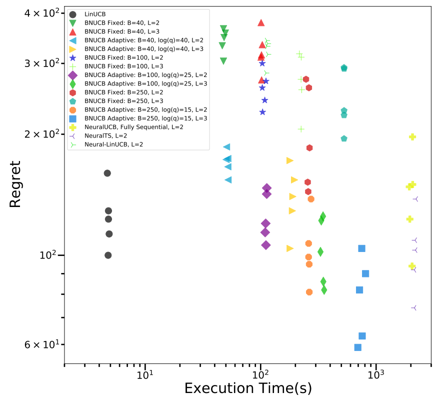
6.2 Real Data

We repeat the above experiments this time using two real datasets: Mushroom and Magic Telescope[2], both from the UCI repository originally designed for the classification task. For each sample in a dataset $s_t = (c_t, l_t)$ with context $c_t \in \mathbb{R}^d$ and label $l_t \in [K]$, we consider the zero-one

[2]This dataset can be found at http://archive.ics.uci.edu/ml/datasets/MAGIC+GAMMA+Telescope

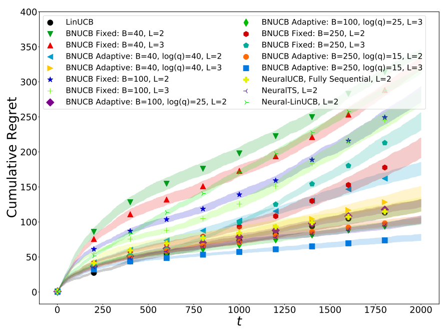


(a) Cosine

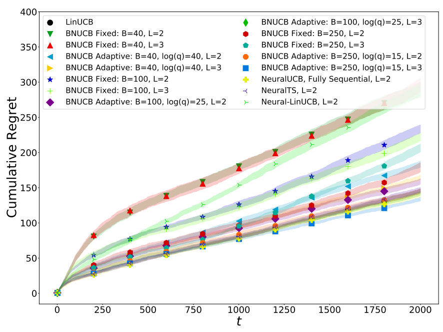


(b) Quadratic

Fig. 4. Scatter plot of regret vs execution time for 5 instances selected at random on synthetic data with synthetic reward.



(a) Mushroom



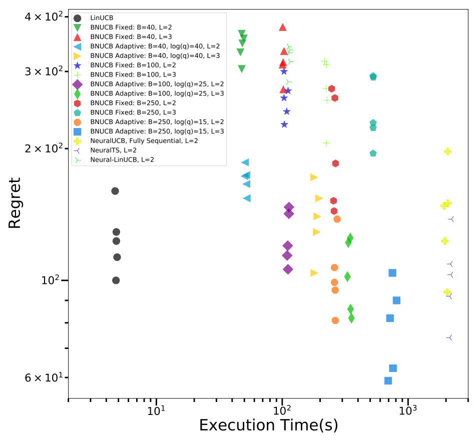
(b) Magic Telescope

Fig. 5. Cumulative regret of algorithms on two real datasets. Note that BNUCB stands for BatchNeuralUCB, our proposed algorithm (Algorithm 1).

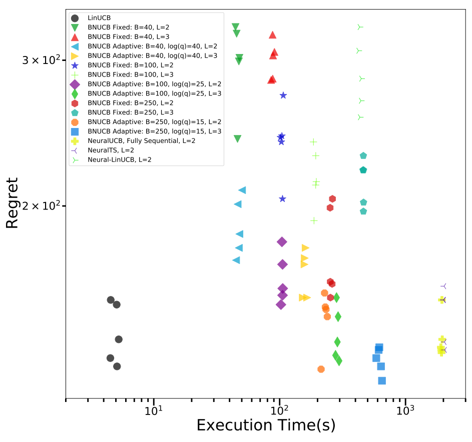
reward defined as $r_{t,a_t} = \mathbb{1}\{a_t = l_t\}$ and generate our context vectors as $\{\mathbf{x}_{t,a} \in \mathbb{R}^{Kd} : \mathbf{x}_{t,a} = [\underbrace{0,\cdots,0}_{a-1 \text{ times}}, c_i, \underbrace{0,\cdots,0}_{K-a \text{ times}}], a \in [K]\}$.

For each dataset, we repeat the experiment for 10 times and report the mean cumulative regret and its confidence intervals for LinUCB, NeuralUCB, NeuralTS [40], Neural-LinUCB [38] and the proposed BatchNeuralUCB for several configurations of parameters. For each instance, we select $T = 2000$ random samples without replacement from the dataset and run all algorithms on that instance. Note that in this simulation, NeuralUCB, NeuralTS use two-layer neural networks with $m = 100$ hidden units per layer, while Neural-LinearUCB uses a two-layer neural networks with $m = 800$ hidden units. Our proposed BatchNeuralUCB uses two-layer or three-layer neural networks with $m = 100$ hidden units.

Hyperparameter Tuning. We select the parameters as follows. For LinUCB, we search over the space of regularization parameters $\lambda \in \{0.001, 0.01, 0.1, 1\}$ and the exploration parameter $\beta_t \in \{0.001, 0.01, 0.1, 1\}$ and select the hyperparameters with the lowest average regret. For NeuralUCB and NeuralTS algorithms, we consider two-layer neural networks with $m = 100$ hidden units. For BatchNeuralUCB, we consider both $L = 2$ and $L = 3$ layers fully connected neural networks with $m = 100$ hidden units in each layer. For Neural-LinUCB we use a two-layer neural network with $m = 800$ hidden layers. For all algorithms with the exception of Neural-LinUCB, during iterations that policy update is allowed (at the end of batches for BatchNeuralUCB and every iteration for NeuralUCB), we use stochastic gradient descent with $J = 200$ iterations to update the network parameters. For Neural-LinUCB, $J = 100$ is selected. The learning rate of BatchNeuralUCB is selected as $\eta = 0.02$ for MagicTelescope and $\eta = 0.05$ for the Mushroom dataset. For NeuralUCB and NeuralTS, $\eta = 0.02$ is selected on the MagicTelescope and $\eta = 0.01$ for the Mushroom dataset. Finally, for Neural-LinUCB, $\eta = 0.01$ on the MagicTelescope and $\eta = 0.002$ on the Mushroom Dataset are selected. For BatchNeuralUCB, we find that choosing parameters $\lambda = 0.001$ and $\beta_t = 0.001$ works well, while for NeuralUCB, NeuralTS, and Neural-LinUCB we search over $\lambda \in \{0.001, 0.01\}$ and $\beta_t \in \{0.0001, 0.001\}$.



(a) Mushroom



(b) Magic

Fig. 6. Scatter plot of regret vs execution time for 5 instances selected at random on real datasets.

Results. The results are depicted in Figures 5 and 6. We can make the following observations. First, three-layer BNUCB algorithm with $B = 250$ batches outperforms all other algorithms. Furthermore, as the number of batches of BNUCB increases, the regret decreases and the performance gets closer to that of fully sequential NeuralUCB, or even starts to outperform NeuralUCB when $L = 3$ layers are used in BNUCB. Meanwhile, the proposed batch algorithms keep the total number of policy changes limited and improve the running time of the NeuralUCB (see Figure 1). Finally, across these configurations, adaptive batch algorithms with fewer batches outperform the fixed batch algorithms with more batches. For example, in all datasets, the adaptive BNUCB algorithm with $B = 100$ outperform fixed BNUCB algorithm with $B = 250$. This validates our theory that the minimum number of batches required for getting the optimal $\widetilde{O}(\sqrt{T})$ regret is much smaller in the adaptive batch setting compared to the fixed batch setting (order of $\log T$ vs $\sqrt{T}$). In terms of running time, since LinUCB is a linear algorithm whose number of parameters is far less than those of neural bandit algorithms (e.g., BatchNeuralUCB), it is not surprising to see that the running time of LinUCB is less than that of neural bandit algorithms. The result suggests that the neural bandit algorithms are more favorable for the case where the sample efficiency (i.e., total regret) is more important than computationally efficiency (i.e., running time).

For the execution time, as can be observed in Figures 6(a) and 6(b), in both datasets the $L = 3$ layer BatchNeuralUCB with $B = 250$ adaptive batches outperforms all other existing algorithms in terms of average regret. Furthermore, this algorithm executes almost 4 times faster than the fully sequential NeuralUCB and NeuralTS algorithms and keeps the total number of policy updates capped at $B = 250$. This clearly shows how batching allows us to add a full layer to our network, outperforms the fully sequential benchmarks, while keeping the total execution time 4 times smaller. The performance of other adaptive BatchNeuralUCBbenchmarks is also very close to that of NeuralUCB and NeuralTS while reducing the total computation cost significantly.

We also report the effective dimension $\widetilde{d}$ (Definition 5.6) of both datasets here. To obtain the NTK gram matrix $\mathbf{H}$, we apply the NEURAL TANGENTS package [30] on both Mushroom and Magic Telescope datasets. We have $K = 2$ and set $T = 2000$ and $\lambda = 0.001$, aligned with our experiment setting. Since our experiments are repeated 10 times, we report the final effective dimension to be the average of 10 effective dimensions over 10 random batches. The effective dimension of Mushroom is 246.32, and the effective dimension of Magic Telescope is 231.49. We can see that both effective dimensions are nearly the same, which explains why BatchNeuralUCB performs nearly the same on Mushroom and Magic Telescope.

7 PROOF OF THE MAIN RESULTS

7.1 Proof of Theorem 5.8

To prove Theorem 5.8, we need the following lemmas. The first lemma from Zhou et al. [41] suggests that at each time t within the b-th batch, the difference between the reward of the optimal action $\mathbf{x}_{t,a_t^*}$ and the selected action $\mathbf{x}_{t,a_t}$ can be upper bounded by a bonus term defined based on the confidence radius β_{t_b}, the gradients $\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_b)$ and the covariance matrix $\mathbf{Z}_{t_b}$.

Lemma 7.1 (Lemma 5.3, Zhou et al. 41). Suppose Assumptions 5.2 and 5.4 hold. Let $a_t^* = \mathrm{argmax}_{a\in[K]}\, h(\mathbf{x}_{t,a})$. There exist positive constants $\{\bar{C}_i\}_{i=1}^4$ such that for any $\delta \in (0,1)$, if $\eta \le \bar{C}_1(mTL + m\lambda)^{-1}$ and $m = \mathrm{poly}(T, L, K, \lambda^{-1}, \lambda_0^{-1}, S^{-1}, \log(1/\delta))$, then with probability at least $1-\delta$, the following holds for all $0 \le b \le B$ and $t_b \le t < t_{b+1}$,

$$h\big(\mathbf{x}_{t,a_t^*}\big) - h\big(\mathbf{x}_{t,a_t}\big) \le 2\widehat{\beta}_{t_b} \min\left\{\|\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_b)/\sqrt{m}\|_{\mathbf{Z}_{t_b}^{-1}}, 1\right\} + m^{-1/6}\sqrt{\log m}\,\xi(T),$$

where $\xi(T) = \bar{C}_3\big(ST^{7/6}\lambda^{-1/6}L^{7/2} + T^{5/3}\lambda^{-2/3}L^3\big)$ and

$$\begin{aligned}\widehat{\beta}_t &= \bar{C}_4\sqrt{1 + m^{-1/6}\sqrt{\log m}L^4 t^{7/6}\lambda^{-7/6}} \\ &\quad\cdot \left(\nu\sqrt{\log\frac{\det \mathbf{Z}_t}{\det \lambda \mathbf{I}} + m^{-1/6}\sqrt{\log m}L^4 t^{5/3}\lambda^{-1/6} - 2\log\delta} + \sqrt{\lambda}S\right) \\ &\quad + (\lambda + tL)\Big[(1-\eta m\lambda)^{J/2}\sqrt{t/\lambda} + m^{-1/6}\sqrt{\log m}L^{7/2}t^{5/3}\lambda^{-5/3}(1+\sqrt{t/\lambda})\Big].\end{aligned}$$

Next lemma bounds the log-determinant of covariance matrix $\mathbf{Z}_{T+1}$ by the effective dimension $\widetilde{d}$.
Lemma 7.2. There exists a constant $\bar{C} > 0$ such that if $m = \text{poly}(T, L, K, \lambda^{-1}, \lambda_0^{-1}, S^{-1}, \log(1/\delta))$, then with probability at least $1-\delta$, we have

$$\log \frac{\det \mathbf{Z}_{T+1}}{\det \lambda \mathbf{I}} \leq \widetilde{d} \log(1 + TK/\lambda) + 1 + \bar{C} m^{-1/6} \sqrt{\log m} L^4 T^{5/3} \lambda^{-1/6}.$$

Proof. By Eqs. (B. 16) and (B. 19) in Zhou et al. [41], we can obtain our statement when m is large enough. □

Lemma 7.3 (Lemma 11, Abbasi-Yadkori et al. 1)**.** For any $\{\mathbf{x}_t\}_{t=1}^T \subset \mathbb{R}^d$ that satisfies $\|\mathbf{x}_t\|_2 \leq L$, let $\mathbf{A}_0 = \lambda \mathbf{I}$ and $\mathbf{A}_t = \mathbf{A}_0 + \sum_{i=1}^{t-1} \mathbf{x}_i \mathbf{x}_i^\top$, then we have

$$\sum_{t=1}^T \min\{1, \|\mathbf{x}_t\|_{\mathbf{A}_t^{-1}}\}^2 \leq 2 \log \frac{\det \mathbf{A}_{T+1}}{\det \lambda \mathbf{I}}.$$

Lemma 7.4 (Lemma 12, Abbasi-Yadkori et al. 1)**.** Suppose $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{d\times d}$ are two positive definite matrices satisfying $\mathbf{A} \succeq \mathbf{B}$, then for any $\mathbf{x} \in \mathbb{R}^d$, $\|\mathbf{x}\|_{\mathbf{A}} \leq \|\mathbf{x}\|_{\mathbf{B}} \cdot \sqrt{\det(\mathbf{A})/\det(\mathbf{B})}$.
Now we begin our proof of Theorem 5.8.

Proof of Theorem 5.8. Define the set C as follows:

$$C = \{b \in \{0, \cdots, B-1\} : \det(\mathbf{Z}_{t_{b+1}})/\det(\mathbf{Z}_{t_b}) > 2\}.$$

Then we have for every $b \notin C$ and $t_b \leq t < t_{b+1} - 1$,

$$\frac{\det(\mathbf{Z}_t)}{\det(\mathbf{Z}_{t_b})} \leq \frac{\det(\mathbf{Z}_{t_{b+1}})}{\det(\mathbf{Z}_{t_b})} \leq 2.$$

Based on C, we decompose the regret as follows.

$$\begin{aligned}
R_T &= \sum_{b\in C} \sum_{t=t_b}^{t_{b+1}-1} [h(\mathbf{x}_{t,a_t^*}) - h(\mathbf{x}_{t,a_t})] + \sum_{b\notin C} \sum_{t=t_b}^{t_{b+1}-1} [h(\mathbf{x}_{t,a_t^*}) - h(\mathbf{x}_{t,a_t})] \\
&\overset{(a)}{\leq} |C| \cdot T/B \cdot 2 + \sum_{b\notin C} \sum_{t=t_b}^{t_{b+1}-1} \left[2\widehat{\beta}_t \min\left\{ \|\mathbf{g}(\mathbf{x}_{t,a_t}; \boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_{t_b}^{-1}}, 1 \right\} + m^{-1/6}\sqrt{\log m}\xi(T) \right] \\
&\overset{(b)}{\leq} 2|C|T/B + m^{-1/6}\sqrt{\log m}\xi(T)T + 2\sqrt{2} \cdot \\
&\quad \widehat{\beta}_T \sum_{b\notin C} \sum_{t=t_b}^{t_{b+1}-1} \min\left\{ \|\mathbf{g}(\mathbf{x}_{t,a_t}; \boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_t^{-1}}, 1 \right\},
\end{aligned}$$

where (a) holds since $h(\mathbf{x}_{t,a_t^*}) - h(\mathbf{x}_{t,a_t}) \leq 1$ and Lemma 7.1 and (b) holds since $b \notin C$ and Lemma 7.4. Hence,

$$\begin{aligned}
R_T - 2|C|T/B - m^{-1/6}\sqrt{\log m}\xi(T)T &\overset{(a)}{\leq} 2\sqrt{2}\widehat{\beta}_T \sqrt{T \sum_{t=1}^T \min\left\{ \|\mathbf{g}(\mathbf{x}_{t,a_t}; \boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_t^{-1}}^2, 1 \right\}} \\
&\overset{(b)}{\leq} 2\sqrt{2}\widehat{\beta}_T \sqrt{T \log \frac{\det \mathbf{Z}_{T+1}}{\det \lambda \mathbf{I}}},
\end{aligned}$$

where (a) holds due to Cauchy-Schwarz inequality and (b) holds due to Lemma 7.3. We can bound $|C|$ as follows:

$$\frac{\det(\mathbf{Z}_{T+1})}{\det(\lambda\mathbf{I})} = \prod_{b=0}^{B-1}\frac{\det(\mathbf{Z}_{t_{b+1}})}{\det(\mathbf{Z}_{t_b})} \overset{(a)}{\geq} \prod_{b\in C}\frac{\det(\mathbf{Z}_{t_{b+1}})}{\det(\mathbf{Z}_{t_b})} \overset{(b)}{\geq} 2^{|C|}, \tag{7.1}$$

where (a) holds since $\mathbf{Z}_{t_{b+1}} \geq \mathbf{Z}_{t_b}$ and (b) holds due to the definition of C. Eq. (7.1) suggests that $|C| \leq \log(\det(\mathbf{Z}_{T+1})/\det(\lambda\mathbf{I}))$. Therefore,

$$R_T \leq 2\log\frac{\det(\mathbf{Z}_{T+1})}{\det(\lambda\mathbf{I})}T/B + m^{-1/6}\sqrt{\log m}\xi(T)T + 2\sqrt{2}\widehat{\beta}_T\sqrt{T\log\frac{\det\mathbf{Z}_{T+1}}{\det\lambda\mathbf{I}}}. \tag{7.2}$$

Finally, with a large enough m, by the selection of J and Lemma 7.2, we have $\log(\det\mathbf{Z}_T/\log\lambda\mathbf{I}) = \widetilde{O}(\widetilde{d})$ and $\widehat{\beta}_T = \widetilde{O}(\nu\sqrt{\log(\det\mathbf{Z}_T/\log\lambda\mathbf{I})}+\sqrt{\lambda}S) = \widetilde{O}(\nu\sqrt{\widetilde{d}}+\sqrt{\lambda}S)$. We also have $m^{-1/6}\sqrt{\log m}\xi(T)T \leq 1$. Substituting these terms into Eq. (7.2), we complete the proof. □

7.2 Proof of Theorem 5.11

Let B' be the value of b when Algorithm 1 stops. It is easy to see that $B' \leq B$, therefore there are at most B batches. We can first decompose the regret as follows, using Lemma 7.1:

$$R_T = \sum_{t=1}^{T}[h(\mathbf{x}_{t,a_t^*}) - h(\mathbf{x}_{t,a_t})] \leq 2\sum_{b=0}^{B'}\sum_{t=t_b}^{t_{b+1}-1}\widehat{\beta}_t\min\left\{\|\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_{t_b}^{-1}}, 1\right\} + m^{-1/6}\sqrt{\log m}\xi(T)T, \tag{7.3}$$

To bound Eq. (7.3), we have the following two separate cases. First, if $B' < B$, then for all $0 \leq b \leq B'$ and $t_b \leq t < t_{b+1}$, we have $\det(\mathbf{Z}_t) \leq q\det(\mathbf{Z}_{t_b})$. Therefore, we have

$$\begin{aligned} R_T - m^{-1/6}\sqrt{\log m}\xi(T)T \quad &\overset{(a)}{\leq} 2\widehat{\beta}_T\sqrt{q}\sum_{b=0}^{B'}\sum_{t=t_b}^{t_{b+1}-1}\min\left\{\|\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_t^{-1}}, 1\right\} \\ &\overset{(b)}{\leq} 2\widehat{\beta}_T\sqrt{q}\sqrt{T}\sqrt{\sum_{t=1}^{T}\min\left\{\|\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\|^2_{\mathbf{Z}_t^{-1}}, 1\right\}} \\ &\overset{(c)}{\leq} 2\widehat{\beta}_T\sqrt{T}\sqrt{q}\sqrt{2\log\frac{\det\mathbf{Z}_T}{\det\lambda\mathbf{I}}}, \end{aligned} \tag{7.4}$$

where (a) holds due to Lemma 7.4, (b) holds due to Cauchy-Schwarz inequality and (c) holds due to Lemma 7.3. Second, if $B' = B$, then for all $0 \leq b \leq B-1$, we have $\det(\mathbf{Z}_{t_{b+1}}) > q\det(\mathbf{Z}_{t_b})$. For $b = B$ and $t_B \leq t < t_{B+1}$, we have

$$\frac{\det(\mathbf{Z}_t)}{\det(\mathbf{Z}_{t_B})} \leq \frac{\det(\mathbf{Z}_T)}{\det(\lambda\mathbf{I})}\cdot\prod_{b=0}^{B-1}\frac{\det(\mathbf{Z}_{t_b})}{\det(\mathbf{Z}_{t_{b+1}})} \leq \frac{\det(\mathbf{Z}_T)}{\det(\lambda\mathbf{I})}\cdot q^{-B}. \tag{7.5}$$

Therefore, by Eq. (7.3) the regret can be bounded as

$$\begin{aligned} &R_T - m^{-1/6}\sqrt{\log m}\xi(T)T \\ &\leq 2\sum_{b=0}^{B-1}\sum_{t=t_b}^{t_{b+1}-1}\widehat{\beta}_t\min\left\{\|\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_{t_b}^{-1}}, 1\right\} \\ &\quad + \sum_{t=t_B}^{t_{B+1}-1}\widehat{\beta}_t\min\left\{\|\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_{t_b}^{-1}}, 1\right\} \end{aligned}$$

$$
\begin{aligned}
&\overset{(a)}{\le} 2\widehat{\beta}_T\sqrt{q}\sum_{b=0}^{B-1}\sum_{t=t_b}^{t_{b+1}-1}\min\Big\{\|\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_t^{-1}},1\Big\}\\
&\quad+\widehat{\beta}_T\sqrt{\frac{\det(\mathbf{Z}_T)}{q^B\det(\lambda\mathbf{I})}}\cdot\sum_{t=t_B}^{t_{B+1}-1}\min\Big\{\|\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_t^{-1}},1\Big\}\\
&\le 2\widehat{\beta}_T\max\Bigg\{\sqrt{q},\sqrt{\frac{\det(\mathbf{Z}_T)}{q^B\det(\lambda\mathbf{I})}}\Bigg\}\cdot\sum_{t=1}^{T}\min\Big\{\|\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\|_{\mathbf{Z}_t^{-1}},1\Big\}\\
&\overset{(b)}{\le} 2\widehat{\beta}_T\max\Bigg\{\sqrt{q},\sqrt{\frac{\det(\mathbf{Z}_T)}{q^B\det(\lambda\mathbf{I})}}\Bigg\}\sqrt{T}\cdot\sqrt{\sum_{t=1}^{T}\min\Big\{\|\mathbf{g}(\mathbf{x}_{t,a_t};\boldsymbol{\theta}_t)/\sqrt{m}\|^2_{\mathbf{Z}_t^{-1}},1\Big\}}\\
&\overset{(c)}{\le} 2\widehat{\beta}_T\sqrt{T}\max\Bigg\{\sqrt{q},\sqrt{\frac{\det(\mathbf{Z}_T)}{q^B\det(\lambda\mathbf{I})}}\Bigg\}\sqrt{2\log\frac{\det\mathbf{Z}_T}{\det\lambda\mathbf{I}}},
\end{aligned}
\tag{7.6}
$$

where (a) holds due to Lemma 7.4 and the following two facts: $\det(\mathbf{Z}_t)\le q\det(\mathbf{Z}_{t_b})$ for all $0\le b\le B-1$, $t_b\le t<t_{b+1}$; Eq. (7.5) for $b=B$, (b) holds due to Cauchy-Schwarz inequality and (c) holds due to Lemma 7.3. Combining Eqs. (7.4) and (7.6), we have under both $B'<B$ and $B'=B$ cases, Eq. (7.6) holds. Finally, by Lemma 7.2 and the selection of J and m, we have

$$
\begin{aligned}
&\frac{\det\mathbf{Z}_T}{\log\lambda\mathbf{I}}=\widetilde{O}((1+TK/\lambda)^{\widetilde{d}}),\\
&\log\frac{\det\mathbf{Z}_T}{\log\lambda\mathbf{I}}=\widetilde{O}(\widetilde{d}),\\
&\widehat{\beta}_T=\widetilde{O}\Bigg(\nu\sqrt{\log\frac{\det\mathbf{Z}_T}{\log\lambda\mathbf{I}}}+\sqrt{\lambda}S\Bigg)=\widetilde{O}\Big(\nu\sqrt{\widetilde{d}}+\sqrt{\lambda}S\Big).
\end{aligned}
\tag{7.7}
$$

Thus, substituting Eq. (7.7) into Eq. (7.6), we have

$$
\begin{aligned}
R_T&=\widetilde{O}\bigg(\Big(\nu\sqrt{\widetilde{d}}+\sqrt{\lambda}S\Big)\cdot\sqrt{T}\cdot\max\Big\{\sqrt{q},\sqrt{(1+TK/\lambda)^{\widetilde{d}}/q^B}\Big\}\cdot\sqrt{\widetilde{d}}\bigg)\\
&=O\bigg(\sqrt{\max\{q,(1+TK/\lambda)^{\widetilde{d}}/q^B\}}\Big(\nu\widetilde{d}+\sqrt{\lambda\widetilde{d}}S\Big)\sqrt{T}\bigg).
\end{aligned}
$$

8 CONCLUSION

In this paper, we proposed a BatchNeuralUCB algorithm which combines neural networks with the UCB technique to balance exploration-exploitation tradeoff while keeping the total number of batches limited. We studied both fixed and adaptive batch size settings and proved that BatchNeuralUCB achieves the same regret as the fully sequential version. Our theoretical results are complemented by experiments on both synthetic and real-world datasets.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their helpful comments. Quanquan Gu and Dongruo Zhou are supported in part by the National Science Foundation CAREER Award 1906169. Amin Karbasi acknowledges funding in direct support of this work from NSF (IIS-1845032), ONR (N00014- 19-1-2406), and the AI Institute for Learning-Enabled Optimization at Scale (TILOS).

REFERENCES

[1] Abbasi-Yadkori, Y., Pál, D. and Szepesvári, C. (2011). Improved algorithms for linear stochastic bandits. In *Advances in Neural Information Processing Systems.*

[2] Agrawal, S. and Goyal, N. (2013). Thompson sampling for contextual bandits with linear payoffs. In *International Conference on Machine Learning.*

[3] Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research* **3** 397–422.

[4] Auer, P., Cesa-Bianchi, N. and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning* **47** 235–256.

[5] Auer, P. and Ortner, R. (2010). Ucb revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica* **61** 55–65.

[6] Bouneffouf, D. and Rish, I. (2019). A survey on practical applications of multi-armed and contextual bandits. *arXiv preprint arXiv:1904.10040* .

[7] Bubeck, S. and Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning* **5** 1–122.

[8] Cesa-Bianchi, N., Dekel, O. and Shamir, O. (2013). Online learning with switching costs and other adaptive adversaries. *arXiv preprint arXiv:1302.4387* .

[9] Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. In *Advances in neural information processing systems.*

[10] Chen, L., Yu, Q., Lawrence, H. and Karbasi, A. (2020). Minimax regret of switching-constrained online convex optimization: No phase transition. In *Advances in Neural Information Processing Systems.*

[11] Chen, Y. and Krause, A. (2013). Near-optimal batch mode active learning and adaptive submodular optimization. *ICML (1)* **28** 8–1.

[12] Chu, W., Li, L., Reyzin, L. and Schapire, R. (2011). Contextual bandits with linear payoff functions. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics.*

[13] Dani, V., Hayes, T. P. and Kakade, S. M. (2008). Stochastic linear optimization under bandit feedback .

[14] Dekel, O., Ding, J., Koren, T. and Peres, Y. (2014). Bandits with switching costs: T 2/3 regret. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing.*

[15] Du, S. S., Zhai, X., Poczos, B. and Singh, A. (2018). Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054* .

[16] Esfandiari, H., Karbasi, A., Mehrabian, A. and Mirrokni, V. (2019). Batched multi-armed bandits with optimal regret. *arXiv preprint arXiv:1910.04959* .

[17] Esfandiari, H., Karbasi, A. and Mirrokni, V. (2021). Adaptivity in adaptive submodularity.

[18] Filippi, S., Cappe, O., Garivier, A. and Szepesvári, C. (2010). Parametric bandits: The generalized linear case. In *Advances in Neural Information Processing Systems.*

[19] Gao, Z., Han, Y., Ren, Z. and Zhou, Z. (2019). Batched multi-armed bandits problem. In *Advances in Neural Information Processing Systems.*

[20] Han, Y., Zhou, Z., Zhou, Z., Blanchet, J., Glynn, P. W. and Ye, Y. (2020). Sequential batch learning in finite-action linear contextual bandits. *arXiv preprint arXiv:2004.06321* .

[21] Jacot, A., Gabriel, F. and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems.*

[22] Jin, T., Xu, P., Xiao, X. and Gu, Q. (2020). Double explore-then-commit: Asymptotic optimality and beyond. *arXiv preprint arXiv:2002.09174* .

[23] Jun, K.-S., Jamieson, K., Nowak, R. and Zhu, X. (2016). Top arm identification in multi-armed bandits with batch arm pulls. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics.*

[24] Kittur, A., Chi, E. H. and Suh, B. (2008). Crowdsourcing user studies with mechanical turk. In *Proceedings of the SIGCHI conference on human factors in computing systems.*

[25] Langford, J. and Zhang, T. (2007). The epoch-greedy algorithm for contextual multi-armed bandits. In *Proceedings of the 20th International Conference on Neural Information Processing Systems.*

[26] Lattimore, T. and Szepesvári, C. (2020). *Bandit Algorithms.* Cambridge University Press.

[27] Le, H., Voloshin, C. and Yue, Y. (2019). Batch policy learning under constraints. In *International Conference on Machine Learning.* PMLR.

[28] Li, L., Chu, W., Langford, J. and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web.*

[29] Li, L., Lu, Y. and Zhou, D. (2017). Provably optimal algorithms for generalized linear contextual bandits. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70.* JMLR. org.

[30] Novak, R., Xiao, L., Hron, J., Lee, J., Alemi, A. A., Sohl-Dickstein, J. and Schoenholz, S. S. (2019). Neural tangents: Fast and easy infinite neural networks in python. *arXiv preprint arXiv:1912.02803* .

[31] Perchet, V., Rigollet, P., Chassang, S., Snowberg, E. et al. (2016). Batched bandit problems. *The Annals of Statistics* **44** 660–681.

[32] Riquelme, C., Tucker, G. and Snoek, J. (2018). Deep Bayesian bandits showdown: An empirical comparison of Bayesian deep networks for Thompson sampling. *arXiv preprint arXiv:1802.09127* .

[33] Ruan, Y., Yang, J. and Zhou, Y. (2020). Linear bandits with limited adaptivity and learning distributional optimal design. *arXiv preprint arXiv:2007.01980* .

[34] Simchi-Levi, D. and Xu, Y. (2019). Phase transitions and cyclic phenomena in bandits with switching constraints. In *Advances in Neural Information Processing Systems*.

[35] Slivkins, A. et al. (2019). Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning* **12** 1–286.

[36] Srinivas, N., Krause, A., Kakade, S. M. and Seeger, M. (2009). Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995* .

[37] Valko, M., Korda, N., Munos, R., Flaounas, I. and Cristianini, N. (2013). Finite-time analysis of kernelised contextual bandits. *arXiv preprint arXiv:1309.6869* .

[38] Xu, P., Wen, Z., Zhao, H. and Gu, Q. (2022). Neural contextual bandits with deep representation and shallow exploration. In *International Conference on Learning Representations*.

[39] Zahavy, T. and Mannor, S. (2019). Deep neural linear bandits: Overcoming catastrophic forgetting through likelihood matching. *arXiv preprint arXiv:1901.08612* .

[40] Zhang, W., Zhou, D., Li, L. and Gu, Q. (2021). Neural thompson sampling. In *International Conference on Learning Representations*.

[41] Zhou, D., Li, L. and Gu, Q. (2020). Neural contextual bandits with ucb-based exploration. In *International Conference on Machine Learning*. PMLR.