Active Heirarchical Metric Learning

Nicolas Beltran
Department of Computer Science
Columbia University
New York City, NY 10027
nb2838@columbia.edu

Ketan Jog
Department of Computer Science
Columbia University
New York City, NY 10027
kj2473@columbia.edu

Abstract

Many problems require a well defined notion of a distance between points in space. Constructing or finding such a measure falls into the field of metric learning. Although many algorithms exist in the field when a learner has access to a fixed dataset, there is room for improvement in terms of samples efficiency that the learner needs to know, imposition of desired structure, especially when the data appears in an *online* manner. We propose a project that reduces the problem of online/active metric learning to bandits. In case our plan turn out to be too ambitious, we have a fallback - an empirical investigation of some algorithms that have dealt with the problem in an online setting or in situations where the learner can selective query the points that it wants to know information about.

1 Introduction

2 Related Work

3 Long-term goals

4 Preliminaries

5 Problem Statement

We consider two different problems. The first problem consists of making a series of sequential predictions while learning a similiarity measure. We refer to this problem as online similarity prediction. The second problem consists of learning a similarity measure while querying points in space. We refer to this problem as active similarity learning.

5.1 Online similarity learning

We consider an online similarity learning problem played over T rounds. At round t the environment samples K pairs of points $(x_{t,k}, y_{t,k}) \in \mathbb{R}^{2n}$. The agent then chooses pair $k_t \in [K]$ and is given a reward $r_{t,k_t} \in \{1, -1\}$. We assume that there exists some similarity function unknown to the agent $\phi : \mathbb{R}^{2n} \to \{-1, 1\}$ and that the rewards are such that if at time $t \in [T]$ the agent chooses pair $(x_{t,k}, y_{t,k})$ then the reward is $\phi(x_{t,k}, y_{t,k})$.

As usual we define the regret as

$$R_T = \mathbb{E}\left[\sum_{t=1}^{T} \phi(x_{t,k_t^\star}, y_{t,k_t^\star}) - \phi(x_{t,k_t}, y_{t,k_t})\right]$$

where $k_t^\star = \text{argmax}_{k\in[K]}\phi(x_{t,k}, y_{t,k})$

5.2 Active similarity learning

We assume that the learner has access to a dataset $D = \{x_i \in \mathbb{R}^n | i \in [N]\}$ of unlabeled points and that there exists some function $\phi : \mathbb{R}^{2n} \to \{-1, 1\}$ which the learner is trying to learn. The learner

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

can query T pairs of points in this set D to obtain a dataset $D_T = \{(x_t, y_t, r_t) \mid t \in [T]\}$ where x_t and y_t are the points in space and r_t is their similarity. The learner mantains and estimate $\hat{\phi}_t \in \mathcal{F}$ of ϕ, where $\mathcal{F}$ is its function class and we denote the loss between an estimate $\hat{\phi}$ a ϕ as

$$\mathcal{L}(\phi, \hat{\phi}) = \mathbb{E}_{(x,y)\sim\mathcal{D}\times\mathcal{D}}[(\hat{\phi}(x, y) - \phi(x, y))^2]$$

In this problem the goal is to find $\min_{\phi\in\mathcal{F}} \mathcal{L}(\hat{\phi}_T, \phi)$.

6 Description of the algorithms

We provide 4 algorithms which can be grouped into neural and linear bandit version. We refer to our algorithms as OnSim-LinUCB, ActSim-LinUCB,OnSim-NeuralUCB, ActSim-NeuralUCB, depeding on whether the bandit algorithm they are based on and the probelm they are trying to solve. The algorithms are all very similar but we provide them in full for completeness.

6.1 Online similarity learning

6.1.1 OnSim-LinUCB

By assuming a linear structure in the similarity function, our first algorithm performs a straighforward reduction of the online learning problem to that of regular contextual linear bandits as described in [1]. Mathematically, we model the similarity of two points as $\phi(x, y) = x^\top Ay$. This is reasonable if we consider that

$$\phi(x, y) = x^\top Ay = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i y_j A_{i,j}$$

which we is linear in A and thus allows us to use the framework of LinUCB for our problem. The algorithm can be found in the appendix in 4. We omit it from the main text due to its similarity with the first algorithm in [1].

6.1.2 OnSim-NeuralUCB

The expressive power of the above framework is quite limited because of the fact that Algorithm 4 assumes that similarity is the result of a dot prouct in which both points are independently mapped to a new representation via a linear transformation. This will obviously not be true for many datasets. To overcome these limitations we adapt NeuralUCB [6] to our circumstances.

Formally, we assume that

$$\phi(x, y) = \cos\left(f(x; \theta), f(y; \theta)\right) = \frac{\langle f(x; \theta), f(y; \theta)\rangle}{||f(x; \theta)||_2 ||f(y; \theta)||_2}$$

[1] Here $f : \mathbb{R}^n \to \mathbb{R}^m$ is a simple neural network of the form

$$f(x; \theta) = b_L + W_L\sigma\left(b_{L-1} + W_{L-1}\sigma\left(\ldots\sigma\left(b_1 + W_1 x\right)\right)\right)$$

for some $L \in \mathbb{N}_{\geq 2}$, $W_1 \in \mathbb{R}^{d\times n}$, $W_L \in \mathbb{R}^{m\times d}$, $W_l \in \mathbb{R}^{d\times d}$ if $l \in [2, L-1]$, $b_L \in \mathbb{R}^m$, $b_l \in \mathbb{R}^d$ for $l \neq L$, and where $\sigma(x) = \max\{0, x\}$ applied component wise. We use θ to denote the flattened vector of $(W_L, b_L, \ldots, W_1, b_1)$. Using this notation OnSim-NeuralUCB is described below in Algorithm 1. We use p to denote the total number of trainable parameters in the neural network.

Intuitively, this algorithm acts optimally with the current set of parameters with a bonus for exploration, and every so often it stops back to train the network. "Train" can be found in the appendix as 3 but it simply describes the normal process of training a neural network with MSE loss.

Our algorithm is very similar to that proposed in [6] but it differs in the following aspects: 1. We adopt a siamese neural network architecture unlike the paper. 2.We add bias 3.We use cosine similarity function in our last layer 4. We restart the the matrix A after every $t \mod \tau_T = 0$ iterations 5.We use a constant exploration parameter. We propose these changes because we believe they are more sensible for our particular application (and do provide better empirical performance) but they destroy any theoretical guarantees. In [6] these assumptions alongside arguments related to the neural tangent kernel matrix [2] are used to provide a $\tilde{O}(\sqrt{T})$, bound on the regret of algorithm.

[1] It is also possible to assume that we don't normalize the last inner product, we refer to this verision and unormalized NeuralUCB. We provide some comparisons of this verison also below.

Algorithm 1: OnSim-NeuralUCB

Input : Rounds T, exploration parameter α, τ_r frequency of resets, τ_T frequency of training, E epochs for training, b_s batch size for training, ϵ learning rate.

$A \leftarrow I_p$;
for $t \in [T]$ **do**
  Observe K pairs of vectors $x_t^k \in \mathbb{R}^n$, $y_t^k \in \mathbb{R}^{\ltimes}$;
  for $k \in [K]$ **do**
    $p_{t,k} \leftarrow \phi(x, y) + \alpha\sqrt{(\nabla_\theta \phi(x_t^k, y_t^k))^\top A^{-1} \nabla_\theta \phi(x_t^k, y_t^k)}$;
  end
  Choose pair $k_t = \text{argmax}_k p_{t,k}$ with ties broken arbitrarily;
  Observe payoff $r_t \in \{-1, 1\}$;
  if $t \mod \tau_r = 0$ **then**
    $\theta \leftarrow \text{Train}(\epsilon, E, \{(x_i^{k_i}, y_i^{k_i}, r_i)\}_{i=1}^t, b_s)$;
  end
  if $t \mod \tau_T = 0$ **then**
    $A \leftarrow I_p$
  end
  $A \leftarrow A + \nabla_\theta \phi(x_t^k, y_t^k)(\nabla_\theta \phi(x_t^k, y_t^k))^\top$;
end

6.2 Active Similarity Learning

Despite the differences with online similarity learning, for the active version of the problem we modify the above algorithms only slightly. The idea is to use the optimism bonus as a measure of uncertainty and ignore completely the reward.

Algorithm 2: Active-LinUCB

Input : Rounds T and exploration parameter α

$A \leftarrow I_{n^2}$;
$b \leftarrow 0_{n^2}$;
for $t \in [T]$ **do**
  $\theta_t \leftarrow A^{-1}b$ Observe K pairs of vectors $x^k \in R^n$, $y^k \in R^n$;
  Create $z_{t,k} = (x_1^k y_1^k, x_1^k y_1^k, \ldots, x_n^k y_{n-1}^k, x_n^k y_n^k)$;
  for $k \in [K]$ **do**
    $p_{t,a} \leftarrow \alpha\sqrt{z_{t,k} A^{-1} x_{t,a}}$
  end
  Choose action pair $k_t = \text{argmax}_a p_{t,a}$ with ties broken arbitrarily.;
  Observe label $r_t \in \{-1, 1\}$;
  $A \leftarrow A + z_{t,k_t} z_{t,k_t}^\top$;
  $b \leftarrow z_{t,j_t} r_t$;
end

Using this framework, each time we have to make a query we can sample a subset of points in the unlabeled dataset D and then we reveal the label of the pair with the highest uncertainty as measured by the bonus. We emphasize that r_t now references a label rather than a reward. Algorithm 2 describes this processes formally in the case of ActiveSim-LinUCB and similar modification applies for ActiveSim-NeuralUCB [2]. In Algorithm 5 the most important change is line 5 and in algorithm 2 the most important change is line 7.

[2] A precise implementation can be found in the appendix

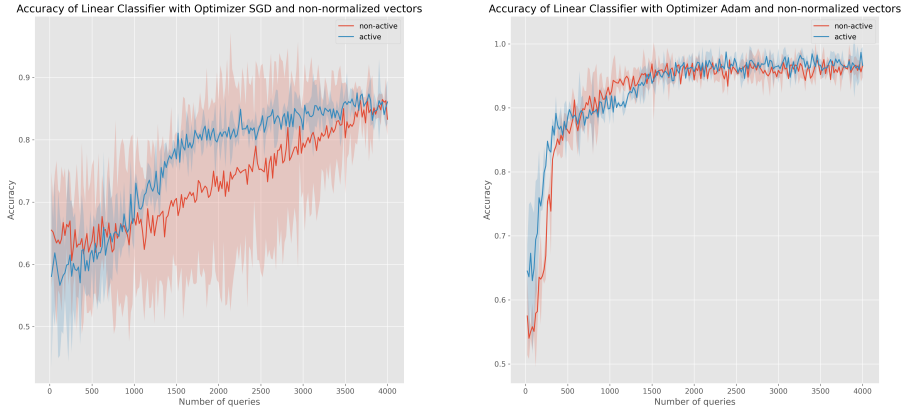

Figure 1: Performance SVM on non-normalized learned features Adam vs SGD

7 Empirical Evaluation

Below we describe the set of experiments that we performed. We test separately NeuralUCB and LinUCB based algorithms as we logically expect them to perform very differently due to the limitations of a linear approach.

7.1 NeuralUCB

Dataset Although we tested on various datasets we found the costs to train on non-toy datasets prohibitive for the neural version of the algorithms. As a result, we mainly tested on toy datasets and provide results only for the famous crescent moons dataset an implementation of which can be found here. We also provide results on MNIST dataset in section (A.2).

Neural Network For our experiments we used a 3 layer neural network with 25 hidden units in each layer and an output dimension of 2. In addition to the description of the algorithm provided above, we also added a dropout layer after each regular layer of the neural network with $p = 0.3$.

Optimizer We attempted to use SGD, Adam, and SGD with momentum with various different hyper parameters. Generally, We found that Adam was the best performing optimizer followed by SGD with momentum. We used a learning rate of 0.001,a momentum of 0.9, and the default hyperparemeters in [3].

Additional Parameters We sampled ten datapoints at time for both the online and active version of the problem. Neural networks were trained after 100 queries (or timesteps) and trained for two epochs. Every 4 epochs we reset the matrices to their orignal states. We rotate the the data the algorithm sees every two time steps. The values reported for losses and accuracy are computed on a heldout set of the data of 1000 datapoints.

7.1.1 Experiment 1: Effectiveness of a linear classifier on learned features

To determine the quality of our embeddings we used an SVM to classify the learned features and observe these values per iteration. We report the results both for SGD with momentum and Adam. Additionally, we also report the performance of NeuralUCB using the unormalized version of cosine similarity. All graphs contain the mean and a 95% confidence interval computed over 10 runs of the algorithm. The results are shown in Figures (1) and (2)

Clearly we see that on normalized version the performance of an active approach yields better results that non-active but when normalized features are used we see that this effec almost disappears completely. Moreover, it seems that using Adam reduces the different between algorithms.

7.2 Experiment 2: Comparison of L2 loss through time

Using the exact setup as in Experiment 1, we report the results of the L2 loss through time for Active-NeuralUCB.

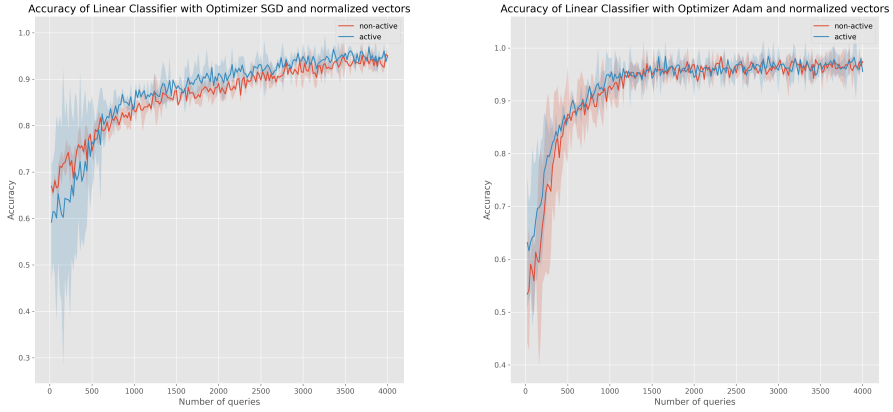

Figure 2: Performance SVM on normalized learned features Adam vs SGD

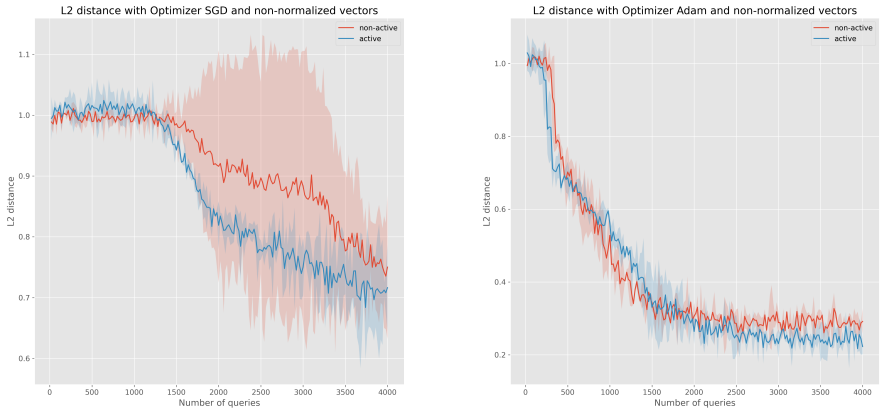

Figure 3: Performance SVM on non-normalized learned features Adam vs SGD

As before, we observe that adam erases to a large extent the differences between algorithms and that the performance of the active approach depends heavily on the particular optimizer used.

7.3 Experiment 3: Regret of OnSim-NeuralUCB

We observe the performance of OnSim-NeuralUCB by compring the regret of the algorithm with the regret of an epsilon-greedy neural network with the same architecture and hyperparameters. We report the results only for Adam as it had a significantly better performance, but provide the comparison both for the normalized and unormalized versions. Like in the previous experiment all graphs contain the mean and a 95% confidence interval. We also compare various values for the exploration parameter.

For this dataset we observe that there is no sigficant difference between using a simpler epsilon greedy approach and our neural-ucb based approach.

7.4 LinearUCB

Dataset: To test the linear version of our algorithm we created a blobs dataset. Every point is drawn from the following process. Assume that $x_i' \sim \mathcal{N}((0,0)^\top, I_2)$ and $p_i \sim \text{Bernoulli}(p)$ where p is some parameter. Then the data point x_i is drawn from the follwing process

$$x_i = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} x_i' - (1-p_i)\begin{bmatrix} 1 \\ 1 \end{bmatrix} + p_i \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Figure 4: Performance SVM on normalized learned features Adam vs SGD

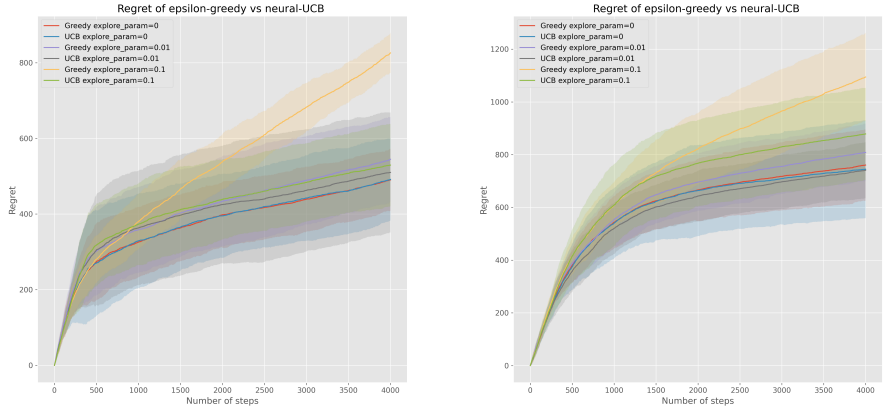

Figure 5: Performance OnSim-NeuralUCB on non-normalized and normalized features

which corresponds to two blobs one placed on the left axis and one on the right axis. We say that two points are similar if they share the same value of p. We refer to this dataset as the Blobs dataset.

7.4.1 Experiment 1: Effectiveness of active learning balanced and unbalanced datset

We test the effectiveness of active learning on the balanced and unbalanced datasets. For the unbalanced instance we use $p = 0.1$ but provide the results of an l2 loss assuming that the real distribution of the data is balanced. This is done to simulate a situation in which our collected unlabeled data is based and to show the improvement in robustness of the algorithm. The plots in (6) and (7) contain the mean and the 95% confidence interval for the regret over 30 runs of each algorithm.

We observe that the performance of our active algorithm is slightly worse than the baseline in the balanced data regime but it is significantly better when we have unbalanced data. More importantly, we see that it is able to mantain low variance in an unbalanced regime while the non-active approach doesn't enjoy this property.

7.4.2 Experiment 2: Regret of active similarity with regime shifts

To show that our approach leads to enhanced robustness we assume that we find ourselves in a situation where the might be regime shifts in the data in the sense that the underlying distribution at which the data is provided to the algorithm changes.

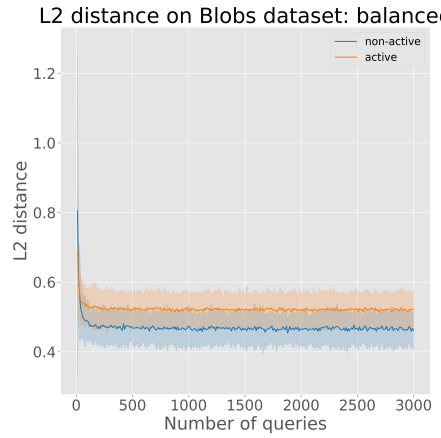

Figure 6: Balanced dataset

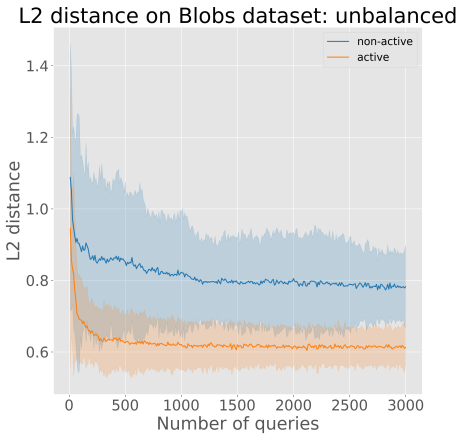

Figure 7: Unbalanced dataset

In particular, in the situation below we assume that initially the data is provided to us where there is an equal changce of getting a point from the left blob and right blob but there at every time-step there is a $1/3000$ chance that we will have a regime shift where the data is now sampled such that $p = 0.1$. Plots (8) and (9) indicate the mean regret of the algorithm at the end and the error bars depict the the 2.5 and 97.5 quantiles observed from 30 runs.

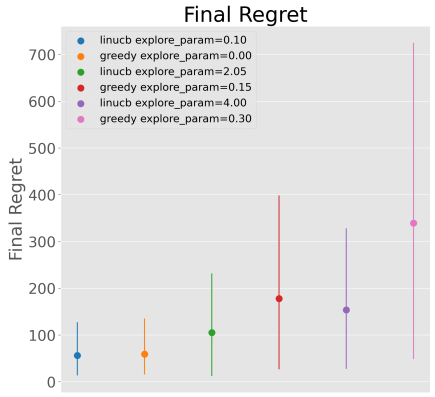

Figure 8: Regret without regime change

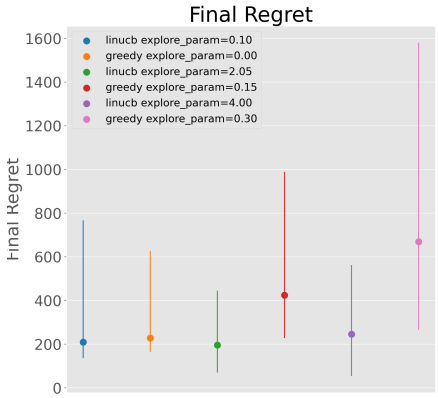

Figure 9: Regret with regime change

When there is no regime change we see that a purely greedy approach has a similar performance to an active approach with exploration parameter of 0.1. However, when we allow for regime changes we see that a linucb based approach with exploration parameter 2.05 beats all greedy algorithms, not only interms of a lower mean regret but also in terms of a significantly lower 97.5 quantile and lower 2.5 quantile.

8 Discussion of results

The results of the experiments show a mixed picture. On the one hand the results for Active-LinUCB and OnSim-LinUCB show the benefits that hour approach has compared to a naive use of the data or an epsilon greedy baseline. This is reasonable as the matrix used in the algorithm naturally favors a diversity of points which allows the algorithm to explore the two blobs more efficiently.

However, we observe that for our neural algorithms the performance varies significantly depending on the particular optimization strategy used and whether we use normalized or non-normalized features.

This situation is aggravated further by $O(p^2)$ memory requirements (due to the matrix computation) of the algorithm and the $O(Tp^2)$ time requirements (due to the computation of the optimism). All of the above suggests that even if there is an advantage the algorithms are not feasible for practical settings.

Finally, it is important to note that even if the linear approach works, the reason why it performs so well is because it implicitely uses some notion of diversity implicit in the eucledean space. This is limiting and contradictory in that precisely such a notion of eucledean space is what we are trying to learn.

An interesting alternative which would solve the aformentioned issues realted to the complexity of the algorithms and the issues pertaining to the notion of space in the linear case is to use only a shallow notion of UCB as proposed in [5] or [4]. This way we can avoid the expensive matrix computations and optimize based on a diversity of vectors with respect to the notion of space used by the neural network, which is constantly changing.

9 Conclusion and Future Work

In this work we studied a variety of approaches to solving the active learning and online versions of similarity learning using UCB based appraches. Although our work suggests that the methods proposed are not enough for practical use, we believe that the tests conducted suggests that there is value in exploring these approaches, especially in situations where there might be shifts in the underlying distribution of the data and the amount of similar pairs that we get. As a future direction forward we propose to work with shallow methods of exploration to avoid expensive matrix computations while retaining the flexibility of neural based appraches.

References

[1] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 208–214, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[2] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. 2018.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[4] Carlos Riquelme, George Tucker, and Jasper Snoek. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling, 2018.

[5] Pan Xu, Zheng Wen, Handong Zhao, and Quanquan Gu. Neural contextual bandits with deep representation and shallow exploration. *CoRR*, abs/2012.01780, 2020.

[6] Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural contextual bandits with ucb-based exploration, 2019.

A Appendix

A.1 Algorithms omitted from main text

Algorithm 3: Train

Input : ϵ learning rate, E number of epochs, dataset $\{(x_i^{k_i}, y_i^{k_i}, r_i)\}_{i=1}^t$, b_s batch size

optimizer $\leftarrow$ SGD(ϵ, θ);
for $j = 1, \ldots, E$ **do**
  $\mathcal{D} \leftarrow \{(x_1^{k_1}, y_1^{k_1}, r_1), \ldots, (x_t^{k_t}, y_t^{k_t}, r_t)$;
  while $\mathcal{D}$ *is not empty* **do**
    Sample minibatch M of size b_s from the training examples;
    $l = \frac{1}{M} \sum_{(x_i^{k_i}, y_i^{k_i}, r_i) \in M} (\phi(x_i^{k_i}, y_i^{k_i}) - r_i)^2$;
    $\theta \leftarrow$ update using optimizer with loss l;
    Remove M from $\mathcal{D}$;
  end
end

Algorithm 4: OnSim-LinUCB

Input : Rounds T and exploration parameter α

$A \leftarrow I_{n^2}$;
$b \leftarrow 0_{n^2}$;
for $t \in [T]$ **do**
  $\theta_t \leftarrow A^{-1}b$ Observe K pairs of vectors $x^k \in \mathbb{R}^n$, $y^k \in \mathbb{R}^n$;
  Create $z_{t,k} = (x_1^k, y_1^k, x_1^k, y_1^k, \ldots, x_n^k y_{n-1}^k, x_n^k, y_n^k)$;
  for $k \in [K]$ **do**
    $p_{t,a} \leftarrow \theta_t^\top z_{t,k} + \alpha \sqrt{z_{t,k}^\top A^{-1} z_{t,a}}$
  end
  Choose action $k_t = \text{argmax}_a p_{t,a}$ with ties broken arbitrarily;
  Observe payoff $r_t \in \{-1, 1\}$;
  $A \leftarrow A + z_{t,k_t} z_{t,k_t}^\top$;
  $b \leftarrow z_{t,k_t} r_t$;
end

A.2 Additonal experiments

This section contains additional experiments we ran but omit from the main text due to space considerations.

Algorithm 5: Active-NeuralUCB

Input : Queries T, exploration parameter α, τ_r frequency of resets, τ_T frequency of training, E epochs for training, b_s batch size for training, ϵ learning rate.

$A \leftarrow I_p$;
for $t \in [T]$ **do**
  Sample K pairs of vectors $x_t^k \in \mathbb{R}^n$, $y_t^k \in \mathbb{R}^n$ from dataset D;
  for $k \in [K]$ **do**
    $p_{t,k} \leftarrow \alpha\sqrt{(\nabla_\theta \phi(x_t^k, y_t^k))^\top A^{-1} \nabla_\theta \phi(x_t^k, y_t^k)}$;
  end
  Choose pair $k_t = \text{argmax}_k p_{t,k}$ with ties broken arbitrarily;
  Query label $r_t \in \{-1, 1\}$;
  if $t \mod \tau_r = 0$ **then**
    $\theta \leftarrow \text{Train}(\epsilon, E, \{(x_i^{k_i}, y_i^{k_i}, r_i)\}_{i=1}^t, b_s)$;
  end
  if $t \mod \tau_T = 0$ **then**
    $A \leftarrow I_p$
  end
  $A \leftarrow A + \nabla_\theta \phi(x_t^k, y_t^k)(\nabla_\theta \phi(x_t^k, y_t^k))^\top$;
end