
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 $\phi : \mathbb{R}^{2n} \rightarrow \{-1, 1\}$ 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

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

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

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 $\phi : \mathbb{R}^{2n} \rightarrow \{-1, 1\}$ which the learner is trying to learn. The learner

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 maintains 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 $\hat{\phi}$ and 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, depending on whether the bandit algorithm they are based on and the problem 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 straightforward 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 $\phi(x, y) = x^\top A y$. This is reasonable if we consider that

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

which 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 5. 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 5 assumes that similarity is the result of a dot product 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 [4] to our circumstances.

Formally, we assume that

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

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

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

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 $(W_L, b_L, \dots, W_1, b_1)$. 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 4 but it simply describes the normal process of training a neural network with MSE loss.

Our algorithm is very similar to that proposed in [4] 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 matrix A after every $t \bmod \tau_T = 0$ 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 [4] 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 version and unnormalized NeuralUCB. We provide some comparisons of this version 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.

```
1  $A \leftarrow I_p$ ;
2 for  $t \in [T]$  do
3   Observe  $K$  pairs of vectors  $x_t^k \in \mathbb{R}^n, y_t^k \in \mathbb{R}^k$ ;
4   for  $k \in [K]$  do
5      $p_{t,k} \leftarrow \phi(x, y) + \alpha \sqrt{(\nabla_\theta \phi(x_t^k, y_t^k))^\top A^{-1} \nabla_\theta \phi(x_t^k, y_t^k)}$ ;
6   end
7   Choose pair  $k_t = \text{argmax}_k p_{t,k}$  with ties broken arbitrarily;
8   Observe payoff  $r_t \in \{-1, 1\}$ ;
9   if  $t \bmod \tau_r = 0$  then
10     $\theta \leftarrow \text{Train}(\epsilon, E, \{(x_i^{k_i}, y_i^{k_i}, r_i)\}_{i=1}^t, b_s)$ ;
11   end
12   if  $t \bmod \tau_T = 0$  then
13     $A \leftarrow I_p$ 
14   end
15    $A \leftarrow A + \nabla_\theta \phi(x_t^k, y_t^k) (\nabla_\theta \phi(x_t^k, y_t^k))^\top$ ;
16 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-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.

```
1  $A \leftarrow I_p$ ;
2 for  $t \in [T]$  do
3   Sample  $K$  pairs of vectors  $x_t^k \in \mathbb{R}^n, y_t^k \in \mathbb{R}^n$  from dataset  $D$ ;
4   for  $k \in [K]$  do
5      $p_{t,k} \leftarrow \alpha \sqrt{(\nabla_\theta \phi(x_t^k, y_t^k))^\top A^{-1} \nabla_\theta \phi(x_t^k, y_t^k)}$ ;
6   end
7   Choose pair  $k_t = \text{argmax}_k p_{t,k}$  with ties broken arbitrarily;
8   Query label  $r_t \in \{-1, 1\}$ ;
9   if  $t \bmod \tau_r = 0$  then
10     $\theta \leftarrow \text{Train}(\epsilon, E, \{(x_i^{k_i}, y_i^{k_i}, r_i)\}_{i=1}^t, b_s)$ ;
11   end
12   if  $t \bmod \tau_T = 0$  then
13     $A \leftarrow I_p$ 
14   end
15    $A \leftarrow A + \nabla_\theta \phi(x_t^k, y_t^k) (\nabla_\theta \phi(x_t^k, y_t^k))^\top$ ;
16 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. Algorithms 2

and 3 describe these processes formally. In Algorithm 2 the most important change is line 5 and in algorithm 3 the most important change is line 7.

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 hyperparameters 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 original states. We rotate 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 unnormalized 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)

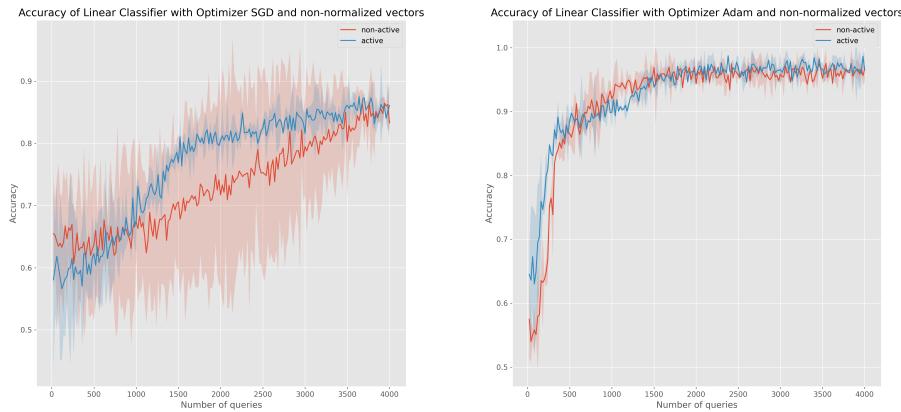


Figure 1: Performance SVM on non-normalized learned features Adam vs SGD

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

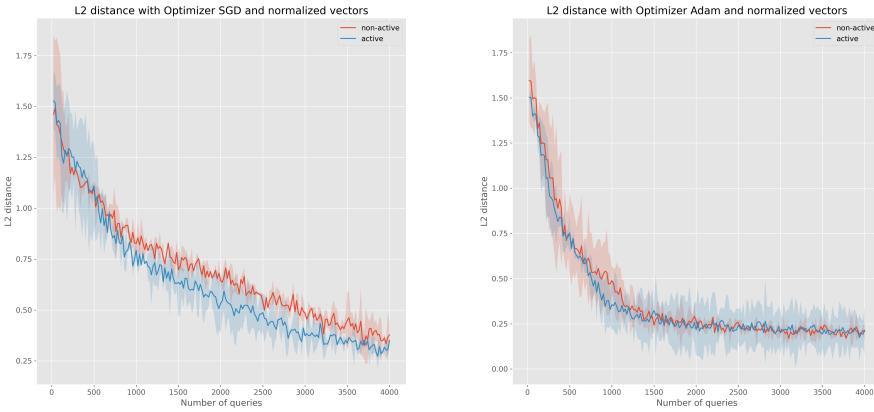


Figure 4: Performance SVM on normalized learned features Adam vs SGD

7.3 Experiment 3: Regret of OnSim-NeuralUCB

We observe the performance of OnSim-NeuralUCB by comprising 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 Adam with normalized and unnormalized 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.²

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 following process

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

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.

Algorithm 3: Active-LinUCB

Input: Rounds T and exploration parameter α

```

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

```

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

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 biased and to show the improvement in robustness of the algorithm. The plots contain the mean and the 95% confidence interval for the regret over 30 runs of each algorithm.

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 there might be regime shifts in the data in the sense that the underlying distribution at which the data is provided to the algorithm changes.

In particular, in the situation below we assume that initially the data is provided to us where there is an equal chance 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 (7) and (8) 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.

²The appendix contains a more thorough graph with even more values.

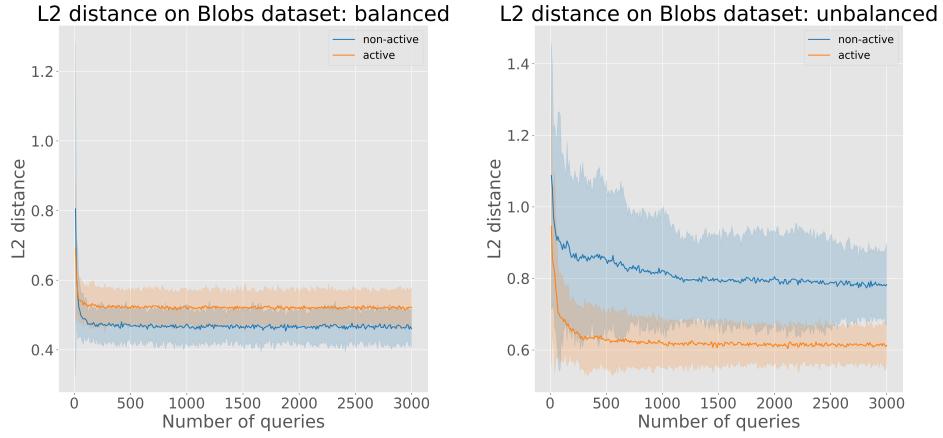


Figure 5: Active vs non-active balanced datasets

Figure 6: Active vs non active balanced datasets

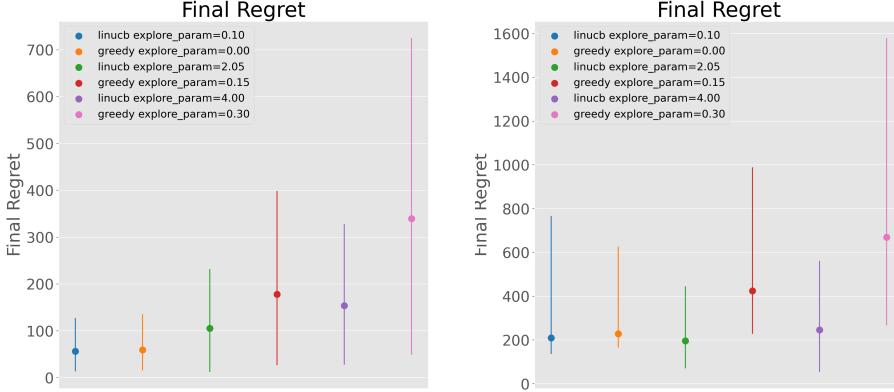


Figure 7: Regret without regime change

Figure 8: Regret with regime change

8 Conclusion

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] 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 4: 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

```

1 optimizer ← SGD( $\epsilon$ ,  $\theta$ );
2 for  $j = 1, \dots, E$  do
3    $\mathcal{D} \leftarrow \{(x_1^{k_1}, y_1^{k_1}, r_1), \dots, (x_t^{k_t}, y_t^{k_t}, r_t)\}$ ;
4   while  $\mathcal{D}$  is not empty do
5     Sample minibatch  $M$  of size  $b_s$  from the training examples;
6      $l = \frac{1}{M} \sum_{(x_i^{k_i}, y_i^{k_i}, r_i) \in M} (\phi(x_i^{k_i}, y_i^{k_i}) - r_i)^2$ ;
7      $\theta \leftarrow$  update using optimizer with loss  $l$ ;
8     Remove  $M$  from  $\mathcal{D}$ ;
9   end
10 end
```

Algorithm 5: OnSim-LinUCB

Input : Rounds T and exploration parameter α

```

1  $A \leftarrow I_{n^2}$ ;
2  $b \leftarrow 0_{n^2}$ ;
3 for  $t \in [T]$  do
4    $\theta_t \leftarrow A^{-1}b$  Observe  $K$  pairs of vectors  $x^k \in \mathbb{R}^n$ ,  $y^k \in \mathbb{R}^n$ ;
5   Create  $z_{t,k} = (x_1^k, y_1^k, x_2^k, y_2^k, \dots, x_n^k, y_{n-1}^k, x_n^k, y_n^k)$ ;
6   for  $k \in [K]$  do
7      $p_{t,a} \leftarrow \theta_t^\top z_{t,k} + \alpha \sqrt{z_{t,k}^\top A^{-1} z_{t,a}}$ 
8   end
9   Choose action  $k_t = \operatorname{argmax}_a p_{t,a}$  with ties broken arbitrarily;
10  Observe payoff  $r_t \in \{-1, 1\}$ ;
11   $A \leftarrow A + z_{t,k_t} z_{t,k_t}^\top$ ;
12   $b \leftarrow z_{t,k_t} r_t$ ;
13 end
```

A.2 Additonal experiments

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