Hash & Bayesian

Fast Pairwise Query Selection for Large-Scale Active Learning to Rank ICDM 2013

Scalable Bayesian Optimization Using Deep Neural Networks ICML 2015

Hyperparameter Tuning for Big Data using Bayesian Optimisation ICPR 2016

2017.7.9

CONTENTS

01 Review

02 LTR

03 DNGO

04 parallel

05 Review

01 Review

A sample from a Gaussian process is a function as $f(x) \sim N(\mu(x), k(x, x'))$

$$k(x, x') = \exp(-\frac{1}{2}||x - x'||^2) \qquad k(x, x') = \exp(-\frac{1}{2}||x - x'||^2)$$

Let us denote $y_{1:p} = f(x_{1:p})$ as the function values corresponding to the data points $x_{1:p}$

$$K = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_p) \\ \vdots & \ddots & \vdots \\ k(x_p, x_1) & \cdots & k(x_p, x_p) \end{bmatrix} \qquad \begin{bmatrix} y_{1:p} \\ y_{P+1} \end{bmatrix} \sim N(0, \begin{bmatrix} K & k \\ k^T & k(x_{p+1}, x_{p+1}) \end{bmatrix})$$

$$k = [k(x_1, x_{P+1}), k(x_2, x_{P+1}), \ldots, k(x_p, x_{P+1})]$$

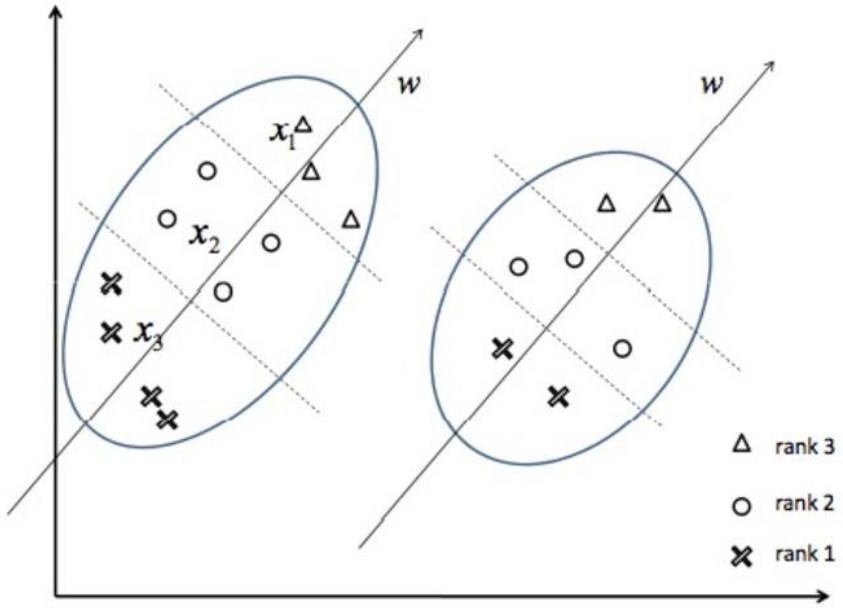
01 Review

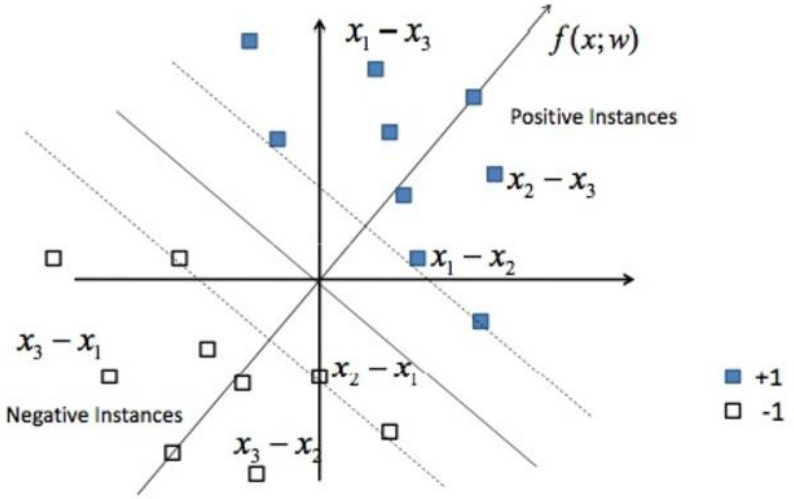
$$P(y_{P+1}|x_{1:p}, y_{1:p}) \sim N(\mu_p(x_{p+1}), \sigma_p^2(x_{p+1}))$$

Algorithm 1 The Generic Bayesian Optimisation Algorithm

1: **Input:** The initial observation $D \equiv \{\mathbf{x}_{1:p}, \mathbf{y}_{1:p}\}$.
2: **Output**: $\{\mathbf{x}_n^*, \mathbf{y}_n^*\}_{n=1}^T$
3: **for** $n = 1, 2, ..T$
4: Find $\mathbf{x}_n^* = \arg\max_{\mathbf{x}} \mathbb{E}(I(\mathbf{x})|GP(D))$ of (11).
5: Evaluate the objective function: $\mathbf{y}_n^* = f(\mathbf{x}_n^*)$.
6: Augment the observation set $D = D \cup (\mathbf{x}_n^*, \mathbf{y}_n^*)$.
7: **end for**

02 LTR





$$\min \frac{1}{2} w^T w + C \sum_{ij} \varepsilon_{ij} \qquad \text{s.t.} \quad w^T (x_i - x_j) \geq 1 - \varepsilon_{ij}, \quad if\ (i,j) \in S;\ \varepsilon_{ij} \geq 0.$$

Enforces the ranking gaps (difference between ranking scores) on ordered pairs.

ε is a non-negative slack variable used to "soften" the constraints;

02 LTR

The overall aim is to find pairs of data points that are relevant to the query but the ranking function is most uncertain about.

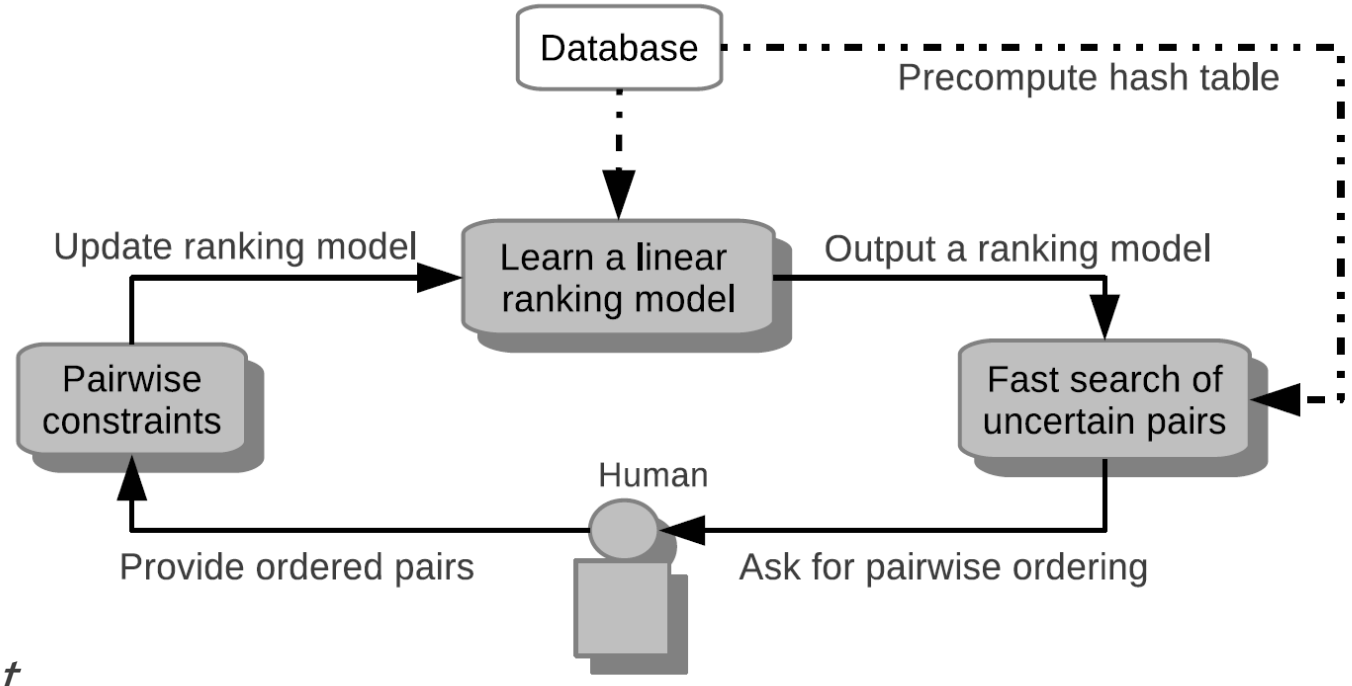


Fig. 2. The cycle of fast active learning to rank.

The first part involves using a *point-to-point* hash function to skip data points that are clearly irrelevant to the query.

We propose an efficient *point-to-hyperplane* hash algorithm, which will be used to find the most uncertain pairs of points.

02 LTR

Point-to-Point $\chi = \{x_1, x_2, \dots, x_n\}$ $x_i \in \mathbb{R}^d$ $b_x = \text{argmax} \frac{b^T x}{||b||\,||x||}$ s.t. $b \in \{0,1\}^d$

$||b||_H = m$, b' that lies at a Hamming radius r from $\left[\sqrt{\frac{m-r}{m}}, \sqrt{\frac{m}{m+r}}\right]$

Point-to-Hyperplane

The hyperplane hashing problem - looking for a b that is most perpendicular to w - is equivalent to minimizing the cosine of the angle between w and b

$$b_w = \text{argmax} \frac{b^T w}{||b||\,||w||} \quad \text{s.t. } b \in \{0,1\}^d$$

$x = x^+ - x^-$ $w = w^+ - w^-$ $b(x) = \begin{pmatrix} b_{x^+} \\ b_{x^-} \end{pmatrix}$

02 LTR

Through optimizing a particular gain/loss measure, we learn a linear ranking function *w*, which can be interpreted as a hyperplane normal vector passing through the origin.

$$S = \{(i, j)\}$$

The retrieval results are represented in a descending order with respect to the ranking scores, which are calculated for each data point separately using a hyperplane *w*, as show in the equation: $\tau(x) = w^T x$

Step 1 - Relevance Hashing

As to construct the first layer hash lookup table, we map each database point $x \in R^d$ to its angle-wise nearest binary code ($\in B^{2d}$)

$$H_{rel}(x) = \begin{pmatrix} \text{argmax} \dfrac{b^T x^+}{||b|| ||x^+||} \\ \text{argmax} \dfrac{b^T x^-}{||b|| ||x^-||} \end{pmatrix} \quad b \in \{0,1\}^d$$

01 LTR

Step 2 - Uncertainty Hashing

For each of the first layer hash buckets, we construct a second layer hash table by mapping the vector of each pair $(x_i - x_j)(\forall(i,j) \in \{(i,j) \in |H_{rel}(x_i) = H_{rel}(x_j)\})$ to its angle-wise nearest binary approximation ($\in B_{2d}$) using the same hash function of relevance hashing H_{rel}

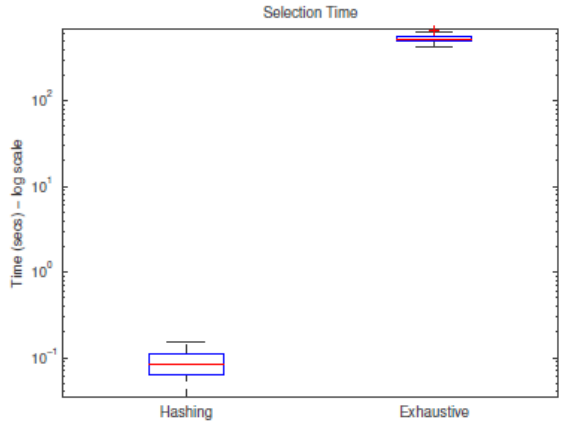
$$H_{unc} = \begin{pmatrix} \text{argmax} \dfrac{b^T(x_i - x_j)^+}{||b||||(x_i - x_j)^+||} \\ \text{argmax} \dfrac{b^T(x_i - x_j)^-}{||b||||(x_i - x_j)^-||} \end{pmatrix}$$

$$H_{rel}(x) = \begin{pmatrix} \text{argmax} \dfrac{b^T w^+}{||b||||w^+||} \\ \text{argmax} \dfrac{b^T w^-}{||b||||w^-||} \end{pmatrix} \text{ or } \begin{pmatrix} \text{argmax} \dfrac{b^T w^-}{||b||||w^-|} \\ \text{argmax} \dfrac{b^T w^+}{||b||||w^+||} \end{pmatrix}$$

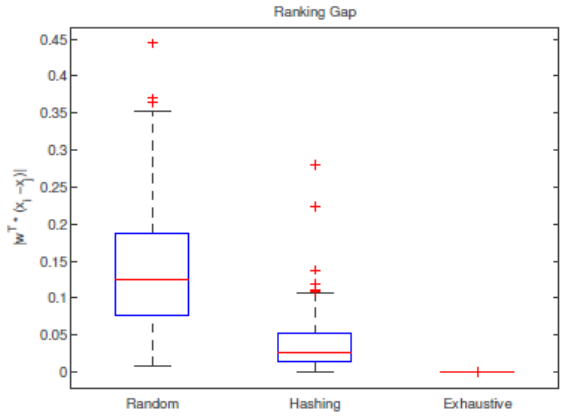
Given a ranking hyperplane *w*, the uncertain pairs are those whose vectors are close to perpendicular to the hyperplane normal vector *w*.

01 LTR

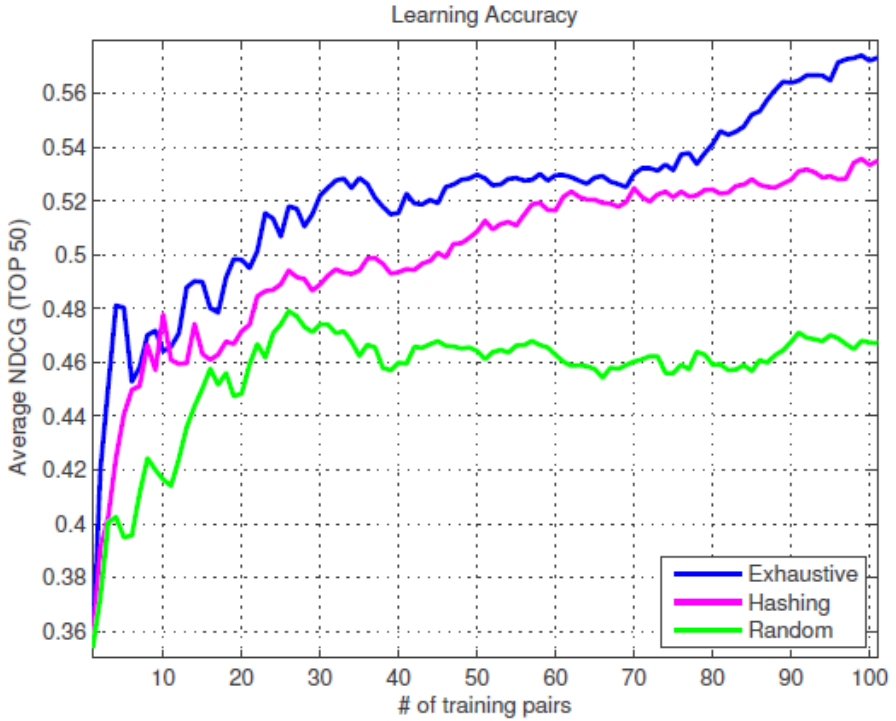
Results



(a) Selection Time



(b) Ranking Gap



(a) Average NDCG (Top 50)

02 DNGO Adaptive Basis Regression with Deep Neural Networks

A key limitation of GP-based Bayesian optimization is that the computational cost of the technique scales cubically in the number of observations. We show that performing adaptive basis function regression with a neural network as the parametric form performs competitively with state-of-the-art GP-based approaches, but scales linearly with the number of data rather than cubically.

We take a pragmatic approach and add a Bayesian linear regressor to the last hidden layer of a deep neural network, marginalizing only the output weights of the net while using a point estimate for the remaining parameters. This results in *adaptive basis regression*.

First of all, without loss of generality and assuming compact support for each input dimension, we scale the input space to the unit hypercube.

$$\boldsymbol{\Phi}(\cdot) = [\phi_1(\cdot), \phi_2(\cdot), \dots, \phi_n(\cdot)]$$

denotes the vector of outputs from the last hidden layer of the network, We take these to be our set of basis functions. In addition, define $\boldsymbol{\Phi}$ to be the design matrix arising from the data and this basis, where $\boldsymbol{\Phi}_{nd} = \phi_d(x_n)$ is the output design matrix, and y the stacked target vector.

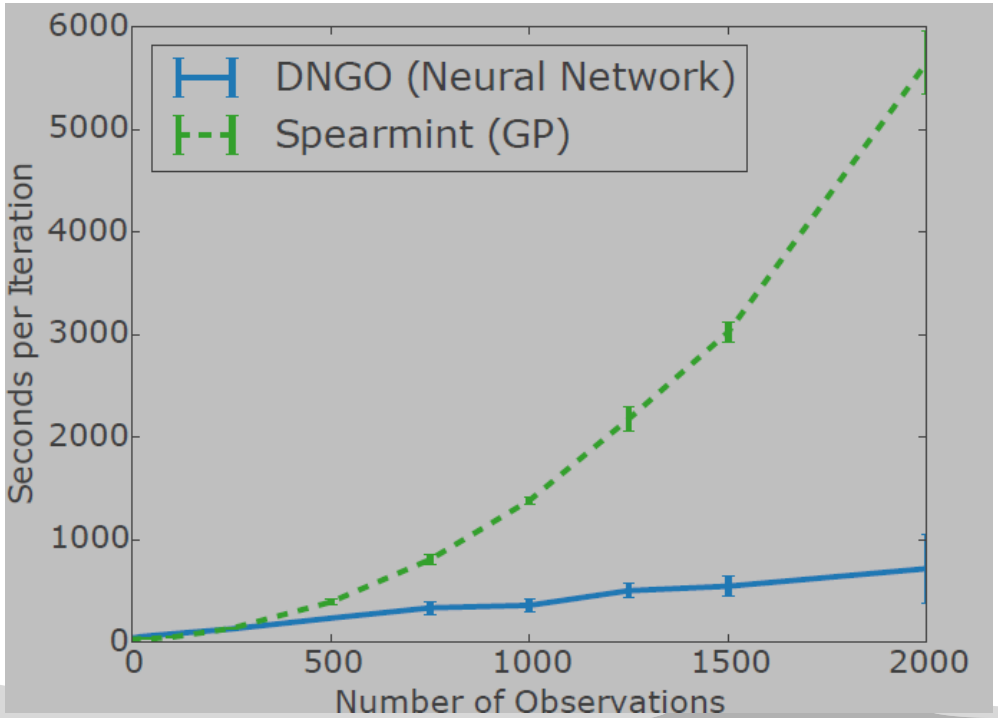
These basis functions are parameterized via the weights and biases of the deep neural network, and these parameters are trained via backpropagation and stochastic gradient descent with momentum. In this training phase, a linear output layer is also fit. This procedure can be viewed as a maximum a posteriori (MAP) estimate of all parameters in the network. Once this "basis function neural network" has been trained, we replace the MAP-parameterized output layer with a Bayesian linear regressor that captures uncertainty in the weights.

The predictive mean $\mu(x;\theta)$and variance $\sigma^2(x;\theta)$ of the model are $\mu(x;D,\theta)=\boldsymbol{m}^T\boldsymbol{\phi}(x)+\eta(x)$ $\sigma^2(x;\boldsymbol{D},\theta)=\boldsymbol{\phi}(x)^T\mathbf{K}^{-1}\boldsymbol{\phi}(x)+\frac{1}{\beta}$ *where* $\boldsymbol{m}=\beta\mathbf{K}^{-1}\boldsymbol{\Phi}^T\tilde{\boldsymbol{y}}\in\mathbb{R}^D$ $\mathbf{K}=\beta\boldsymbol{\Phi}^T\boldsymbol{\Phi}+\mathbf{I}\alpha\in R^{D\times D}$

Here, $\eta(x)$is a prior mean function, and $\tilde{\boldsymbol{y}}=\boldsymbol{y}-\eta(x)$In addition, α $\beta$are regression model hyperparameters. We integrate out α β using slice over the marginal likelihood, which is given by $\log p(\boldsymbol{y}|\mathbf{X},\alpha,\beta)=\frac{D}{2}\log\alpha+\frac{N}{2}log\beta-\frac{N}{2}\log(2\pi)-\frac{\beta}{2}\left||\tilde{\boldsymbol{y}}-\boldsymbol{\Phi}\mathbf{m}|\right|^2-\frac{\alpha}{2}\mathbf{m}^T\mathbf{m}-\frac{\mathbf{1}}{\mathbf{2}}\log|\mathbf{K}|$

02 DNGO

It is clear that the computational bottleneck of this procedure is the inversion of **K**. However, note that the size of this matrix grows with the output dimensionality D, rather than the number of observations N as in the GP case. This allows us to scale to significantly more observations than with the GP as demonstrated in Figure1.



Dashed lines correspond to weights that are marginalized.

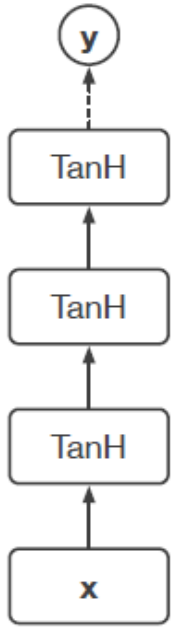
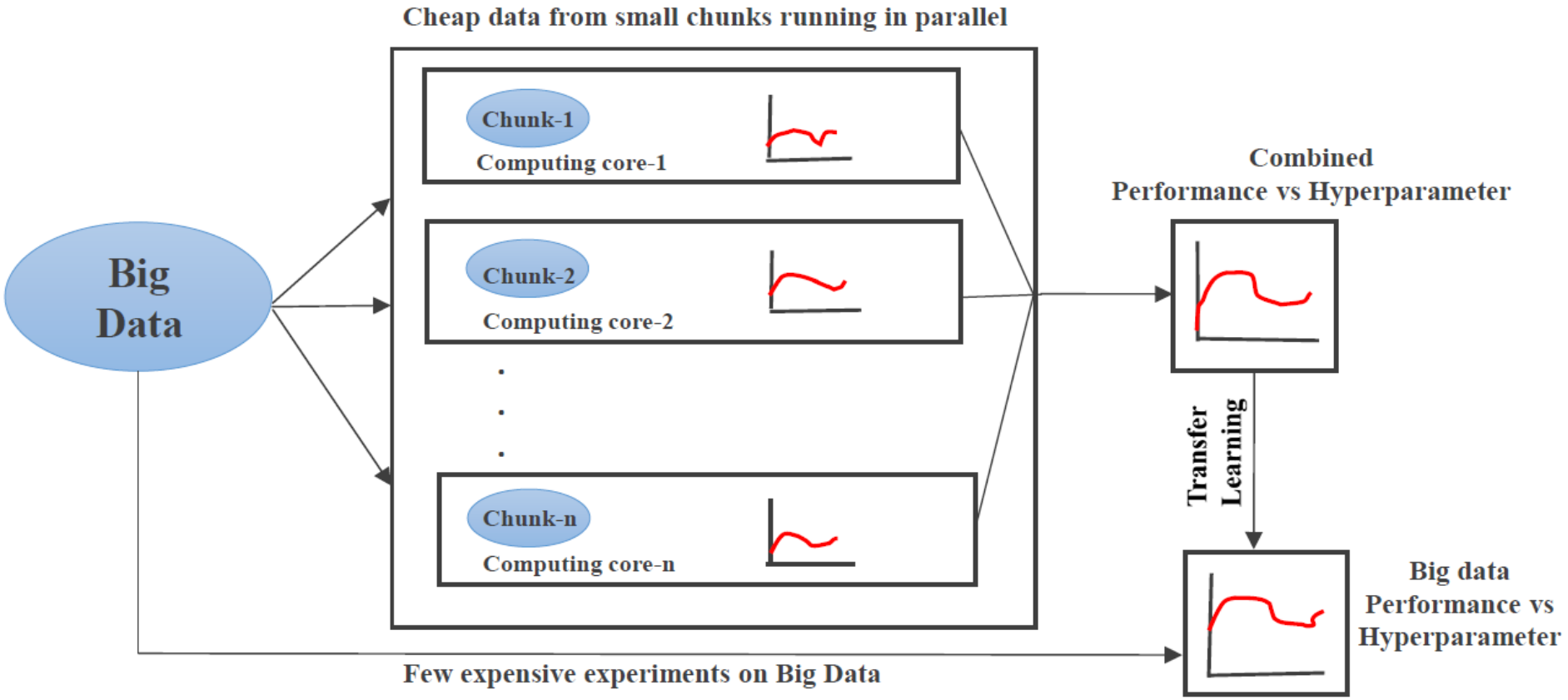


Figure 1. The time per suggested experiment for our method compared to the state-of-the-art GP based approach from Snoek et al. (2014) on the six dimensional Hartmann function. We ran each algorithm on the same 32 core system with 80GB of RAM five times and plot the mean and standard deviation.

03 parallel



Divide the big data into chunks and generate hyperparameter configurations for the chunks using the standard Bayesian optimization. We utilize this information from the chunks for hyperparameter tuning on big data using a transfer learning setting.

03 parallel

$$\mu_{combined} = 1/m \sum_{i=1}^{m} \mu_i \qquad \sigma^2_{combined} = 1/m \sum_{i=1}^{m} \sigma^2{}_i$$

The performance of the model as function of hyperparameters on the big data, which is denoted as $f^b(\cdot)$,The Bayesian optimization uses the $\mathrm{GP}_{\mathrm{combined}}$, as the posterior model for tuning the hyperparameters on the big data. We model the observations from the chunks as a noisy observations on the big data, i.e, as a noisy measurement of $f^b(\cdot)$, $y_i^c = f^b(x_i^c) + \alpha_i^c$, $\forall i = 1, \ldots, N_c$ $\alpha_i^c \sim N(0, \sigma_c^2)$ We estimate α^c between the response surfaces of the hyperparameters on the big data and chunks in a Bayesian setting.

The kernel matrix for each of the chunk GPs, $\mathbf{K}_c$, is computed after combining the observations from chunks and the big data. The kernel matrix $\mathbf{K}c$ is then updated by adding different noise variance levels as $\mathbf{K}_c = \mathbf{K}_c + \begin{bmatrix} \sigma_c^2 \mathbf{I}_{N_c \times N_c} & \mathbf{0} \\ \mathbf{0}^T & \sigma_b^2 \mathbf{I}_{N_b \times b} \end{bmatrix}$

03 parallel

Algorithm 2 Proposed Algorithm.

1: **Input**:
Initial Observations:$\{\mathbf{x}^b, \mathbf{y}^b\}$,
2: **Initial Settings:**
Generate hyperparameter settings for Chunks : $GP^c_{1:m}$
Build the combined GP $GP^{combined}$ using (12) and (13).
3: **Output**:
$\{\mathbf{x}^*_n, \mathbf{y}^*_n\}^T_{n=1}$.
4: **for** $n = 1, 2, ..T$
5: Find $\mathbf{x}^*_n = \arg\max_{\mathbf{x}} \mathbb{E}(I(\mathbf{x})|GP^{combined})$ of Eq. (11).
6: Evaluate $\mathbf{y}^*_n = f^b(\mathbf{x}^*_n)$
8: Compute $\mathbf{y}^c_n = f^c(\mathbf{x}^*_n)$ for different data chunks.
9: Update $a^{1:m}_n$,$b^{1:m}_n$ and $\alpha_{1:m}$ using Eq. (18), (19), 20)
10: Compute $\mathbf{K}_c$ for each GP^c by augmenting$\{\mathbf{x}^*_n, \mathbf{y}^*_n\}$.
11: Update each of the chunk GPs GP^c using Eq. (15).
11: Update the combined GP $GP^{combined}$.
12: **end for**

$$\mu_{combined} = 1/m \sum_{i=1}^{m} \mu_i$$

$$\sigma^2_{combined} = 1/m \sum_{i=1}^{m} \sigma^2{}_i$$

$$(a^i_n)_{\ i=1:m} = a_0 + n/2$$

$$(b^i_n)_{\ i=1:m} = b_0 + \frac{\sum_{j=1}^{n}(y^b_{ij} - y^c_{ij})^2}{2}$$

$$(\sigma^2_c)_{\ i=1:m} = \frac{(b^i_n)_{\ i=1:m}}{(b^i_n)_{\ i=1:m} + 1}$$

03 parallel

Estimation of Noise Variance (σ_c^2)

We estimate the variance σ_c^2 of α_c using a Bayesian framework. We place an inverse gamma distribution with parameters a_0 and b_0 as the prior distribution

$$\sigma_c^2 \sim \text{InvGamma}(a_0, b_0)$$

We start with a wide prior and then update the posterior from the observation of output value of the big data (y_b) and the data chunks (y_c) for the same hyperparameter setting. Since the inverse gamma is a conjugate prior to the variance, the posterior is also an inverse gamma distribution with updated parameters a_n and b_n as

$$P(\sigma_c^2 / \{y_{ij}^b - y_{ij}^c\}_{i=1}^N) \sim InvGamma(a_n, b_n)$$

03 parallel

$$(a_n^i)_{i=1:m} = a_0 + n/2 \qquad (b_n^i)_{i=1:m} = b_0 + \frac{\sum_{j=1}^{n}(y_{ij}^b - y_{ij}^c)^2}{2}$$

n refers to the number of trials and i denotes the particular chunk.

We use the mode of the posterior distribution as the value of noise variance and it is given as

$$(\sigma_c^2)_{i=1:m} = \frac{(b_n^i)_{i=1:m}}{(b_n^i)_{i=1:m}+1}$$

03 parallel

results

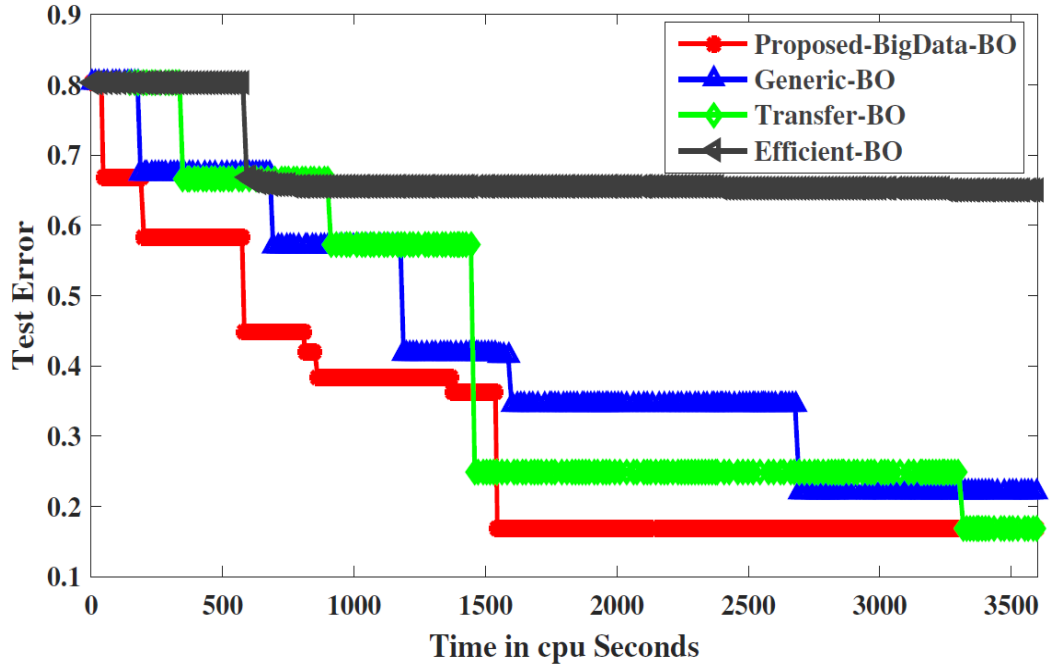


Figure 2. Tuning the hyperparameters of Deep Neural Networks on MNIST dataset.

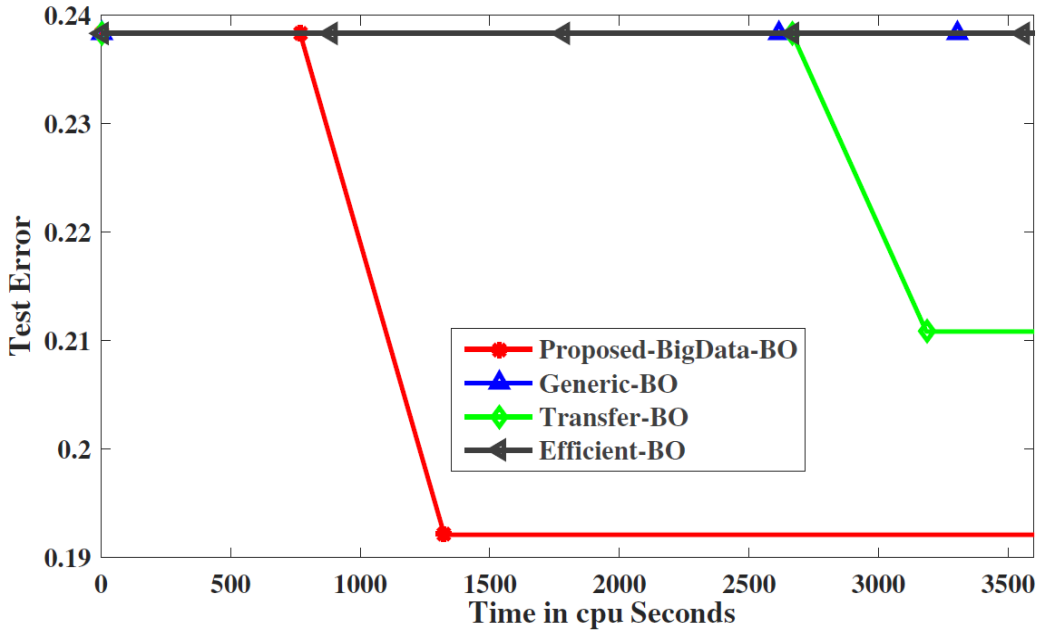
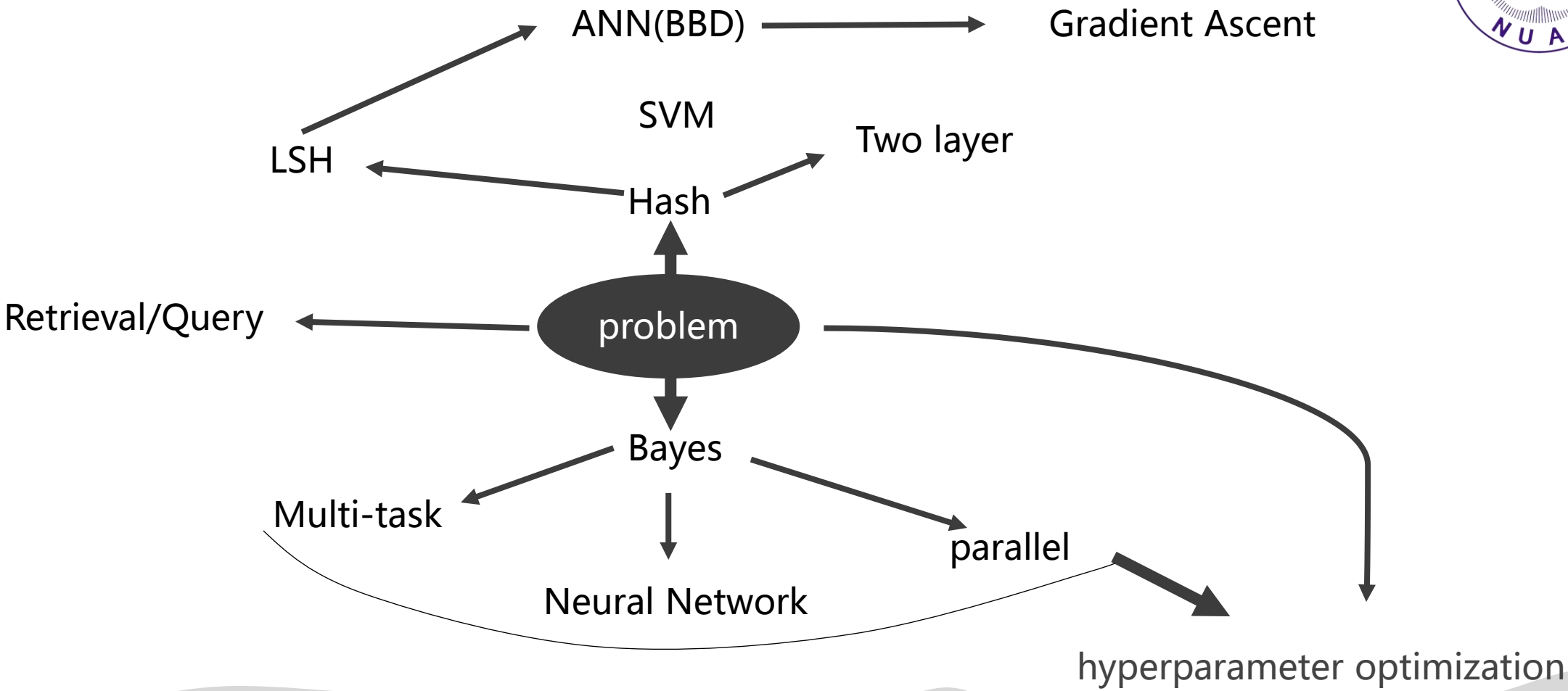


Figure 3. Tuning the Hyperparameters of SVM with rbf Kernel on a8a dataset: The Efficient-BO and Generic-BO did not move from the initial result and overlapped each other.

04 Review


ANN(BBD)
Gradient Ascent
SVM
LSH
Two layer
Hash
Retrieval/Query
problem
Bayes
Multi-task
parallel
Neural Network
hyperparameter optimization