Learning to Warm-Start Bayesian Hyperparameter Optimization

A joint work with Saehoon Kim and Seungjin Choi

Jungtaek Kim (jtkim@postech.ac.kr)

Machine Learning Group,
Department of Computer Science and Engineering, POSTECH,
77 Cheongam-ro, Nam-gu, Pohang 37673,
Gyeongsangbuk-do, Republic of Korea

June 19, 2018

Table of Contents

Motivation

Motivation

- Hyperparameter optimization usually suffers a cold-start problem.
- We can mimic human experts' behavior on selecting initial hyperparameters to learn historical initializations.
- Our method can learn meta-features over datasets.
- The learned meta-features can be used to initialize Bayesian hyperparameter optimization.

- ArXiv version (`https://arxiv.org/abs/1710.06219`)

Background

Hyperparameter Optimization

- It determines the best hyperparameter configuration $\boldsymbol{\theta}^*$ by minimizing a validation error $\mathcal{J}(\boldsymbol{\theta})$, given training and validation datasets.
- It uses random search or grid search as a candidate method of hyperparameter optimization [Bergstra and Bengio, 2012].
- SMAC [Hutter et al., 2011], Spearmint [Snoek et al., 2012], and TPE [Bergstra et al., 2011] have been proposed.

Bayesian Hyperparameter Optimization

- BHO searches minimum of validation error $\mathcal{J}(\boldsymbol{\theta})$, gradually accumulating a pair of hyperparameters and validation error.
- A surrogate function $\mathcal{M}_{\mathrm{surrogate}}$ estimates a black-box function with the previously observed hyperparameter vectors and validation errors.
- A next point $\boldsymbol{\theta}^{\dagger}$ is queried, maximizing an acquisition function $a(\cdot)$:

$$\boldsymbol{\theta}^{\dagger} = \arg\max_{\boldsymbol{\theta}} a(\boldsymbol{\theta} | \mathcal{M}_{\mathrm{surrogate}}). \tag{1}$$

- In this paper, we utilize GP regression as surrogate function, and expected improvement (EI) and GP upper confidence bound (GP-UCB) as acquisition functions.

Bayesian Hyperparameter Optimization

Algorithm 1 Bayesian Hyperparameter Optimization

Input: Target function $\mathcal{J}(\cdot)$, k initial hyperparameter vectors $\{\boldsymbol{\theta}_1^\dagger, \ldots, \boldsymbol{\theta}_k^\dagger\} \subset \boldsymbol{\Theta}$, limit $T \in \mathbb{N} > k$
Output: Best hyperparameter vector $\boldsymbol{\theta}^* \in \boldsymbol{\Theta}$

```
Initialize an acquired set as an empty set
for i = 1, 2, ..., k do
    Evaluate J_i := J(θ_i^†)
    Accumulate (θ_i^†, J_i) into the acquired set
end for
for j = k+1, k+2, ..., T do
    Estimate a surrogate function M_surrogate with the acquired set {(θ_i^†, J_i)}_{i=1}^{j-1}
    Find θ_j^† = arg max_θ a(θ | M_surrogate)
    Evaluate J_j := J(θ_j^†)
    Accumulate (θ_j^†, J_j) into the acquired set
end for
return θ* = arg min_{θ_j^† ∈ {θ_1^†, ..., θ_T^†}} J_j
```

- Initialize an acquired set as an empty set
- **for** $i = 1, 2, \ldots, k$ **do**
    - Evaluate $\mathcal{J}_i \coloneqq \mathcal{J}(\boldsymbol{\theta}_i^\dagger)$
    - Accumulate $(\boldsymbol{\theta}_i^\dagger, \mathcal{J}_i)$ into the acquired set
- **end for**
- **for** $j = k+1, k+2, \ldots, T$ **do**
    - Estimate a surrogate function $\mathcal{M}_{\text{surrogate}}$ with the acquired set $\{(\boldsymbol{\theta}_i^\dagger, \mathcal{J}_i)\}_{i=1}^{j-1}$
    - Find $\boldsymbol{\theta}_j^\dagger = \arg\max_{\boldsymbol{\theta}} a(\boldsymbol{\theta} | \mathcal{M}_{\text{surrogate}})$
    - Evaluate $\mathcal{J}_j \coloneqq \mathcal{J}(\boldsymbol{\theta}_j^\dagger)$
    - Accumulate $(\boldsymbol{\theta}_j^\dagger, \mathcal{J}_j)$ into the acquired set
- **end for**
- **return** $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}_j^\dagger \in \{\boldsymbol{\theta}_1^\dagger, \ldots, \boldsymbol{\theta}_T^\dagger\}} \mathcal{J}_j$

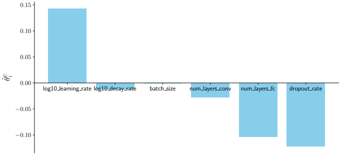
Why We Need to Learn Meta-Features

- Target distance can be understood as the ground-truth pairwise distance.
- However, there are two reasons why the target distance cannot be employed as the ground-truth distance directly:
  - a distance for new dataset without prior knowledge cannot be measured,
  - a distance function has a different multi-modal distribution with other mappings for different datasets.
- We compute a coordinate of center of validation error (CCoV) θ_i^{c} for a dimension i:

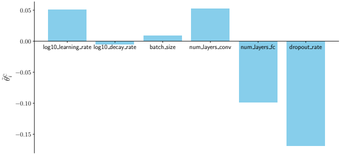
$$\theta_i^{\mathrm{c}} = \frac{\sum_{s=1}^{n} \tilde{\theta}_{si} \mathcal{J}_s}{\sum_{s=1}^{n} \mathcal{J}_s} \tag{2}$$

where a normalized hyperparameter $\tilde{\theta}_{si} = \frac{\theta_{si} - \min_{s=1,\ldots,n} \theta_{si}}{\max_{s=1,\ldots,n} \theta_{si} - \min_{s=1,\ldots,n} \theta_{si}}$ for $1 \leq i \leq d$ is provided.

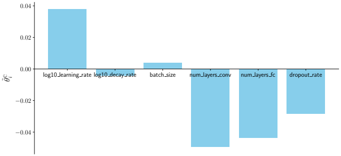
Why We Need to Learn Meta-Features



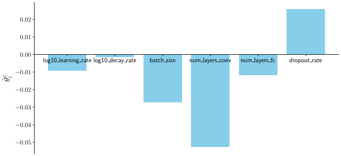
(a) Caltech256 (90%)



(b) CUB200-2011 (80%)



(c) MNIST (70%)



(d) VOC2012 (10%)

Target Distance

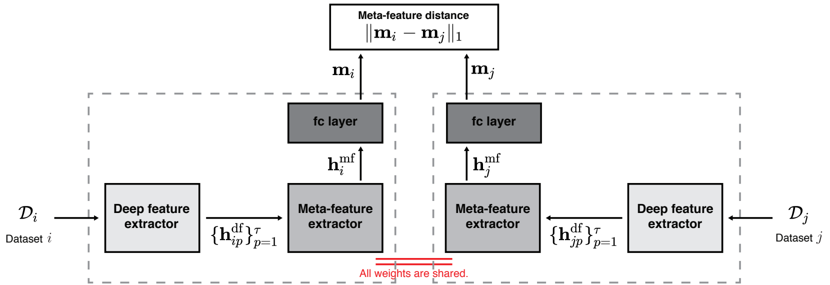
- Assume that there are $\{(\boldsymbol{\theta}_s, \mathcal{J}_s^{(t)})\}_{s=1}^n$ for $1 \leq t \leq K$, where
  - n is the number of historical tuples of hyperparameter vector and validation error for each dataset,
  - K is the number of the datasets which we have prior knowledge.
- We define a target distance function between two validation error vectors for the datasets $\mathcal{D}_i$ and $\mathcal{D}_j$, defined as L_1 distance:

$$d_{\text{target}}(\mathcal{D}_i, \mathcal{D}_j) = \left\| \left[\mathcal{J}_1^{(i)} \cdots \mathcal{J}_n^{(i)}\right] - \left[\mathcal{J}_1^{(j)} \cdots \mathcal{J}_n^{(j)}\right] \right\|_1 = \sum_{s=1}^{n} |\mathcal{J}_s^{(i)} - \mathcal{J}_s^{(j)}| \tag{3}$$

where $1 \leq i, j \leq K$.

Proposed Model

Overall Structure of Siamese Network



Overall Structure of Siamese Network

- A Siamese network is used to learn a metric such that distance between meta-features of datasets is well matched to the target distance.
- Our Siamese network has two identical wings each of which is composed of
  - deep feature extractor (denoted as $\mathcal{M}_{\mathrm{df}}$),
  - meta-feature extractor (denoted as $\mathcal{M}_{\mathrm{mf}}$).
- Specifically, the deep feature extractor transforms an instance of $\mathcal{D}_i$ into $\mathbf{h}_{ip}^{\mathrm{df}}$ for $p = 1, \ldots, n_i$, where n_i is the number of instances in the dataset $\mathcal{D}_i$.
- The meta-feature extractor transforms a set of deep features $\{\mathbf{h}_{ip}^{\mathrm{df}}\}_{p=1}^{n_i}$ into a meta-feature of $\mathcal{D}_i$, denoted as $\mathbf{m}_i$.

Learning a Siamese Network over Datasets

Algorithm 2 Learning a Siamese Network over Datasets

Input: A set of n datasets $\{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, target distance function $d_{\text{target}}(\cdot, \cdot)$, number of subsamples in a dataset $\tau \in \mathbb{N}$, number of iterations $T \in \mathbb{N}$

Output: Deep feature extractor and meta-feature extractor $(\mathcal{M}_{\text{df}}, \mathcal{M}_{\text{mf}})$ trained over $\{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$

```
Initialize M_df and M_mf
for t = 1, 2, ..., T do
    Sample a pair of datasets, i.e., (D_i, D_j) for i ≠ j, i, j = 1, ..., n
    Sample τ instances from each dataset in the pair (D_i, D_j) selected above, to make |D_i| = |D_j| = τ
    Update weights in M_df and M_mf using d_target(D_i, D_j) via optimizing Equation (1)
end for
return (M_df, M_mf)
```

Meta-Feature Extractor

- Dataset is a set of instances (i.e., images in this paper).
- It is permutation-invariant and the number of instances for each dataset can be varied.
- To resolve these problems, we consider two designs into meta-feature extractor:
  - aggregation of deep features (ADF): deep features $\mathbf{h}^{\mathrm{df}}$ are aggregated as summation or arithmetic mean of them:

$$\mathbf{h}^{\mathrm{mf}} := \mathbf{h}_{\mathrm{ADF}} = \sum_{p=1}^{\tau} \mathbf{h}_p^{\mathrm{df}} \quad \text{or} \quad \frac{1}{\tau} \sum_{p=1}^{\tau} \mathbf{h}_p^{\mathrm{df}}, \tag{4}$$

  - bi-directional long short-term memory network (Bi-LSTM): the deep features $\mathbf{h}^{\mathrm{df}}$ are fed into Bi-LSTM. Bi-LSTM can be written as

$$\mathbf{h}^{\mathrm{mf}} := \mathbf{h}_{\mathrm{Bi\text{-}LSTM}} = \mathrm{Bi\text{-}LSTM}(\mathbf{h}_{1:\tau}^{\mathrm{df}}), \tag{5}$$

where $\mathbf{h}_{1:\tau}^{\mathrm{df}}$ denotes $[\mathbf{h}_1^{\mathrm{df}}, \mathbf{h}_2^{\mathrm{df}}, \ldots, \mathbf{h}_{\tau-1}^{\mathrm{df}}, \mathbf{h}_\tau^{\mathrm{df}}]$.

Experiments

Experiment Setup

- We created a collection of datasets for training our model, using eight image datasets: AwA2, Caltech-101, Caltech-256, CIFAR-10, CIFAR-100, CUB-200-2011, MNIST, and VOC2012.
- In this paper we optimized and warm-started convolutional neural network, created by six-dimensional hyperparameter vector (log10_learning_rate, log10_decay_rate, batch_size, num_layers_conv, num_layers_fc, dropout_rate).
- We employed Bayesian optimization package, `GPyOpt` [The GPyOpt authors, 2016] in BHO.
- GP regression with ARD Matérn 5/2 kernel is used as surrogate function, and EI and GP-UCB are used as acquisition functions.

Bayesian Hyperparameter Optimization with Warm-Starting

- Our methods initialize BHO with 3-nearest best vectors predicted by ADF and Bi-LSTM.
- We used three initialization techniques:
  - naïve uniform random sampling (denoted as *uniform*),
  - Latin hypercube sampling (denoted as *Latin*),
  - quasi-Monte Carlo sampling with one of low discrepancy sequences, Halton sequence (denoted as *Halton*).

Bayesian Hyperparameter Optimization with Warm-Starting

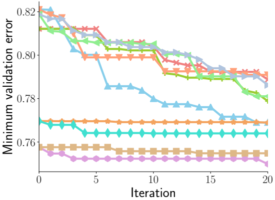
Algorithm 3 Bayesian Hyperparameter Optimization with Warm-Starting

Input: Learned deep feature and meta-feature extractors $(\mathcal{M}_{\mathrm{df}}, \mathcal{M}_{\mathrm{mf}})$, target function $\mathcal{J}(\cdot)$, limit $T \in \mathbb{N}$, number of initial vectors $k < T$

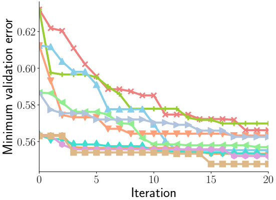
Output: Best hyperparameter vector $\boldsymbol{\theta}^*$

Find k-nearest neighbors using the learned deep feature and meta-feature extractors, $(\mathcal{M}_{\mathrm{df}}, \mathcal{M}_{\mathrm{mf}})$
Obtain k historical sets of tuples $\{\{(\boldsymbol{\theta}_s, \mathcal{J}_s^{(1)})\}_{s=1}^n, \ldots, \{(\boldsymbol{\theta}_s, \mathcal{J}_s^{(k)})\}_{s=1}^n\}$
Initialize an acquired set as an empty set
for $i = 1, 2, \ldots, k$ **do**
    Find the best vector $\boldsymbol{\theta}_i^\dagger$ on grid of the i-th set of tuples $\{(\boldsymbol{\theta}_s, \mathcal{J}_s^{(i)})\}_{s=1}^n$
    Evaluate $\mathcal{J}_i \coloneqq \mathcal{J}(\boldsymbol{\theta}_i^\dagger)$
    Accumulate $(\boldsymbol{\theta}_i^\dagger, \mathcal{J}_i)$ into the acquired set
end for
for $j = k+1, k+2, \ldots, T$ **do**
    Estimate a surrogate function $\mathcal{M}_{\mathrm{surrogate}}$ with the acquired set $\{(\boldsymbol{\theta}_i^\dagger, \mathcal{J}_i)\}_{i=1}^{j-1}$
    Find $\boldsymbol{\theta}_j^\dagger = \arg\max_{\boldsymbol{\theta}} a(\boldsymbol{\theta} | \mathcal{M}_{\mathrm{surrogate}})$.
    Evaluate $\mathcal{J}_j \coloneqq \mathcal{J}(\boldsymbol{\theta}_j^\dagger)$
    Accumulate $(\boldsymbol{\theta}_j^\dagger, \mathcal{J}_j)$ into the acquired set
end for
return $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}_j^\dagger \in \{\boldsymbol{\theta}_1^\dagger, \ldots, \boldsymbol{\theta}_T^\dagger\}} \mathcal{J}_j$

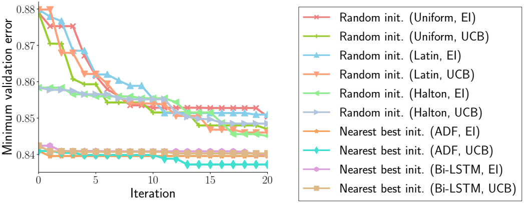
Bayesian Hyperparameter Optimization with Warm-Starting



(e) AwA2

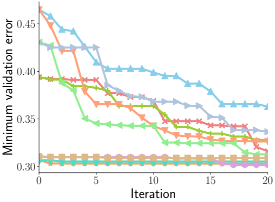


(f) Caltech101

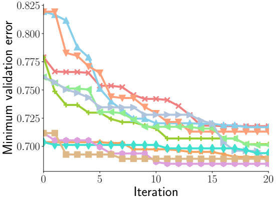


(g) Caltech256

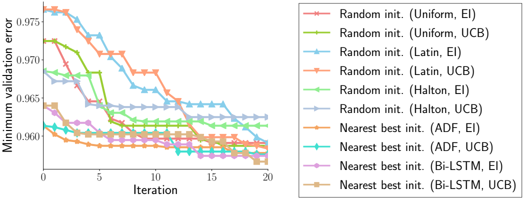
Bayesian Hyperparameter Optimization with Warm-Starting



(i) CIFAR-10

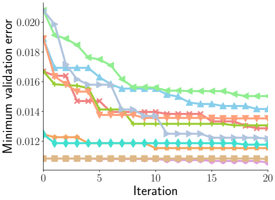


(j) CIFAR-100

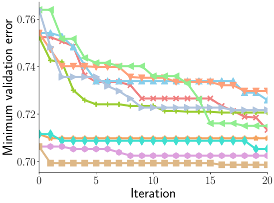


(k) CUB200-2011

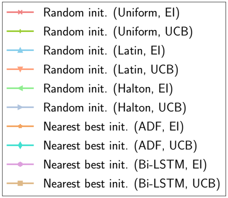
Bayesian Hyperparameter Optimization with Warm-Starting



(m) MNIST



(n) VOC2012



References

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems (NIPS)*, volume 24, pages 2546–2554, Granada, Spain, 2011.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization*, pages 507–523, Rome, Italy, 2011.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, volume 25, pages 2951–2959, Lake Tahoe, Nevada, USA, 2012.

The GPyOpt authors. GPyOpt: A Bayesian optimization framework in Python, 2016. `https://github.com/SheffieldML/GPyOpt`.