Ranking Loss Surrogates

MSc Thesis

Abdus Salam Khazi [Email]

Github Repository [3]

Supervisors: JProf. Josif Grabocka & Sebastian Pineda

April 29, 2022

Abstract

Abstract goes here

Contents

Chapter 1

Introduction

The performance of any machine learning model is sensitive to the hyper-parameters used during the model training. Instead of using a new model type, it is more helpful to tune the hyper-parameters of an existing model to improve its performance. Learning the best hyper-parameter for an ML model is called, Hyperparameter optimization (HPO in short). This thesis studies various existing approaches to HPO and proposes a new idea for the same using the concept of ranking. The proposed idea in this thesis is called **HPO using Ranking Loss Surrogates**. The results obtained using this model are compared against the state-of-the-art results obtained using models like FSBO, RGPE, TAF, and others.

1.1 Problem Definition - HyperParameter Optimization

To find out the best hyper-parameter for any machine learning model m, we must first quantify a given hyper-parameter configuration $\mathbf{x}$ by a real-valued number $v \in \mathbb{R}$. If we define that

$$\mathbf{x}_1 \succ \mathbf{x}_2 \iff v_{\mathbf{x}_1} < v_{\mathbf{x}_2}$$

then HPO can be defined mathematically by an abstract function, say, $f(\mathbf{x}) \mapsto \mathbb{R}$ as

$$\underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{S}$$

where $\mathbb{S}$ is the hyper-parameter search space.

This function $f(\mathbf{x}) \mapsto \mathbb{R}$ is evaluated in the following chronological steps:

1. Using a given hyper-parameter configuration $\mathbf{x}$, we train our model m to obtain the model $m_{\mathbf{x}}^{trained}$. It consists of learning the parameters of our model, E.g. learning the weights and biases of a Deep Neural Network. We use the training data to learn this model.

2. The validation data is passed through $m_{\mathbf{x}}^{trained}$ to obtain the required results. These results are evaluated based on an evaluation criterion 'eval'. This criterion is different for different problems, e.g. Regression, Classification, etc. The result of this evaluation is a real-value that gives a score for the configuration $\mathbf{x}$.

Hence the function $f(\mathbf{x}) \mapsto \mathbb{R}$ can be written as

$$\text{eval}(m_{\mathbf{x}}^{trained}(\text{Data}_{\text{val}})) \mapsto \mathbb{R}$$

Finally, the HPO problem can be defined using the following equation:

$$\underset{\mathbf{x}}{\text{argmin}} \;\; \text{eval}(m_{\mathbf{x}}^{trained}(\text{Data}_{\text{val}})) \mapsto \mathbb{R} \quad \forall \mathbf{x} \in \mathbb{S}$$

1.2 Optimization Constraints

Hyper-parameter optimization is different from other optimization methods because it has different constraints. It is because of the peculiar properties of the hyper-parameter search spaces. Finding out the correct hyper-parameter setting is generally not feasible using a brute-force approach (trying out all possible combinations of hyper-parameters) because the search space itself has many dimensions, and the search space may be continous. More specifically, some of the important constraints of this optimization problem are:

1. The evaluation of a given HPO configuration is computationally expensive.

2. It is a non-convex optimization problem.

3. The process of getting $m_{\mathbf{x}}^{trained}$ from m is stochastic hence the value $v_{\mathbf{x}}$ is noisy.

4. Some dimensions have conditional constraints. The values of some dimensions may depend on the values of others. For example, the number of neurons in layer 3 only makes sense if we have 3 or more layers in a neural network.

5. The search space is hybrid in terms of continuity. Some of the dimensions (variables) may be continuous, while others may be discrete. Using a gradient method is hence not trivial.

1.3 Overview

This sections contains the overview of the paper and how the thesis report is organised.

Chapter 2

Literature Review

To deal with the constraints of HPO problems, researchers have used different strategies for developing HPO algorithms and models. Some of the straightforward methods include

- Manual Search
- Grid Search
- Random Search

Manual Search in the HPO search space is feasible when we have expert knowledge of the problem and the given model. The idea is to select configurations step by step by observing the results obtained so that we do not waste computation time evaluating similar configurations through intuition. This approach may be helpful for small models with lesser constraints. However, as the HPO search space becomes very large or conditional constraints become too complex, the selection of configurations becomes more and more difficult. Hence a more systematic and automated approach is more practical.

Grid search is a more systematic approach in which we divide the search space into grid points similar to ones in a Euclidean graph. Let there be m dimensions in the search space $\mathbb{S}$. Let the selected number of values for each dimension be $n_1, n_2, ...n_m$. In the case of a discrete variable, the selected values will be a subset of the possible values, whereas, in the case of a continuous variable, we need to select the values based on a selected granularity. The cross-product of the selected subsets gives us the configurations to be evaluated. Hence, the number of configurations to evaluate will be $n_1 * n_2 * ...n_m$.

The number of configurations we need to evaluate in this approach becomes a big issue for this method as the dimensions of the search spaces increase. Hence this approach becomes intractability for large search spaces.

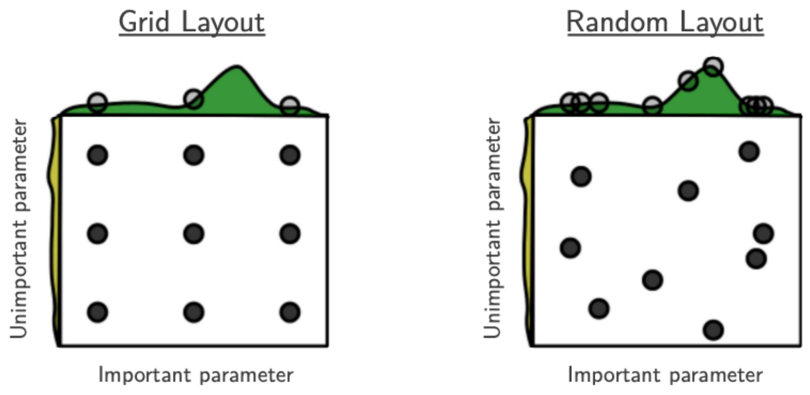

Figure 2.1: Illustrates Grid search and Random search in the case where 2 parameters are not equally important. Adpated from [1].

One issue with the Grid Search approach is that we assume that all dimensions in the HPO search space are equally important. It is not the case in many HPO problems. The Grid layout in Figure 2.1(left) shows illustrates this. For example, the learning rate in deep neural networks is much more important than many other parameters. If dimension p is the most important in the search space, then it makes sense to evaluate more values of p. Random Search helps us solve this problem. The Random layout in Figure 2.1(right) illustrates this. Hence Random Search can be used as a trivial baseline for comparing other HPO models.

One advantage of these methods is that there are no restrictions on the HPO search spaces. Hence, they are suitable for any HPO problem at hand. On the other hand, these methods are non-probabilistic. Hence they cannot deal with noisy evaluations of the HPO configuration well. Moreover, these methods are computationally expensive. The reason is that they do not use surrogate evaluators and hence train and evaluate the whole model. Also, these search methods give us optimal HPO configurations only by chance.

In the remainder of this section, we discuss some sophisticated probabilistic models for doing HPO that use surrogates to reduce computational costs.

2.1 Bayesian Optimization

Bayesian optimization tries to solve both computational costs and noisy evaluations of our objective function (section 1). It does this by building a model of the HPO objective function. This model is called a surrogate function. Bayesian optimization uses known evaluations as its data to build the surrogate model. The data is of the form x, f(x) pairs. The surrogate model is a probabilistic model. Hence, it also learns about the noise in the evaluations of the objective function.

The core procedure of the optimization process is the following:

- From known data $D = (x_1, f(x_1)), (x_2, f(x_2)), (x_3, f(x_3))....$, build a probabilistic model that learns the mean and variance of the objective function
- Use the surrogate to sample the next best HPO configuration x'. Evaluate f(x')
- Append (x', f(x')) to D and repeat the process.

The above process repeats till the computational resources are finished (here time) or we find an acceptable HPO configuration. This procedure is also called SMBO (Sequential model-based Optimization). The procedure alternates between collecting data and fitting the model with the collected data [2].

There are two essential components of Bayesian optimization:

- Probabilistic model of the objective function
- The acquisition function

There are many probabilistic models such as Random Forests, Gaussian Processes, Tree parson Estimators, etc. But for the purpose of this thesis, we briefly mention Random forests and discuss in-depth the Gaussian processes.

2.1.1 Probabilistic models

Random Regression Forest

The core idea of this model is to train a Random Regression Forest, using the known data as in any SMBO procedure [2]. Random regression forests are an ensemble of regression trees. This property is used to our advantage to predict the mean and the variance. The mean of the prediction of all the trees is the mean of the surrogate model. The variance in the prediction of all trees is the variance of the surrogate model.

The advantages of this model are

- It can handle both continuous and discrete variables trivially without any modifications to the model. The data splitting during training is done using any variable be it discrete or continuous.

- It can handle conditional variables, unlike Gaussian processes, by making sure that data is not split based on a variable till it is guaranteed that no conditionality is broken by the split.

Gaussian Process Regression

Gaussian processes [6] are predictive machine learning models that work well with few data points (or data pairs). They are inherently capable of modeling uncertainty. Hence, they are used widely in problems such as hyperparameter optimization, where uncertainty estimation is essential. In this section, we briefly explain the Gaussian process regression intuitively.

Before we proceed, we need to understand normal (Gaussian) distributions. Consider a scalar random variable X that is distributed normally (a.k.a Gaussian distribution) around a mean μ with a variance of σ^2. The following equation defines the probability density function (PDF) of X:

$$P_X(x) = \frac{1}{\sqrt[2]{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Here, X represents the random variable, and x represents an instance of the variable [6]. In this case, the mean μ, variance σ^2, and any sample x are all scalars.

If the random variable $\mathbf{X}$ is a vector in $\mathbb{R}^d$ where $d \in I^+$, then each component of the vector can be considered as a random variable. In this case the mean $\boldsymbol{\mu} \in \mathbb{R}^d$ whereas variance, represented by Σ, is in the $R^{d\times d}$ space. It is because the variance of all components in any valued vector random variable $\mathbf{X}$ should contain the following two types of variance

- Variance of a vector component w.r.t itself. d diagonal values of the matrix Σ represent this variance.

- Variance of each vector component w.r.t all other components. These variances are represented by the upper/lower triangular values in the matrix Σ.

The matrix Σ, also known as the Covariance matrix, thus has all values necessary to represent the variance of any vector-valued random variable.

The probability density function of a vector valued variable $\mathbf{X} \in \mathbb{R}^d$ with a mean $\boldsymbol{\mu}$ and covariance matrix Σ is given by [5]:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{d}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

This equation defines the PDF of a multivariate normal distribution.

The core idea used in the Gaussian processes is that functions can be considered as vectors of infinite dimensions. Consider any function f that has a domain $\mathbb{R}$. If f is considered to be a vector in $\mathbb{R}^\infty$, then each point $i \in \mathbb{R}$ can be represented by a component f_i of the function f. A function, hence, is nothing but a sample from $\mathbb{R}^\infty$. Unfortunately, functions sampled from $\mathbb{R}^\infty$ are too general and not useful by themselves.

The idea of Gaussian processes is to sample smooth functions from $\mathbb{R}^\infty$. In any smooth function f, if any point g is close to x in the domain of f, then $f(g) \approx f(x)$. It is mathematically represented by the following equation:

$$\lim_{\delta x \to 0} f_{x+\delta x} \approx f_x^+ \text{ and } \lim_{\delta x \to 0} f_{x-\delta x} \approx f_x^-$$

$$\text{where } \delta x > 0 \text{ and } x, \delta x \in \mathbb{R}$$

The above definition is nothing but the definition of a smooth function in terms of vector notation. Moreover, nearby components of f "vary" similarly w.r.t each other. These properties can be naturally encoded using a covariance matrix. Hence, we obtain smooth functions if we sample them from a multivariate normal distribution with the required covariance matrix. The Gaussian process restricts the function sample space to a multivariate normal distribution.

The similarity between 2 points in a domain is defined by a function called **kernel** in Gaussian processes. Using this kernel function, the values in the required covariance matrix are populated. The smoothness of the sampled function f is controlled by the kernel in the GP process. Formally kernel k is defined as,

$$k(\mathbf{x}, \mathbf{x'}) \mapsto \mathbb{R}$$

Here, $\mathbf{x}, \mathbf{x'}$ belong to a domain in the most abstract sense. For example, when the input domain is a euclidean space, $\mathbf{x} \in \mathbb{R}^{\mathbb{I}^+}$.

2.1.2 Acquisition functions

Due to the robustness of the GP process, we use this as one of the baselines in our thesis.

2.2 Deep Ensembles

One way to do Hyper Parameter Optimization is by using Uncertainty prediction is the key in the paper [4].

2.3 RGPE

This section is necessary because our model performs similar to this

2.4 TAG

Some other model if also necessary and you mention this in your report.

2.5 Deep Kernel Learning

2.6 Ranking Loss functions

Types of ranking losses

2.6.1 Pair wise ranking losses

2.6.2 ...

2.6.3 List wise ranking losses

Chapter 3

Baseline Implementations

3.1 Dataset

This section defines the requirement of the dataset and why meta data sets are important

3.1.1 Models used

3.1.2 Meta dataset

Various search spaces sizes

3.1.3 HPO_B Dataset

How HPOB datasets is organised.

3.2 Deep Ensemble

3.2.1 Requirement of restarting training

We find in our experiments that the deep ensemble performance much better in our HPO cycle when we train it from scratch at every acquisition step. This is counter intuitive because an already trained model will converge to a local optima quickly. The reason for this performance however is that the model gets biased towards the points that are observed in the beginning. Lets say we have 2 models

- $\text{DE}_{\text{restart}}$ which always restarts training at every acquisition step
- DE_{reuse} which uses the previously trained model for subsequent training of the optimization cycle.

The following figures show the number of times each observed data points is used for training at 25th 50th and 100th optimization cycle. All the data is used in the optimization cycle as the amount of data is very less. Hence the diagrams are scaled with the number of epochs.

The heavy bias that you can see in the figures is actually not intended. This is because all observations should be treated equally in any training/fine-tuning step.

Create Bias diagram...

The second reason is that we may not be able to get out of the local minima. Support using a diagram.

3.3 Few Shot BO: Deep Kernel Learning

Chapter 4

Proposed Idea: Ranking Loss Surrogates

The search spaces that have less data need more uncertainty The search spaces having more data need less uncertainty

We use list wise ranking losses because they have been found to be more efficient

4.1 case study: Sorting

4.2 Scoring function analysis

4.2.1 Scoring function range

4.3 Optimization cycle

4.3.1 Meta training

4.3.2 Fine tuning

4.4 Loss function implementation

The loss function used is listMLE the implementation is done by removing to element check comments The latest implementation is used due to its efficiency otherwise it is the same. (note)

4.4.1 Weighted Loss

Weighting becomes important because we are doing fine tuning at every step. Show the equation for weighting Used log weighting because Advantages: Can be used to select best n configs and then evaluated at random based on the ranks. This is not as trivial to get in otherplaces. So there is an amount of parallelism that can be built it.

Parallelisation of elements is possible.

show the architecture of with only scorer

4.5 Use of Deep Sets

why are deep sets important.

Architectecture with deep sets.. Image

4.6 Uncertainty implementation

Mention about the requirement to use Module list. Show the new architecture.

4.7 Advantages and Limitations

Compare this loss function and method with other base lines..

Chapter 5

Experiments and Results

experiment and results protocol data split. . .

First show the resutls of implementation of GP (M = 5 and M =10) Benchmark this...

Next with DKT (Benchmark this)

Next show the best results obtained by architecture.

5.1 Evaluation

5.1.1 Testing

5.1.2 Ablation

show the results of raw without deep set.

Next show different strategies used for building the ranking loss model one step at a time. First with only scorer. then with deep set. Then with raw deep set fine tuning and deep set adding uncertainty

Checking the early stop and hypothesing why is was wrong.

box plot variation of each of the scorers... for 1 or more data sets?

what about training independently , this requires normalization. as explained by sebastian.

Chapter 6

Conclusion

6.1 Further work

Further study required with other baselines that deal with ranking loss.

6.2 Conclusion

Bibliography

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, feb 2012.

[2] Holger H. Hoos Frank Hutter and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. Paper.

[3] Abdus Salam Khazi. Code. [Online; accessed 06-Feb-2022].

[4] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles, 2017.

[5] K. P. Murphy. Machine learning: A probabilistic perspective, 2012.

[6] Jie Wang. An intuitive tutorial to gaussian processes regression, 2020.

Appendix A

More information

This thesis was completed in the representation learning lab of Albert-Ludwig-Universität Freiburg. (Figure A.1)

Figure A.1: Logo: Albert-Ludwig-Universität Freiburg