Ranking Loss Surrogates

MSc Thesis

Abdus Salam Khazi [Email]

Github Repository [6]

Supervisors: JProf. Josif Grabocka & Sebastian Pineda

April 30, 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 **Hyperparamter Optimization 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 Methodology

1.2 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:

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

1.3 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 continuous. 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.

1.4 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.

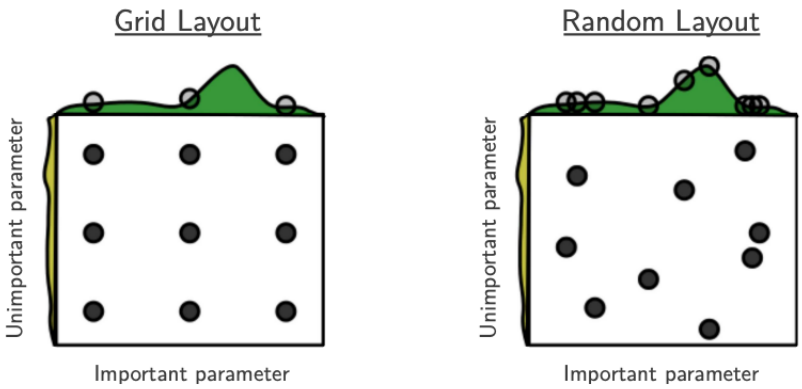

Figure 2.1: Illustrates Grid search and Random search in the case where 2 parameters are not equally important. Adpated from [1].

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' using a function known as acquisition function. 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 [3].

Hence, 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 [3]. 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 [12] 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 [12]. 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 [8]:

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

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:

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

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

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, $\mathbf{x} \in \mathbb{R}^{\mathbb{I}^+}$.

Some well known kernels are:

- **Radial Basis Function Kernel**

- **Matern Kernel**
- **Periodic Kernel**

Finally, a Gaussian Process specifies that any new observation y^* for input $\mathbf{x}^*$, is jointly normally distributed with known observations $\mathbf{y}$ (corresponding to the input $\mathbf{X}$) such that

$$Pr\left(\begin{bmatrix}\mathbf{y}\\ y^*\end{bmatrix}\right) = \mathcal{N}\left(m(\mathbf{X}), \mathbf{\Sigma}\right) \tag{2.1}$$

Here, $m(\mathbf{X})$ is the mean of the vectors which is commonly taken as $\mathbf{0}$. $\mathbf{\Sigma}$ is the covariance matrix defined as

$$\mathbf{\Sigma} = \begin{bmatrix}\mathbf{K} & \mathbf{K}_*\\ \mathbf{K}_*^T & \mathbf{K}_{**}\end{bmatrix}$$

Where $\mathbf{K} = k(\mathbf{X}, \mathbf{X})$, $\mathbf{K}_* = k(\mathbf{X}, \mathbf{x}_*)$ and $\mathbf{K}_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$ for any given kernel k [12]. Due to the robustness of the GP process, we use this as one of the baselines in our thesis.

2.1.2 Acquisition functions

The acquisition functions need to do balance exploitation of information from the known data points and exploration of unknown data points in their domain. The following functions are some of the most prominent acquisition functions found in the literature [12]

- **Upper Confidence Bound (UCB)**: It returns the best possible using a linear combination of the mean and the standard deviation.
- **Probability of Improvement**: It gives The probability with which we can get better values than the best so far is the acquisition function.
- **Expected Improvement**: Given a gaussian distribution at a new input point, it finds the expectation of improvement i.e $(f(x) - f_{max})$ over the part of normal that is greater than f_{max}.

2.2 Deep Ensembles

Deep Neural Networks (DNNs) are machine learning models with very high representational capacity [5]. Due to this property, one can use them as surrogates for the HPO objective function f. But the issue is that DNNs do not quantify uncertainity trivially. In the following section we discuss briefly the methods used to do uncertainty quantification using neural networks.

2.2.1 Uncertainty Quantification in ANN

In the current literature, uncertainty quantification methods using deep neural networks can be broadly classified into the following methods:

- Bayesian Nueral networks [4].
- Ensemble Approach using monte carlo [11].
- Ensemble approach using multiple neural networks.

In Bayesian neural network(BNN), a prior over weights and biases is specified prior to training the BNN. Given the data, a posterior predictive distribution is calculated for all the parameters of the network (Weights and Biases). One issue with this approach is that BNNs are very complex and difficult to train. [citation]

Monte carlo drop out is a regularization technique used during the training of neural networks. With a certain probability, connections between neurons are dropped. Using this technique one obtains possibly 2^N neural networks where N is the number of connections in the artifical neural network. We can get an ensemble of high capacity models for free. It is normally only used during training to obtain regularization.

However, if one uses monte carlo dropout during the evaluation, we can get multiple results from the same input using this approach. Given input x and output $y = \text{NN}(x)$. If we have m neural networks obtained using monte carlo dropout, we get $\{y_1, y_2 ... y_m\}$ outputs, we can obtain the mean and variance of

$$y_{\text{mean}} = \frac{\Sigma_i y_i}{m} \qquad y_{\text{variance}} =$$

There are 2 ways of using an Ensemble of Deep Nueral networks to get uncertainty prediction. One is by using Monte Carlo Dropout during evaluation. [citation of monte carlo dropout]

One way to do Hyper Parameter Optimization is by using Uncertainty prediction is the key in the paper [7].

2.3 RGPE

Read the FSBO paper and write the summary like that or read the RGPE paper This section is necessary because our model performs similar to this

2.4 TAG

Read the TAG paper. Some other model if also necessary and you mention this in your report.

2.5 Deep Kernel Learning

2.6 Ranking Loss functions

Ranking as a concept used in information retrieval to rank documents based on relevance to a query.

Consider a problem of ranking a set of objects $\{d_1, d_2, d_3...d_n\}$. For this a ranking function (rf) needs to be learnt. One can learn the ranking function by modelling this learning process as an optimization problem. This optimization needs a criteria (or a loss function in the jargon of machine learning) to do its optimization. This criteria is called a ranking loss function. This function (rf) is hence learnt by optimizing a ranking loss function.

After the learning rf, one can use this function to rank newly given objects.

Ranking functions can be broadly classified into 3 types

- Point-wise ranking
- Pair-wise ranking
- List-wise ranking

In Point-wise ranking function, the input to the function is a single data point vector. The output of this function is a label or a score for that data point. Correspondingly, the ranking function is called point wise ranking function

In pair-wise ranking functions the input to the function are pairs of input objects. This ranking function can be understood as a classification problem. The function tries to separate the input data points as much as possible in the output space. Correspondingly, the loss function that optimizes/learns a pair wise ranking function is called pair-wise ranking loss.

In list-wise ranking functions, the input to the function is a list of objects to rank. This is the most general ranking function and also the most useful one. To understand this, consider 2 opposite objectives. One objective to rank the object. The second objective is to put other objects below this one in the list. Loss is approximately (by the equation) given by $-$ rank $+$

number of objects below it. If the rank is low and more number of objects are below it, then loss tends to be positive which is not desirable On the contrary, if the rank is high it can bear more objects below it as Loss would not be so high.

Correspondingly, the loss function that aids in learning this list-wise ranking function is called list wise ranking loss function.

2.7 Deep Sets

Chapter 3

Baseline Implementations

In this thesis 2 baselines have been implemented

- Deep Ensembles
- FSBO with Deep Kernel Networks

Both these models have built in capability for uncertainty estimation. Hence we have to use an acquisition function during the fine tuning step. We use expected improvement in all models that deal with uncertainty. This is to maintain constant results across all the models. We also use this in the proposed idea. Also due to the advantages of this acquisition function.

Few Shot Bayesian Optimization deals has to be run in 2 chronological steps

- Meta training using the existing meta data
- Fine tuning during the optimization cycle.

For the purpose of meta training, there is an existing meta dataset called HPOB. HPOB is a meta dataset for blackbox HPO [9]. The following section describes HPOB dataset and its organization

3.1 Meta Dataset

The meta data set in HPOB is divided into test and train data. Using meta dataset, one can learn meta learn a model using the training split. Then the model is evaluated in the testing split. This data approach is used both in the baselines and proposed idea models.

3.1.1 HPO-B Dataset

How HPOB datasets is organised. The meta data in HPOB is formatted in a json format.

3.1.2 HPO-B evaluation

To use the HPO-B code, one must write function observe_and_suggest and observe_and_suggest_continous to deal with discrete and continous optimisation case respectively. Due to out of scope nature of the continous case, in our implementation we assume that we are dealing only with the discrete case only.

The implementation of observe_and_suggest functions differs based on the surrogate model used by our problem.

The following 2 sections discuss the implementation details of both these methods.

3.2 Deep Ensemble

As discussed in the literature review section, deep ensemble uses a list of artificial neural networks to find out uncertainties given data points D.

Some of the important implementation details are

- We make sure that we enforce the positivity of the variance (as mentioned in the paper) by adding $exp(-6)$ to the values for numerical stability.
- We do not use adverserial examples because it was giving bad results.

3.2.1 Requirement of restarting training

We find in our experiments that the deep ensemble performs 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 at the beginning of the optimisation cycle. 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.

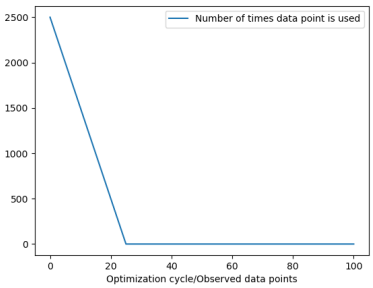

Figure 3.1: Bias at 25th optimization cycle

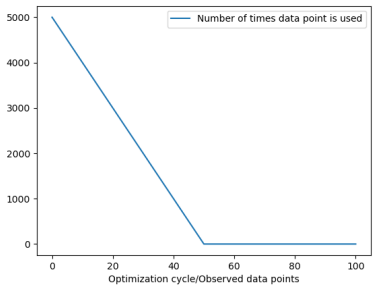

Figure 3.2: Bias at 50th optimization cycle

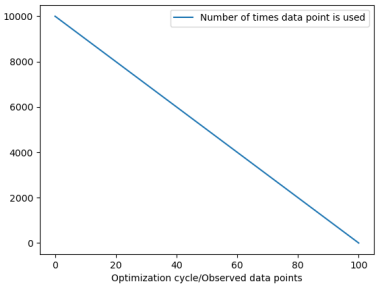

Figure 3.3: Bias at 100th optimization cycle

Let DE_{reuse} be fine tuned for 100 epochs during the optimization cycle. Let there be just 1 data point to start with. The figures 3.1, 3.2, and 3.3 show the number of times each observed data points is used for the fine tuning at the 25th, 50th and 100th optimization cycle by DE_{reuse} respectively. Generally all the data is used during fine tuning due to its scarcity at this step. Hence, the biases are scaled with the number of epochs. $\text{DE}_{\text{restart}}$ does not face this issue as the fine tuning step restarts everytime.

The heavy bias that is present in the figures is actually not intended. This is because all observations should be treated equally in any training/fine-tuning step. Hence it is better to restart the model before fine tuning step at each optimization step.

The second reason is that the model may get stuck at a stubborn local minima at any fine tuning step n $1 <= n <= 100$. Coming out of this local minima may require vastly different data points than actually seen. This is not generally the case in our sequential optimization.

3.2.2 Optimization Procedure

Deep ensemble is not a meta transfer model. So it does not use the train data of HPO-B meta dataset. It directly optimises during the optimisation loop.

The optimization loop can seen in the following algorithm.

The follwoing should be written in an algorithm. First fitting the model based on observations. Predicting then the pending configuration for evaluation. Returning the index that gives us the best results.

3.3 Few Shot BO: Deep Kernel Learning

Some of the important implementation details are

- Scaling is ignore because it gives very bad results.
- We first used RBF kernel, then we used matern kernel for this.

Scaling should be ignored because it gives very bad results. if scaling:
We used early stopping mechanism for this because we wanted to avoid

- Huge computation costs
- overfitting of the model during training.

We saved the best model in our implementation and if the validation loss went greater than lowest validation loss for some number of steps, then we stop the meta training.

3.3.1 Implementation issues

After completing the own implementation, we found that the results obtained were not in par with the paper. For this reason, the results are used only for the first stage. The results already present in the benchmarking data of [9] were used for comparing this result with our implementation.

3.3.2 Using cosine annehealing

We obtined loss curves that were very bad during the training and were varying quite a bit The reason was that the learning rate during the fine tuning was too high Hence we used cosine annehealing for this to get the a better local minima.

Show before and after curves.

The results of this are discussed in the results section.

Chapter 4

Proposed Idea: Ranking Loss Surrogates

In this chapter we chronologically build a Ranking loss surrogate model for HPO using SMBO. First we discuss the implementation of loss function used. Thereafter we discuss a case study of sorting. Then we add the following components one after another

- Deep Set
- Weight
- Uncertainty
- Different training mechanisms.

We decided to model a scoring function using a deep neural network. This is because the representational capacity of the neural network is very high.

4.1 Loss function implementation

The loss function used is listMLE As discussed in the literature review, we use list wise ranking losses because they have been found to be more efficient in most general case. 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)

Our implementation is given by the following algorithm Write the algorithm of our implementation here

Implementation given by the paper [10]. Here give the equations used by this algorithm

Main Idea: We need to use ranking losses instead of other methods for the purpose of HPO algorithms.

Applying this to HPOB... (With or without query) First learning from first search space. Even with this the loss curves are very smooth if we use 2 * tanh(0.01 * x)

4.2 Case study

In this case study, we study how the learnt ranking function behaves when we use different parameters to train. For this a toy example of sorting is considered.

4.2.1 Sorting Problem

The main problem that we try to solve here is - Is it possible to train a ranking function using the Listwise maximum likelihood estimator. such that it learns to sort a given list of numbers in the descending order?

Consider a list $l = \{x_1, x_2, ...x_n\}$ where $x_i \in [1, 100]$. Let $s(x \mid \theta) \mapsto \mathbb{R}$ be our scoring function parameterised by θ. One list contains n data points sampled from $[1, 100]$. Let $\mathrm{L}_{\mathrm{mle}}$ represent the maximum log likelihood loss. Using this loss function we optimize for the following equation

$$\arg\min_{\theta} \quad \mathrm{L}_{\mathrm{mle}}(s(l \mid \theta), -k * l) \tag{4.1}$$

where $k \in \mathbb{R}$.

Note that second parameter of list wise loss function is scale invariance because it only uses the rank of the objects in the list and not the objects directly. Hence scaling the list has no effect on the loss output. This can be seen from the algorithm of implementation. (refer algorithm)

The validation data taken from 3 different ranges

- Same range as the training data $[1, 100]$
- Completely different range as seen by the scorer during training i.e $[-100, -1]$
- Hybrid range i.e $[-50, 50]$

To evaluate our scorer, we first sample the validation data from the above ranges, We then check the percentage of the lists that are correctly sorted during our testing time. The training batch is 100 and the results are averaged over 5 runs.

tabulate the results and Check the affect of list / batch size on the results

Training Epochs	List size	Learning Rate	In-range Acc.	Out-range Acc.	Hybrid Acc.
1000	3	0.0001	0.99	0.99	0.99

Table 4.1: Sorting Accuracies at test time

show the architecture of with only scorer

4.2.2 Observations

As we know that the second parameter of the loss function is not bound to any range, the score function itself can learn scores of arbitrary range. Hence the model can learn score values arbitrarily large. We must control the output domain so that the training does not lead to underflow or overflow. This is because we use the exp function for getting the strict positive increasing function.

The restriction of output was done using tanh() so that the range of values of the scorer remains within a managable value. One drawback we found was that the learnt model wsa extremely sensitive to the learning rate and the number of epochs.

Propose using a higher list size.

4.3 Scoring function analysis

4.3.1 Scoring function range

4.4 Optimization cycle

Explain the training loop with an algorithm.

explain get_batch_HPBO implementation how it is important to have all the data sets How it is organised into a higher order tensor TO compute 2 things can be done 1. View it as a 2d tensor (Done by efficient implementation) 2. Deal with it direction (Done by us)

replace=False thing in the above function and why it is important.

4.4.1 Meta training

Give the meta training algorithm

4.4.2 Fine tuning

Given the Fine tuning algorithm. For reasons described in the 3.2.1, we restart before finetuning at every optimization cycle step.

4.5 Use of Deep Sets

why are deep sets important.

Architectecture with deep sets.. Image

4.6 Weighted Loss

General list wise loss functions are only required in the problem domains where it is essential to get the ranks of all the objects correctly. In our case where ranking functions are used as surrogates for the black box hyper-parameter optimization, this constraint can be relaxed significantly. As discussed in section 4.4.2, we always restart our ranking function model before fine-tuning it. Hence, after fine-tuning we are only bothered about getting the best hyper parameter configuration from the pending configurations. This means that it is more important for our ranking function to order the the top part of the list than the bottom part. In this section we describe and propose the concept of weighted ranking loss functions.

The concept of biased ranking loss is discussed in the paper, "Top-Rank Enhanced Listwise Optimization for Statistical Machine Translation" by Chen et. al [2]. Here Chen et. al proposes a position based weighting of the ranking function such that:

$$w_j = \frac{k - j + 1}{\Sigma_{t=1}^{k} t} \tag{4.2}$$

where w_j represents weight of the object at position j in the ordered list.

However, there are also some simple weightings that a person can use. For instance inverse linear weighting given by

$$w_j = \frac{1}{j} \tag{4.3}$$

and inverse logrithmic weighting given by

$$w_j = \frac{1}{\log(j+1)} \tag{4.4}$$

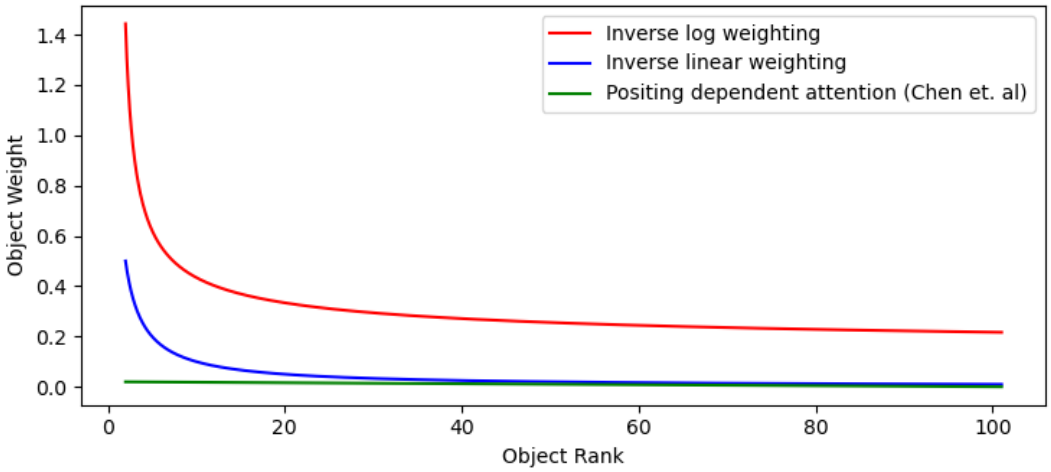

Figure 4.1: Different Weighting Functions

Figure 4.1 shows all 3 weighting strategies. Of the 3 weighting strategies, the inverse log weighting was found to be most suitable. This is because the this weighting is by far the least sharp. As the weighting does not decrease very rapidly, there is a possibility of parallelizing during the optimization cycle. For example, at every step the top n configurations can be selected, and all of them can be tried for optimization. The parallelism is less effective when using inverse linear weighting or weighting proposed by Chen et al.

4.7 Uncertainty implementation

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

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

4.7.1 Independent training

Implementation yet to be done

4.7.2 Training with mean and restricted output

Implementation yet to be done

4.8 Advantages and Limitations

Compare this loss function and method with other base lines.. Advantages of the proposed Idea: The amount of data instances for training is exponential in number. Which is very good for a deep learning model For example if we have 100 observation set and we use a list size of 15 to train our model, we will have 100C15 unique instances to train.

Observed disadvantages: The learnt model is extremely sensitive to the learning rate and the number of epochs

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

explain how a ranking graph works ar implemented Explain the regret rank@ some location.

5.1.2 Ablation

Result tabulation of case study: sorting: 1. Within range 2. Outside range mean of 3 times should be written.

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?

show results of independent training and training with output restriction

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

Chapter 6

Conclusion

6.1 Limitations

6.2 Further work

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

6.3 Conclusion

Bibliography

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

[2] Huadong Chen, Shujian Huang, David Chiang, Xinyu Dai, and Jiajun Chen. Top-rank enhanced listwise optimization for statistical machine translation, 2017.

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

[4] Ethan Goan and Clinton Fookes. Bayesian neural networks: An introduction and survey. In *Case Studies in Applied Bayesian Data Science*, pages 45–87. Springer International Publishing, 2020.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

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

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

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

[9] Sebastian Pineda-Arango, Hadi S. Jomaa, Martin Wistuba, and Josif Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on openml. *Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2021.

[10] Przemyslaw Pobrotyn, Tomasz Bartczak, Mikolaj Synowiec, Radoslaw Bialobrzeski, and Jaroslaw Bojar. Context-aware learning to rank with self-attention. *ArXiv*, abs/2005.10084, 2020.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[12] 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