Ranking Loss Surrogates

MSc Thesis

Abdus Salam Khazi [Email]

Github Repository [3]

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 **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.

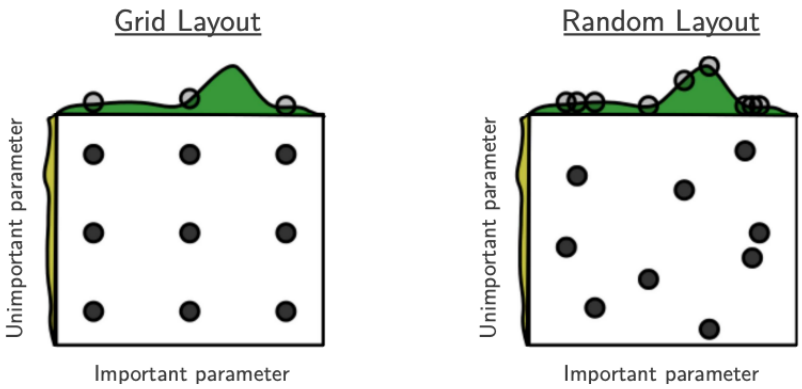

Figure 2.1: Illustrates Grid search and Random search in the case where 2 parameters are not equally important. Adpated from [1].

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' 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 [2].

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 [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:

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

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

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 [6]. 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 [6]

- **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

Artificial Neural Networks are machine learning models with very high representational capacity. [high representation capacity reference] Due to this property, one can think of using them as surrogates for our HPO objective

function f. But the issue is that ANNs 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. [citation]
- Ensemble Approach using monte carlo.
- 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 [4].

2 3 Problem: How can we use DNN to quantify uncertainty in a prediction? 4 5 Solutions Previously: 6 1. Bayesian neural networks. 7 A prior over weights and biases is specified. Given the data, a posterior is calculated. 8 Hard and complex to train. 9 10 2. Ensemble approach. E.g Using Monte

Carlo Dropout during evaluation. (Behaves like an ensemble) 11 Consider 10 predictions in an ensemble. Using these, we could calculate the variance and the 12 mean of the gaussian. Example: Section 3.2 Regression on toy datasets. 13 14 Evaluation of Predictive uncertainties: 15 Scoring rules - Measure quality of predictive uncertainty. 16 Out of distribution uncertainty 17 18 Contributions by authors: 19 Non Bayesian approach. 20 Ensemble of neural networks. 21 Adversarial training for smooth predictions (Adversarial training is optional as it improves the prediction uncertainty quality only in some cases). 22 23 24 Proposed Implementation: 26 p = distribution over real values in regression 27 p = Discrete probablistic distribution for classification. 28 29 3 step approach - Proper scoring rules, adversarial training and ensemble of NNs. 30 31 Understanding proper scoring rules: 32 A scoring function gives a numerical value to the predictive distribution. 33 The better the predictive distribution calibrated to the actual distribution the better the score. 34 Compare p (prediction distribution) and q (original distribution). The closer p and q are the better the score.

Observations: 37 How to combine regression and classification problems in deep ensembles - This is possible by having the correct loss function. For ex ample addition of different parameters. 38 Understand the intuition behind the adversarial examples y = nn(x). We assume that nn is a smooth function hence the small 39 change in x should only have a small change in y. Hence nn(neighbourhood(x)) = y 41 Used a custom NN architecture rather than using 1 hidden layer with 50/100 units depending on the data available. 42 This is because the architecture given in the paper was not helpful. 43 44 Evaluations of confidence vs accuracy. 45 The deep ensembles seem to have a better accuracy with higher confidence as compared to MC-Dropout. 46 47 For Using Deep Ensembles for HPO: 48 We can only use the regression version of the research because the objective scoring is a real valued number 49 How to model discrete input dimensions as inputs into the Deep Ensemble? 50 51 Doubts: 52 Could not understand section 3.5 last paragraph correctly. 53

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

Paper: https://arxiv.org/pdf/2101.07667.pdf

Problem Domain: Hyperparameter searching. Keyword: Use Deep Kernel surrogates

Key points: 1. Use transfer learning surrogates to get faster convergence 2. Treat HPO as a few shot learning problem (In the context of transfer learning) 3. Use kernel k(phi(x), phi(x')) where phi is a neural network that transforms x to a latent vector 4. Learn parameters of phi(W) and kernel(theta) together based on historical meta data 5. Fine tune both W and theta based on the given task. If possible use warm start for initialization. 6. Learn output range variance by creating augumented tasks.

Conceptual points: 1. The deep kernel learnt does not have any task dependent parameters. The task dependent parameters are marginalized out. Hence finetuning during test time (after pretraining) should be complete.

Quick note on expected improvement. First the mean and variance is calculated for the target Hyperparameter Setting. Considering the mean and variance of a particular HP, we calculate the following: For a continous curve we need to integrate. Then we get the Expected Improvement for this HP setting.

Gaussian Processes: The output is considered to be a random variable. When have more than 1 data point, the output becomes a multivariate normal The output is a joint gaussian distribution.

Personal points (Probability vs Likelihood) Probability and likelihood are reverse in nature. Probability starts with a given set of parameters and caculates the probability of a given outcome to occur Likelihood starts with a given outcome, and we would like to determine the parameters.

Marginal likelihood (Likelihood after marginalization of some parameters) integrates the effects of parameters that are not of your interest.

Note: Warm start is not essential for us. (It is not the main contribution of the paper)

Main motive of deep kernel learning: Learn the kernel function in the gaussian process.

How to do implementation with FSBO with HPO-B Metadataset? There

are 16 search spaces (model optmizations here) The train test split is already done metatraindata has the training data and metatest data has the test data. What about validation?? Really required? The search space id can be used for identifying correct search space for test and training data by using:

1 ML model optimization 1 search space (with a unique searchspace id) Dataset 1 (with unique dataset id) Dataset 2 Dataset 3

Pretrain the FSBO model with the train split of the dataset. M' Fine tune it on 1 dataset and evaluate using Expected Improvement i.e run loop like the training with the given data.

Did not use strictGP –¿ Is it a problem?

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.

Main Idea: Given a set of objects, we need to rank the objects. (E.g. Ranking documents based on relevance to a query)

3 types of ranking functions:

Evalution of learnt RF = Ranking measures. Relationship of Ranking measures and RLs is unkown.

Main AIM: Find relationship between Ranking Measuress and Pointwise/Listwise Ranking Losses. Pointsize relationship already clear Goal to do the same for pair wize and listwise case.

Proposed IDEA: Use an essential loss. (Need more reading)

Loss function understanding: Pointwise - Try to get the label as per data Pairwise - Try to separate the labels as much as possible. (Because of -z in all forms of phi). It is a classification of 2 objects with a boundary — Hence the effort to separate things. Listwise - The anology of this is that of 2 oppposing forces. One the rank of the object. Secondly the number of objects below it in the list. Loss ===¿ -rank + number of objects below it. If the rank is low and more number of objects are below it ==¿ Loss ¿ 0 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. Permutation invariance is obtained by using random valid (best case) ==¿ This is true as we are doing K-Layer classfication.

Ranking Losses: Point Pair List Subset regression Ranking SVM ListNET McRank RankBoost ListMLE RankNet

Brief note on Pairwise approach: from the introduction of https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2007-40.pdf The data is created by creating (x1, x2) -¿ label tuples for all possible x1, x2 in the ranked list. The label can be -1, 0, or 1 ,for instance, if x1 is having lower, equal or higher rank to x2 respectively.

Ranking Measures NDCG = K level ratings MAP = 2 level ratings

Read: https://faculty.cc.gatech.edu/ zha/CS8803WST/dcg.pdf Read: https://www.microsoft.com/

us/research/wp-content/uploads/2016/08/letor3.pdf (Page 15 for a clearer picture) CG = Cumulated gain (of information) DCG = Discounted cumulated gain (of information) NDCG = Normalized DCG i.e Divide each position of the DCG by Ideal DCG for the results. You get a list of values in [0,1] which lenght of list = number of rankings considered. NDCG@k = required real valued function of the ranking measure.

Question: Only 2 level rankings necessary for our case?

Understanding listwise loss function: Queries Ranking(f(Q, D)) Ground Truth scores q1 [d1, d2, d3 ... d10] [y1, y2, y3 ... y10] q2 [d1, d2, d3 ... d15] [y1, y2, y3 ... y15] q3 [d1, d2, d3 ... d7] [y1, y2, y3 ... y7]

D = Set of Documents Q = Set of Queries f : QxD -¿ R [Note: Here D is conditioned on Q]. The function is defined for 1 (query, document) pair.

Point to note - Each feature input = a concatenation of the query vector and the document vector.

One instance of our training data is (X, Y) where X = Set of all documents returned by query Y = Set of the respective ground truth scores. Basically 1 query is one instance. Hence our loss function has to take in vector of outputs from f.

Loss = L(Xi , Yi) where Xi and Yi are refer to 1 query qi. Full Batch Loss = mean (L(Xi, Yi) for all elements i in the training objects)

Here f(Q, D) itself is the scoring function that is the main model to be learnt in our framework.

*** Complete explanation found in Section 3 and Section 4 of paper: *** https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2007-40.pdf

* ListNet * We need to make sure that f returns scores that are SIMILAR IN RELAVANCE/ORDER to the ground truth scores. * This would make the Ranking(f(Q, D)) equal to the ground truth and we would have learnt our ranking function. * Note that we do not need to get the exact ground truth scores. Which increases the space of acceptable functions in the function space F (Here f belongs to F) we are searching from 1 to INF. i.e It becomes easier to search if we only want a subspace and not the exact function. * RANKING is nothing but sorting the results based on their respective scores/relavance (decreasing order) * Since the sorting function is non differentiable, the loss function in question should not be composed of the RANKING function. * Moreover, leaving the sorting function makes our loss permutation independent by making use of the + permutation invariant operator [Check the final step for more clarity on this. Cross entropy uses + operator] * This leaves us with 2 lists - a. List of scores given by our ranking function f b. List of relavance scores given to us by ground

truth * The loss function finds the distance of these 2 lists. * A probabilitic approach is taken so as to take into account for any uncertainities. a more relavant document * However the score of the document can negative as well. Hence a strictly positive and increasing funciton phi is taken * The probability of a permutation is nothing but the probability of selecting one document after another without replacement (Reminder: Discrete probability calculation)Which itself becomes a probability distribution i.e sigma = 1 * One possible way to find the distance betwen the 2 lists a and b is to find the probabilities of all permutations for a and b and compare both the distriputions. * Complexity of this O(n!) ==¿ Intractible * Instead take the probabilities of selecting every document first in any possible permutation. Which also is a probability distribution i.e sigma = 1 * The first selection probability (Top 1 probability in the research paper) for a and b are calculated separately using their respective scores. * The final Loss for the list = Cross entropy of probability distribution (a) w.r.t that of b. * This loss is backpropogated through the network to set the parameters. ListMLE: http://icml2008.cs.helsinki.fi/papers/167.pdf * Main difference is that the loss function used is different. * Loss function is intuitive in that they would want to raise the probability of getting the ground truth permutation. * They increase the probability of getting the exact (ground truth) permutation using the scores given by f.

Both the papers use linear network model for some simplicity. But we can use the non-linearity due to available library implementations Very nice summary of both loss functions section 3.2.1 : https://arxiv.org/pdf/1707.05438.pdf

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 other places. So there is an amount of parallelism that can be built it.

Best Model for training and ft giving auful resuts - Reason can be that we are selecting a very unfitted model due to too much variation in the validation losses. Hence we are start only ft best model. Ih the hope that we will get better resutls. only fine tune best fit is not giving good results. Investigation in progress.

2.7 Deep Sets

Chapter 3

Baseline Implementations

We use expected improvement in all models that deal with uncertainty. This is to maintain constant results across all the models. Also due to the advantages of this acquisition function.

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. First fitting the model based on observations. Predicting then the pending configuration for evaluation. Returning the index that gives us the best results.

3.2 Deep Ensemble

Estimator architecuture used.

Enforcing the positivity of the variance as mentioned in the paper. Also adding 1e-6 for numerical stability

3.2.1 Adverserial training and not using it

3.2.2 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.

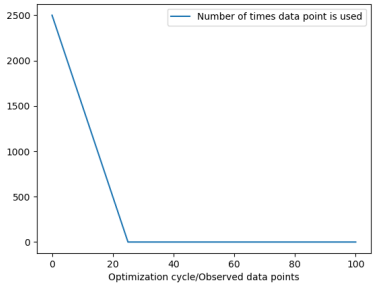

Figure 3.1: Bias at 25th optimization cycle

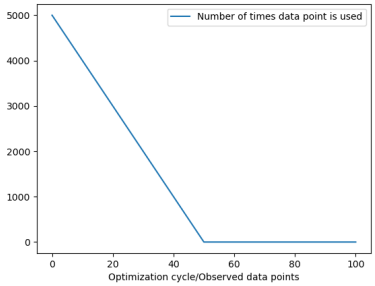

Figure 3.2: Bias at 50th optimization cycle

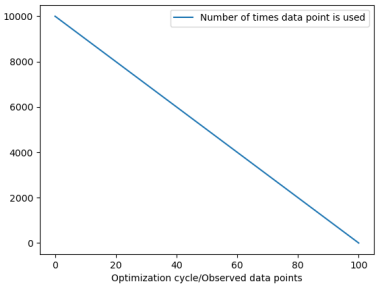

Figure 3.3: Bias at 100th optimization cycle

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. $DE_{restart}$ 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 we may not be able to get out of the local minima. Support using a diagram.

3.3 Few Shot BO: Deep Kernel Learning

Scaling should be ignored because it gives very bad results. if scaling:

3.3.1 Save best model

3.3.2 Early quict

3.3.3 Implementation not in par with that of paper.

3.3.4 Using cosine annehealing

during the implementation to get better fine tune loss curves.

Issues faced during implementions of all models need to be written.

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

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

Documentation (Agenda): First we need to Implement/create the infrastructure to learn ranking loss functions. So we need to read and understand / revise the ranking losses concept.

We decided to model a ranking function using a deep neural network. Reason: reprentation capacity. Creating a toy example to build and run our ranking function Can our scoring function learn to sort a list of numbers?

Obervations after completing the implementation of toy example with list wise ranking function. 1. We do not have any output range of values as a reference. Hence the model can learn score values arbitrarily large 2. 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. Perhaps using other increasing positive function helps? (More reading/research required on this topic) 3. This may make the output domain restrictive. Works for now, however, we may need a better solution for this. 4. The learnt model is extremely sensitive to the learning rate and the number of epochs

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

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)

Possible extensions: Top ranked enhanced loss can be used as it is simple to understand and implement Section 4 in https://arxiv.org/pdf/1707.05438.pdf This is because it is more important to get the correct top rankings than later rankings In fact the top1 rank is what is most important to us in any optimization step of an HPO algorithm.

4.1 case study: Sorting

Check the affet of list / batch size on the results Hence propose using a higher list size.

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.

4.2 Scoring function analysis

4.2.1 Scoring function range

4.3 Optimization cycle

Explain the training loop with an algorithm.

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

Paper = Top-Rank Enhanced Listwise Optimization for Statistical Machine Translation

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.

4.6.1 Independent training

4.6.2 Training with mean and restricted output

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

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.

Paper: https://arxiv.org/pdf/2012.06731.pdf (Impt Ref) https://arxiv.org/pdf/1903.08850.pdf

Key Idea: Use permutation matrices to represent sorting. Learn the matrix with the loss function. [0 1 0] [x] [y] —1 0 0— —y— = —x— [0 0 1] [z] [z] Permutation Matrices are not smooth in the input space Hence relaxation is necessary for differentiability Relaxation is done by using a double schocastic matrices i.e all row and column matrices sum to 1.

Unimodal Double schochastic

The idea of Permutation Matrices is taken from Neural sort - Neural sort (Impt ref)

PLACKETT-LUCE DISTRIBUTIONS -¿ Very important to explain the rationale about using score as a probability measure for our scoring function (Check section 2.1 of paper 2 (Neural Sort))

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] 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