Discrete Simulation Optimization for Tuning Machine Learning Method Hyperparameters

Varun Ramamohan, Shobhit Singhal, Aditya Raj Gupta, Nomesh B Bolia

Department of Mechanical Engineering, Indian Institute of Technology Delhi, India

Abstract

Machine learning methods are being increasingly used in most technical areas such as image recognition, product recommendation, financial analysis, medical diagnosis, and predictive maintenance. The key question that arises is: how do we control the learning process according to our requirement for the problem? Hyperparameter tuning is used to choose the optimal set of hyperparameters for controlling the learning process of a model. Selecting the appropriate hyperparameters directly impacts the performance measure a model. We have used simulation optimization using discrete search methods like ranking and selection methods such as the KN method and stochastic ruler method and its variations for hyperparameter optimization and also developed the theoretical basis for applying common R&S methods. The KN method finds the best possible system with statistical guarantee and stochastic ruler method asymptotically converges to the optimal solution and is also computationally very efficient. We also bookmarked our results with state of art hyperparameter optimization libraries such as *hyperopt* and *mango* and found KN and stochastic ruler to be performing consistently better that Hyperopt Rand and stochastic ruler to be equally efficient in comparison with *hyperopt tpe* in most cases, even when our computational implementations are not yet optimized as par professional packages.

Keywords: Hyperparameter tuning, Simulation optimization, Ranking and selection, Random search

Email address: varunr@mech.iitd.ac.in (Varun Ramamohan)

1. Introduction

Most of the machine learning algorithms are often characterized by multiple parameters that can be selected by the user and are used to modify and control training process of the problem. For example, a simple Support Vector Machine classifier in scikit-learn can be fed with parameters like the kernel type, C- the penalty parameter, gamma and others to control the training process. Similarly, a simple Feed-forward Neural Network can be fed with parameters like the learning rate, learning type, solver function, number of layers, number of nodes in each layer to name a few. The combinatorial space for these hyperparameters becomes so large that arriving at the best set via a brute force enumerative search for a given a problem would be a tedious job and would most of the times lead to non-optimal results. We can very well automate this process of finding the best hyperparameter set for a problem, recently many approaches have been developed for the same. Few prominent approaches being Grid Search, Manual Search and Bayesian optimization (Yang and Shami, 2020).

For a simulation Optimization problem, first the simulation is defined as training the data for multiple iterations on different sets of hyperparameters, generating the performance measure for each hyperparameter, then the associated optimization is performed to find the best performing system. For the optimization part, people have used Grid Search and Random Search methods. Grid Search simulates performance for all the hyperparameter combinations and returns the hyperparameter set with best performance. Though, it performs fairly well but it is computationally very expensive. The Random Search method randomly searches and samples a hyperparameter set from all the possible combinations and simulates its performance. It returns the best system found for a given computational budget. Random search performs better than Grid Search in terms of computational efficiency but it does not provide a statistical guarantee of convergence. Moreover, Bayesian Optimisation gives state-of-the-art performance in simulation optimisation of hyperparameters. But its

time complexity scales cubic with the number of hyperparameters and thus it is impractical to be used for many deep learning models that can have large amount of hyperparameters.

In this paper, for the optimization problem, we propose using both Ranking and Selection and Random Search methods to reach the optimum point. Starting with the former, we first used statistically rigorous Ranking and Selection methods like the KN method (Henderson and Nelson, 2006) for Support Vector Machines and Feed Forward Neural Networks and obtained positive results of convergence to the best performance measure giving us the optimal hyperparameter set for the problem. We then used the Stochastic Ruler method Yan and Mukai (1992), its variant Agrawal et al. (2020) and it's relaxation Alrefaei and Andradóttir (1996) on Multi Layer Perceptron(MLP) Classifiers, LSTM and CNN models and found the algorithm converging asymptotically to the best system, detailed results of which are stated in later sections. Moreover, we have benchmarked our algorithms against popular and state of the art python hyperparameter optimization libraries namely Hyperopt and Mango.

While doing our extensive literature review, we did not find, to the best of our knowledge, any other study which uses simulation optimization methods for hyperparameter tuning problems. In our research work, we introduce a new class of algorithms for hyperparameter tuning, as discussed above, that provide statistical guarantees of convergence in contrast with at least some commonly used existing algorithms (such as hyperopt rand). We also provide the theoretical backing for applying these techniques – especially ranking and selection techniques which require the *iid* property for simulation replication outputs – to ML problems in general. For example, a similar approach can be used in feature selection.

2. Literature Review

Problem of hyperparameter optimisation for machine learning algorithms belongs to a class of non-convex and discrete problems, where traditional optimisa-

tion methods such as gradient descent is less suitable due to non-differentiability and discrete nature. Thus, other methods which can handle discrete nature and general non-convex structures are considered. This optimization problem is also stochastic in nature and this stochasticity is introduced by a large number of permutations of the data set between training and validation set. Thus, discrete optimisation procedures applicable to these problems are discussed.

2.1. Stochastic Optimization methods

2.1.1. Discrete optimization methods

One class of such methods is decision theoretic methods including Random Search (RS) and Grid Search (GS) (Yang and Shami, 2020). GS is one of the most commonly-used methods to explore hyper-parameter configuration space owing to its simplicity and intuitiveness. It exhaustively searches the optimal configuration by evaluating the Cartesian product of a user-specified finite set of values (Alrefaei and Andradóttir, 1996). The demerit of using GS is its inefficiency due to high computational budget requirement.

RS involve moving successively between neighbouring feasible points in search of the optimal solution. It selects a random point from the neighbourhood N(x) of x and uses some criterion to test and evaluate it. Stochastic Ruler Method was first proposed by Yan and Mukai (1992), after that a modification was proposed by Alrefaei and Andradóttir (1996), which reduced the computational time and made convergence faster. Another relaxation of the same was proposed by Agrawal et al. (2020) which showed faster convergence. As the name suggests, we used comparison of the objective function with a scale, the scale here is defined with a uniform random variable around the range in which observations are produced. This methods converges in probability to a global optimum, as shown by Yan and Mukai (1992).

Another class of methods called Ranking and selection procedures are statistically valid methods of comparing a finite number of simulated alternatives (Henderson and Nelson, 2006). They choose the best system or a subset of systems that contains the best system design from a set of competing alterna-

tives (Goldsman and Nelson, 1994). In general, these methods guarantee the probability of optimal selection at some specified confidence level. Bechhofer (1954) proposed the indifference-zone (IZ) formulation of the problem in which an indifference-zone is defined which means that algorithm does not differentiate between alternatives with a score within the indifference zone of the best. Gupta (Gupta, 1965), (Gupta, 1956) proposed the subset selection formulation of the problem. At each iteration in subset selection, we select alternatives satisfying a criteria and reject others. Using these concepts, Kim and Nelson (2001) proposed the KN procedure and the NSGS procedure. These employ the IZ approach and utilize elimination to gain efficiency in case of large number of alternatives.

Further, methods known as Multiple Comparison procedures make use of certain pairwise comparisons to make conclusions in the form of confidence intervals about relationships among all systems (Fu, 1994). In contrast to Ranking and Selection methods, in which the aim is to choose the best, the goal of MCPs is to quantify the differences between systems' performance (not guarantee a decision). Broadly, there are three classes of MPCs. Tukey (1953) proposed the all-pairwise Multiple Comparisons procedure, Dunnett (1955) proposed the Multiple Comparisons with control while Hsu et al. (1984) proposed Multiple Comparison with the best procedure.

Further, metaheuristic algorithms are a set of algorithms mainly inspired by biological theories and widely used for optimization problems which are non-convex, noncontinuous, and non-smooth optimization problems (Yang and Shami, 2020). Population-based optimization algorithms (POAs) are one major type of metaheuristic algorithms, including genetic algorithms (GAs), evolutionary algorithms, evolutionary strategies, and particle swarm optimization (PSO). POAs initialize by creating and updating a population as each generation. Each individual in every generation is then scored until the global optimum is identified. The main differences between different POAs are the methods used to generate and choose populations.

2.1.2. Continuous optimization methods

Gradient-based methods make use of gradient information to estimate direction of decrease and continue this until a local optimal point is reached. Its application to stochastic systems is enabled by stochastic approximation techniques introduced by Robbins and Monro (1951) and Kiefer et al. (1952). More methods of gradient estimation has been developed though the SPSA method by Spall (2003) is revolutionary as it makes the time complexity independent of problem size.

2.2. Current Hyperparameter Tuning methods

Babysitting, also called 'Trial and Error' has been used by researchers due to lack of an efficient hyperparameter optimization framework. This method requires prior experience and is infeasible for problems with large search space.

Further, Grid Search (GS) is one of the first automatic optimization procedures used for hyperparameter optimization. It is a discrete, brute force and exhaustive search method. Though it works well, it is infeasible for large search spaces with expensive model training due to its exponentially increasing computational time with grid dimension.

To counter the brute force nature of GS, application of Random Search to hyperparameter optimization was proposed in Bergstra and Bengio (2012) and shown to be better performing than GS for same computational budget. Though its time complexity is $O(n)$, the search methodology is still independent of the prior observations and thus wastes resources evaluating even the poorer performing regions equally. Moreover, it can be only used on a discrete search space.

To counter some of these limitations, people have largely applied Bayesian Optimization (BO) techniques for optimizing hyperparameters. BO is a powerful framework for minimizing expensive objective functions while using very few function evaluations as it uses previously obtained observations to determine next evaluation points. Snoek et al. (2012) introduced BO for hyperparameter optimization and demonstrated its application for several ML algorithms

surpassing expert level optimization. It uses an acquisition function mostly a Gaussian Process to model the objective function and determines the next evaluation point by predicting the best system accroding to the acquisition function. Acquisition function dynamically changes as we receive function evaluations at different points. Some drawbacks of basyesian optimisation include sensitivity to the choice of acquisition function and other parameters of the surrogate model and cubic time complexity with number of hyperparameters. Moreover, it experieneces difficulty in handling categorical and integer valued hyperparameters.

Furthermore, Population-based Metaheuristic Algorithms (POAs) have been used too. Among these, Genetic Algorithm (GA) and Particle Swarm Optmimzation (PSO) are prominent ones. Lessmann et al. (2005), Hutter et al. (2011) and Peng-Wei Chen et al. (2004) used GA to optimize Support Vector Machines (SVMs). Moreover, Lorenzo et al. (2017) applied PSO to optimize Deep Neural Network hyperparameters.

Further, stochastic gradient based techniques are used for continuous hyperparameters only. Maclaurin et al. (2015) and Pedregosa (2016) showed application of stochastic gradient descent for optimizing continuous hyperparameters of various machine learning algorithms.

2.2.1. Hyperparameter tuning packages

There exists some well optimised python libraries for hyperparameter optimisation of machine learning models. One of them is HyperOpt, which implements Tree of Parzen Estimators which is a type of Bayesian Optimisation method. It also implements pure Random Search (Bergstra et al., 2013). Moreover, Mango is another python library that implements a batch Gaussian process bandit search using upper confidence bound as the acquisition function. It achieves state-of-the-art performance (Sandha et al., 2020).

We have already listed the research contributions in the introduction section.

3. Theoretical framework

Consider a machine learning problem where we are attempting to find the best-fit function f that predicts an outcome y as a function of features x. Here $x \epsilon R^n$, $y \epsilon R^n$. Let the training data-set be denoted by (X, Y), where X is an $m \times n$ matrix and Y is an m-dimensional vector of labels. Let $\theta \epsilon R^d$, be a set of d hyperparameters that characterize the training process, and let P be the performance measure used to judge the quality of the fit of $f(x)$ (increasing P denotes increasing quality of fit). Typically, P is a random variable whose value depends upon the organization of the training + test data-set, and hence hyperparameter tuning attempts to find θ that maximizes $E[P]$. $E[P]$ may be determined, for example, via a cross-validation exercise. We propose to use simulation optimization methods – more specifically, ranking and selection methods such as the KN method Henderson and Nelson (2006) and Random Search methods like Stochastic Ruler method Yan and Mukai (1992), its variant Agrawal et al. (2020) and it's relaxation Alrefaei and Andradóttir (1996) - for selecting optimal hyperparameter values.

3.1. Simulation Algorithm

In order to apply simulation optimization methods to find optimal hyperparameter values, we first need to define the corresponding 'simulation' and the associated optimization problem. We propose considering the process of finding the optimal parameter estimates of the function f given a particular organization of the training set $(X, Y)_i$ and a particular k_{th} set of hyperparameters θ_k, as the i_{th} replication of the simulation. The particular version of the training set used in the i_{th} replication is generated by a single permutation of the data-set, denoted by $\sigma_i(X, Y)$. Each permutation can be considered to be a random mapping - i.e., if the set of indices of the m points in the training set is denoted by $M = \{1, 2,..., m\}$, then the permutation σ randomly samples (without replacement) an index from the training set and maps it to the set M. The permuted data-set $(X, Y)_i$ can then be divided into the training and holdout validation

data-sets (e.g., first 80% used for training, next 20% for holdout validation). Thus output of one replication (e.g., the i_{th} replication) of this simulation can be considered to be the performance measure $P_i(\theta_k)$, denoted in short as P_{ki}. The simulation algorithm can be written as follows:

Algorithm 1: Simulation algorithm for generating an estimate of $E[P]$ for a given hyperparameter set θ_k.

Initialize with data-set (X, Y), I = number of replications, θ_k,

$P(\theta_k) = 0$.

Set random number seed $s_k \sim g_s(s)$.

for $i = 1 - I$ **do**

  $(X, Y)_i = \{\}$, $M = \{1, 2, ..., \mathrm{m}\}$, $J = |M|$

  for $j = 1 - J$ **do**

    $m \leftarrow |M|$

    $l \leftarrow U(0, 1) \times m$

    $(X, Y)_i \leftarrow (X, Y)_i + (x_l, y_l)$

    $M \leftarrow M - l$

  end

  $P(\theta_k) = P(\theta_k) + fit((X, Y)_i, f, \theta_k)$

end

$\overline{P(\theta_k)} \leftarrow \frac{P(\theta_k)}{I}$

In the Algorithm 1, the term $fit((X, Y)_i, f)$ can be thought as the training and validation subroutine that takes as input the i_{th} permutation of the data-set (X, Y), the function to be fit f and the hyperparameters θ_k, and outputs the performance measure P_{ki} for the data-set $(X, Y)_i$.

3.2. Optimization Problem

The optimization problem can thus be written as follows:

$$\min_{\theta \epsilon S} E[P(\theta)]$$

Here S represents the set of allowable values for θ. $E[P(\theta)]$ is estimated as $P(\theta)$ by the simulation described above.

3.3. IID Requirements

In order to apply common ranking and selection methods, and random search methods such as the KN or NSGS Henderson and Nelson (2006), and Stochastic Ruler procedures Yan and Mukai (1992) respectfully, a standard *iid* requirement must be satisfied by the simulation replications associated with a given hyper-parameter setting θ_k. That is, the I replications $P_{k1}, P_{k2}, ..., P_{kI}$ must be *iid*. Note that, if $|S| = K$ (that is, there are K allowable values of θ, some procedures such as the NSGS method impose the additional requirement that the random variables $P(\theta_k)$, $k = (1, 2, ..., K)$ are also independent; however, other methods such as the KN method do not, as they permit the use of common random numbers Henderson and Nelson (2006). Hence our focus in next section will be the *iid* requirement that the P_{ki}, $i = 1 - I$ have to satisfy.

For typical machine learning exercises, the samples in the dataset (X, Y) are assumed to be *iid*. However, this is typically difficult to verify, especially if the analyst is not involved in the generation of the data-set. In such situations, the operational definition of *iid* property to be applied comes from the class of representation theorems summarized in Ressel (1985). Broadly, according to these theorems, a sequence (typically infinite) of random variables $X_1, X_2, ...$ which may not be independent in themselves can be shown to be conditionally independent given a realization of an underlying random variable as long as they are exchangeable. The key requirement here is exchangeability - that is, the joint distribution of the sample must not change if the order in which the X_i are generated changes Ressel (1985).

In order to demonstrate the applicability of this class of theorems to our case, we must identify the underlying distribution that generates the particular set of replications $P_{ki}, i = 1$-I and then demonstrate that these replicate values of P_{ki} are exchangeable.

Each value of P_{ki} is a function of the i_{th} permutation of the training set. Each permutation of the training set $(X, Y)_i$ can be considered to be governed by the sequence of uniform random numbers that are used to sample (without replacement) from the training set (X, Y). For example, per algorithm 1, the

i_{th} permuted dataset $(X, Y)_i$ is generated by the sequence of random numbers $U_i = \{U_{i1}, U_{i2}, ..., U_{iJ}\}$. However, the U_i are a subsequence of the stream of pseudorandom numbers generated for the entire simulation in algorithm 1, which can be thought of as a sequence in itself, given by $U = \{U_1, U_2, ..., U_I\}$. This sequence is governed by a specific seed s_k fed to the random number generator at the start of the simulation. That is, if we denote the random number seed used to generate U as s_k, then U in effect becomes a function of s_k. Now, if the s_k are themselves generated from some distribution $g_S(s)$ - for example the s_k are randomly sampled (with replacement) integers from 1-L, where L is a very large number when compared to K to minimize the likelihood of sampling the same s_k twice or more - then the underlying distribution of the P_{ki} becomes this uniform random integer distribution. Thus we have constructed the underlying distribution of the P_{ki}.

Exchangeability is easy to demonstrate, given that each P_{ki} is a function of the random number seed s_k. Therefore, the joint distribution of the P_{ki} depends only on the distribution of the s_k, and not on the order in which the P_{ki} are generated. Now it can be argued that the P_{ki} become a deterministic sequence once s_k has been sampled; however, this can easily be countered by introducing some additional stochasticity into the generation of each P_{ki} - for example, by initializing the parameters of the machine learning algorithm f by sampling from some distribution in each replication. This completes the argument.

3.4. Benchmarking

To compare our algorithms with current off the shelf hyperparameter optimization libraries, we benchmarked our results with 2 very popular python libraries. We have used Student's T test as a test of significance of equality of means.

3.4.1. Benchmarking Methodology

To compare one optimization algorithm with another, we use Student's t-test. Let $h_{1,i}$ be the hyperparameter set given by algorithm 1 and let $h_{2,i}$ be

the hyperparameter set given by algorithm 2 in i^{th} replication. Let n be the total no. of function evaluations. The null and alternate hypothesis is given by

$$H_0 : \mu_1 = \mu_2$$

$$H_1 : \mu_1 \neq \mu_2$$

And the t-statistic is given by

$$\mu_j = \frac{\sum_{i=1}^{n} h_{j,i}}{n} \tag{1}$$

$$s_j^2 = \frac{\sum_{i=1}^{n} (h_{j,i} - \mu_j)^2}{n-1} \tag{2}$$

$$T = \frac{\mu_1 - \mu_2}{\sqrt{(s_1^2 + s_2^2)/n}} \tag{3}$$

For 95% significance level, critical t-value is ≈ 2 and for 90% confidence interval, it is ≈ 1.7 for sufficiently large replications.

3.4.2. Framework 1: Hyperopt

Hyperopt is a well known python library for machine learning hyperparameter tuning which implements Bayesian optimization and random search Bergstra et al. (2013). The bayesian optimization works on updating a probability model such as gaussian processes, based on the replications of performance measure observed for different hyperparameter sets. And outputs the set for which $\underset{max}{score}$ P(score — hyperparameter set) is maximum. The bayesian optimisation in hyperopt implements an algorithm called Tree of Parzen Estimators as probability model. It estimates a GMM (Gaussian Mixture Model) for both best and worst models, and tries to maximize the ratio of output of both.

While random search randomly searches for hyperparameter set configurations and stores performance measure replication. It outputs the system for which maximum score is found for the finite budget of function evaluations.

3.4.3. Framework 2: Mango

Mango is a recent python library which implements a state of the art optimization algorithm based on batch Gaussian process bandit search using upper confidence bound as the acquisition function Sandha et al. (2020). It implements adaptive exploitation vs. exploitation trade-off as a function of search space size, number of evaluations, and parallel batch size.

We have shown the benchmarking results with respect to KN and SR in 4.4 and 5.3 respectively.

4. Ranking and Selection method: KN for Hyperparameter Tuning

Most of the hyper-parameters for Machine Learning algorithms are discrete valued, so for hyper-parameter optimization, we started stochastic simulation with discrete optimization methods. Ranking and Selection is a class of discrete optimization methods, which keep comparing the outcomes at each stage and eliminates the outcome if it is clearly not the contender for the best performer. In KN method of Ranking and Selection class Henderson and Nelson (2006), we define the criterion for comparison which is discussed in the procedure below. If the outcome can not be clearly rejected, then we keep those values in the Indifference Zone(IZ). We keep on generating the results, till the number of systems left in the IZ is 1, that is the single best system.

In KN method, the number of evaluations of the ML model are large as we are executing the algorithm one more time for each replication, and the evaluations are expensive. So, we use KN method for cases where the hyperparameter space is relatively small, and generating replicate estimates of the performance measure is not too expensive, as the cases we have taken in 4.2.

4.1. Procedure

The procedure of KN method is as follows Henderson and Nelson (2006):

1. Setup: First the overall desired PCS (Probability of Correct Selection), $1-\alpha$, IZ(indifference zone) parameter δ and common first-stage sample size $n_o \geq 2$ are selected. Here k is the total number of systems.

$$\eta = \frac{1}{2}[\frac{2\alpha}{k-1}^{\frac{-2}{n_o-1}} - 1]$$

2. Initialization: Let $I = \{1, 2, ..., k\}$ be the set of systems still in contention, and let $h^2 = 2\eta(n_o - 1)$.

   Obtain n_o outputs $X_{ij}, j = 1, 2, ..., n_o$, from each system $i, i = 1, 2, ..., k$, and,

   Let $X_i(n_o) = n_o^{-1} \sum_{j=1}^{n_o} X_{ij}$ denotes the sample mean of the first n_o outputs from system i.

   For all $i \neq l$ calculate:

   $$S_{il}^2 = \frac{1}{n_o-1} \sum_{j=1}^{n_o} (X_{ij} - X_{jl} - [X_i(n_o) - X_l(n_o)])^2$$

   the sample variance of the difference between systems i and l.

   Set $r = n_o$.

3. Screening: Set $I^{old} = I$.

   Let

   $$I = \{i : i \in I^{old} \text{ and } X_i(r) \geq X_l(r) - W_{il}(r), \forall l \in I^{old}, l \neq i\}$$

   where, $W_{il}(r) = max\{0, \frac{\delta}{2r}(\frac{h^2 S_{il}^2}{\delta^2} - r)\}$

4. Stopping rule: If $|I| = 1$, then stop and select the system whose index is in I as the best.

   Otherwise, take one additional output $X_{i,r+1}$ from each system $i \in I$, set $r = r + 1$ and go to Screening.

4.2. Application and Results

We applied the KN method on Support Vector Machines and Feed-Forward Neural Network(MLP Classifier) models for testing our methods. As stated earlier, the number of function evaluations in KN becomes large, so we choose the models for KN in which the number of hyperparameters are small and also the computational cost for each evaluation is not too expensive. We used a dataset from Scikit-learn repository titled 'Breast Cancer Wisconsin Dataset' containing 569 samples in total with 30 features. The problem is to build a machine

learning model that can classify such samples between two categories 'malignant' and 'benign'. Our objective is to find optimal hyperparameter estimates corresponding to the best performance.

For executing all these problems and tests, we used public version of Google Colaboratory, with approximate specifications of CPU with 2.2GHz clock cycle and 12GB RAM.

1. Support Vector Machine (SVM)
   Dataset used: Breast cancer from sklearn datasets

   Hyperparameters of SVM Model:
   kernel $\in \{'rbf', 'linear'\}$
   gamma $\in \{0.001, 0.01, 0.1, 0.5, 1, 10, 30, 50, 80, 100\}$
   C $\in \{0.01, 0.1, 1, 10, 100, 300, 500, 700, 800, 1000\}$

   KN Method parameters (as defined in Section 4.1):
   $k = 200$, $n_o = 10$,
   $PCS = 95\%$, $\delta = 10$ %

   Results:
   Accuracy value of the best system found: 0.968
   No. of function evaluations taken: 2001
   Total run-time: 6709s
   No. of iterations(r value): 10

2. Feed-forward Neural Networks (MLP Classifiers)
   Dataset used: Breast cancer from sklearn datasets

   Hyperparameters:
   Hidden layer sizes $\in \{(3,), (5,), (8,), (12,), (15,), (20,), (25,), (30,), (50,), (80,)\}$
   Activation function $\in$ {relu, logistic, tanh}

Solver ∈ {adam, sgd}

Learning rate type ∈ {constant, adaptive}

Learning rate value (Discretized) ∈ {0.0005,0.001,0.01,0.05,0.1}

KN Method parameters(as defined in Section 4.1):

k= 600, $n_o = 10$

$PCS = 95\%$, $\delta = 10$ %

Results:

Accuracy Value of the best system found: 0.934

No. of function evaluations taken: 9871

Total run-time: 3093s

No. of iterations(r value): 195

We see in the results that in the Ranking and Selection method, KN procedure converges asymptotically to the optimum hyperparameter set for both Support Vector Machine and Feed-Forward Neural Network, but, a drawback of these methods is that it requires complete enumeration of the solution space, which makes it costly.

4.3. KN Method Parameters

We are using KN methods for finding the best performing system and thus the optimal hyper-parameter system, but KN itself takes α (1 - PCS) and δ (IZ(indifference zone parameter) as its parameters, which has to be carefully selected. Values of α and δ considerably effects the performance of KN by effecting the IZ zone formed. α is 1- PCS, which is the Probability of Correct Selection. PCS is generally best around 95% and value of δ is to be adjusted.

For the tests on α and δ, we use the same problem statement defined in 4.2. Our objective is to find the relation between different values of r (No. of iterations required to reach the optimal system) and δ. We used the Feed-forward

Neural Networks (MLP Classifiers) for this exercise with the following hyper-parameter sets:

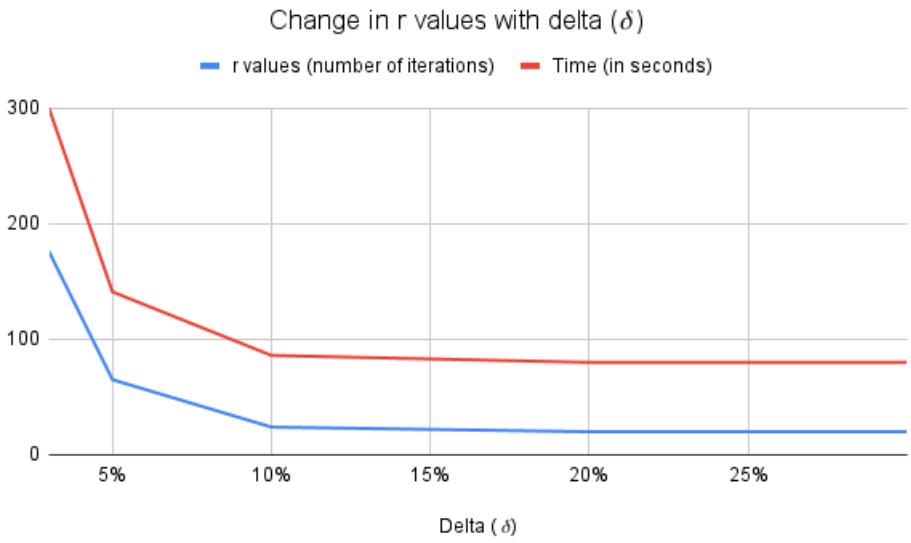
Hidden layer sizes $\in \{(3,), (10,), (25,), (50,), (80,)\}$
Learning rate value (Discretized) $\in$ {0.0005,0.001,0.01}
Activation function $\in$ {relu}
Solver $\in$ {sgd}
Learning rate type $\in$ {adaptive}

We have performed the KN procedure with different value of δ and as expected, it can be seen from Chart 1 that for lower values of δ higher values of r, the number of iterations, is required. Also, Table 1 shows the evolution of IZ zone for $\alpha = 5\%$ and $\delta = 3\%$. The accuracy values of the systems in IZ zone keeps updating by small amount with each iteration as the number of evaluations keep increasing for each system to factor for randomness, as described in Section 4.1.

Table 1: IZ zone and number of function evaluations for $\alpha = 5\%$ *and* $\delta = 3\%$

r value	Function evals	IZ Zone	Run-time
20	301	0.63, 0.76, 0.86, 0.72, 0.90, 0.72	83s
32	373	0.75, 0.88, 0.74, 0.89, 0.86	120s
33	378	0.76, 0.88, 0.73, 0.89	122s
51	450	0.86, 0.72, 0.88	150s
57	468	0.86, 0.88	160s
176	706	0.88	300s

Figure 1: Change in r values with δ



4.4. Benchmarking KN with Hyperopt Random Search t-test

As discussed in section 3.4, we benchmarked the results of KN with Hyperopt Random library. Table 2 contains the benchmarking results of KN w.r.t. best system returned by Hyperopt Rand for different number of function evaluations. To compare the actual performance of the best systems returned, we executed the model with the best systems obtained 10 time and took it's mean to average out any randomness. The results clearly shows that KN performs far better than almost all the cases of Hyperopt Rand with statistical guarantee. Hyperopt Rand doesn't factor randomness while searching the best system, so the performance of the system, when taken average for multiple iterations, drastically gets reduced from expected. Also, increasing the number of function evaluation, within ranges of normal computational power, doesn't insure that we will get better result from Hyperopt Rand. Therefore, we can conclude that KN is performing much better when benchmarked against Hyperopt Rand and gives the best system with statistical guarantee.

Table 2: Benchmarking of best system returned by KN w.r.t. best system returned by Hyperopt Rand for different number of function evaluations

	KN	Hyoeropt Rand	Hyoeropt Rand	Hyoeropt Rand	Hyoeropt Rand	Hyoeropt Rand
No. of Function Evaluations	Takes 1381 evals to reach best system	Is ran 1000 evals	Is ran 500 evals	Is ran 300 evals	Is ran 200 evals	Is ran 100 evals
Run-time	635s	463s	226s	123s	81s	42s
Accuracy value for best system returned	0.940	0.982	0.982	0.982	0.973	0.964
t-test Results						
Mean accuracy value of given best system for 10 function replications	0.927	0.787	0.891	0.736	0.887	0.934
t-value	Compared to KN	1.443	0.476	2.076	0.508	-0.116
Result	Base	KN is better	KN is better	KN is better	KN is better	Hyperopt is better

5. Random Search method: Stochastic Ruler for Hyperparameter Tuning

Yan and Mukai proposed a method for discrete stochastic optimization called the 'Stochastic Ruler Method'. This method involves comparing the observed objective function values $H(x)$, where $X \epsilon S$, with a uniform random variable (the stochastic ruler) defined on the range of the observed objective function values. This uniform random variable is used as a scale against which the observations $H(x)$ are measured. Yan and Mukai show that their method converges in probability to a global optimal solution. In this paper, we use the modifications in the original version proposed by Agrawal et al. (2020) and Alrefaei and Andradóttir (1996). KN method is computationally expensive for high dimensional search spaces. Though, it guarantees best system, we are not always interested in the best, but an improvement. Stochastic ruler method continuously finds improvement with time and can be stopped when sufficient improvement is achieved or computational budget is exhausted. Unlike KN method, as it enumerates the whole search space in the first iteration. In this section, there are two types of stopping criterion applied one at a time. First is a computational budget applied in terms of time for each experiment. This is done to mimic a realistic setting where these algorithms can be applied. In a real scenario, there is no knowledge of the best performance but there are time and computational budget constraints, thus it is natural to stop the algorithm after some specified time duration. Secondly, we impose a specific performance to be reached. This performance measure is found out by applying KN method, which provides some guarantee on the best system. This stopping criterion is applied to examine if Stochastic ruler will reach the optimal given sufficient time.

5.1. Procedure

Definitions:

1. Sample space (S): Set of all feasible points
2. Neighborhood of x, ($N(x)$): For each $x \epsilon S$, there exists a set $N(x), N(x) \subseteq S\{x\}$, called the set of neighbours of x.

3. $\theta(a, b)$: A uniform random sample defined on the interval $[a, b]$, which covers the while range of $H(x)$
4. $H(x)$: The objective function which is a random variable

Procedure:

1. Select a starting point $X_o \epsilon S$. Let $V_o(X_o) = 1$, and $V_o(x) = 0, \forall x \epsilon S\{X_o\}$. Let $k = 0$ and $X_k^* = X_o$.
2. Given $X_k = x$, choose a candidate Z_k from $N(x)$ with probability distribution

$$P\{Z_k = z; X_k = x\} = \frac{1}{|N(x)|}$$

where $z \epsilon N(x)$

3. Given $Z_k = z$, draw a sample $h(z)$ from $H(z)$. Then draw a sample θ from $\theta(a, b)$. If $h(z) > \theta$, then let $X_{k+1} = X_k$ and go to step 4.
Otherwise, draw another sample $h(z)$ from $H(z)$ and draw another sample θ from $\theta(a, b)$. If $h(z) > \theta$, then let $X_{k+1} = X_k$ and go to step 4.
Otherwise continue to draw and compare. If all M tests, $h(z) > \theta$, fail, then accept the candidate Z_k and set $X_{k+1} = Z_k = z$.
4. Let $k = k+1, V_k(X_k) = V_{k-1}(X_k)+1$, and $V_k(x) = V_{k-1}(x), \forall x \epsilon S, x \neq X_k$. If

$$\frac{V_k(X_k)}{D(X_k)} > \frac{V_k(X_{k-1}^*)}{D(X_{k-1}^*)}$$

then let $X_k^* = X_k$; otherwise let $X_k^* = X_{k-1}^*$.
Go to step 2

Stop until a computational budget is reached or sufficient improvement is observed.

5.2. Application and Results

To test this algorithm, we took a set of machine learning problems which included different types of problems namely classification, image classification and time-series regression. We experimented with different types and size of search space and stopping criterion with MLP. Stages refers to stochastic ruler

iterations k in the procedure 5.1. In these problems, we chose the value of a and b based on our idea of accuracy or loss values specific to the ML models used.

For executing all these problems and tests, we used public version of Google Colaboratory, with approximate specifications of CPU with 2.2GHz clock cycle and 12GB RAM.

1. In this section, we used a Feed-forward Neural Network (MLP Classifiers) to solve cancer classification problem as defined in 4.2. We apply the optimal performance stopping criterion here and the results are summarized in table 3. Our objective is to find optimal hyperparameter set.

   Hyperparameters:
   Neurons in hidden layer 1 $\in \{2, 4, 6, 8, 10\}$
   Neurons in hidden layer 2 $\in \{1, 2, 3, 4, 5\}$
   Learning rate $\in \{0.001, 0.004, 0.007, 0.01\}$

   Stochastic Ruler procedure parameters:
   $N(X_k) = \{X \epsilon S; X_i = X_{k+j,i}; X \neq X_k\}, j \in \{-1, 0, 1\}$
   $a = 0, b = 1$

Table 3: Average stages and run-time for connvergence MLP-1

M	α	Stages	Run-time
5	1	1227	332s
5	0.8	823	391s
$\frac{\ln(k+10)}{\ln(5))}$	1	1354	285s
$\frac{\ln(k+10)}{\ln(5)}$	0.8	1017	212s

2. We applied Stochastic Ruler on MLP NN with a larger search space. We noted the mean improvement observed within the computational budget stopping criterion, as summarized in table 4.

   Hyperparameters:
   Neurons in hidden layer 2 $\in \{2, 4, 6, 8, 10\}$
   Neurons in hidden layer 1 $\in \{1, 2, 3, 4, 5\}$
   Learning rate $\in \{1e{-}6, 5e{-}6, 1e{-}5, 4e{-}5, 7e{-}5, 1e{-}4, 4e{-}4, 7e{-}4, 1e{-}3\}$

   Stochastic Ruler procedure parameters:
   $N(X_k) = \{X \epsilon S; X_i = X_{k+j,i}; X \neq X_k\}, j \in \{-1, 0, 1\}$
   $a = 0, b = 1$

Table 4: Average stages and run-time for convergence MLP-2

M	α	Stages	Run-time	Mean improvement	SD (Mean improvement)
5	1	2051	1502s	0.5005	0.048
5	0.8	1536	1500s	0.4559	0.119
5	0.6	1401	1501s	0.3622	0.126
$\frac{\ln(k+10)}{\ln(5))}$	1	2699	1500s	0.4982	0.083
$\frac{\ln(k+10)}{\ln(5)}$	0.8	2649	1500s	0.473	0.107
$\frac{\ln(k+10)}{\ln(5)}$	0.6	1946	1503s	0.4999	0.084

3. Using a different hyperparameter space for optimizing MLP NN with Stochastic Ruler. We use the optimal performance stopping criterion and the results are summarized in table 5.

Hyperparameters:

Neurons in hidden layer 2 $\in \{2, 4, 6, 8, 10\}$

Neurons in hidden layer 1 $\in \{1, 2, 3, 4, 5\}$

Learning rate $\in \{0.001, 0.004, 0.007, 0.01\}$

Activation function $\in$ {relu, tanh, logistic}

Learning rate type $\in$ {constant, invscaling, adaptive}

Stochastic Ruler procedure parameters:

$N(X_k) = \{X \in S; X \neq X_k\}$

$a = 0, b = 1$

Table 5: Average stages and run-time for convergence MLP-3

M	α	Stages	Run-time
5	1	2244	1111s
5	0.8	1901	1023s
$\frac{\ln(k+10)}{\ln(5))}$	1	3273	1463s
$\frac{\ln(k+10)}{\ln(5)}$	0.8	2712	1045s

4. Running Stochastic Ruler on MLP NN with only discrete type hyper-parameter space. We noted the mean improvement observed within the computational budget stopping criterion, as summarized in table 6.

Hyperparameters:

Neurons in hidden layer 2 $\in \{2, 4, 6, 8, 10\}$

Neurons in hidden layer 1 $\in \{1, 2, 3, 4, 5\}$

Stochastic Ruler procedure parameters:

$N(X_k) = \{X \epsilon S; X_i = X_{k+j,i}; X \neq X_k\}, j \in \{-1, 0, 1\}$
$a = 0, b = 1$

Table 6: Average stages and run-time for convergence MLP-4

M	α	Stages	Run-time	Mean improvement (MI)	SD (MI)
5	1	401	250s	0.5429	0.018
5	0.8	278	250s	0.5599	0.013
5	0.6	347	250s	0.5096	0.086
$\frac{\ln(k+10)}{\ln(5))}$	1	599	250s	0.5747	0.074
$\frac{\ln(k+10)}{\ln(5)}$	0.8	620	250s	0.5334	0.024
$\frac{\ln(k+10)}{\ln(5)}$	0.6	585	250s	0.5581	0.065

5. For testing the results in time-series prediction problems, we applied Stochastic Ruler on LSTM to solve stock price prediction problem as defined below. We use the optimal performance stopping criterion and the results are summarized in table 7.

   Problem Statement:
   For testing our methods on LSTMs, we used Stock prices of "TATA Global" at NSE, starting from 21st July 2010 to total 700 data points. We used the opening Prices for each day for training and testing the model. We used stock prices of last 30 days to predict the price of the 31st day, sequentially for all the 500 training points, and then tested it on 200 testing points.

   Hyperparameters:

Neurons per layer $\in \{10, 50\}$

Dropout Rate $\in \{0.2, 0.5\}$

Epochs $\in \{5,10,50\}$

Batch Size $\in \{47, 97, 470\}$

Stochastic Ruler procedure parameters:

$N(X_k) = \{X \epsilon S; X_i = X_{k+j,i}; X \neq X_k\}, j \in \{-1, 0, 1\}$

$a = 102, b = 108$

Table 7: Average stages and run-time for connvergence LSTM

M	α	Stages	Run-time
5	1	91	818s
5	0.8	23	381s
$\frac{\ln(k+10)}{\ln(5))}$	1	13	88s
$\frac{\ln(k+10)}{\ln(5)}$	0.8	13	83s

6. We also tested Stochastic Ruler Procedure on CNN model using 'mnist' data set as defined below. We use the optimal performance stopping criterion and the results are summarized in table 8.

   Problem Statement:

   For CNN, we used "MNIST" data-set from scikit-learn repository. We used 400 samples for training our model and 100 samples for testing it. The problem was to identify the digit written on an image of 28×28 pixels.

   Hyperparameters:

   Neurons per layer $\in \{32, 64\}$

Table 8: Average stages and run-time for convergence CNN

M	α	Stages	Run-time
5	1	16	101s
5	0.8	6	55s
$\frac{\ln(k+10)}{\ln(5))}$	1	10	83s
$\frac{\ln(k+10)}{\ln(5)}$	0.8	9	80s

Epochs $\in$ {3,5}

Batch Size $\in$ {50, 100, 400, 800}

Stochastic Ruler procedure parameters:

$N(X_k) = \{X \epsilon S; X_i = X_{k+j,i}; X \neq X_k\}, j \in \{-1, 0, 1\}$

$a = 0, b = 0.08$

For the Random Search method, both the Original Stochastic Ruler method and the modified version, converges to the optimum solution corresponding to the best performance measure.

Many other discrete methods exist (e.g., COMPASS), but this is a demonstration of how random search can be applied to HPT, and that other methods can be applied in a similar manner to how we have applied the stochastic ruler method.

5.3. Benchmarking Stochastic Ruler results

We test out algorithms in comparison with HyperOpt and Mango packages. HyperOpt TPE is the Tree of Parzen Estimators algorithm of the HyperOpt package. It is a Bayesian optimization based technique. HyperOpt Rand is the pure Random Search method of HyperOpt package. HyperOpt Rand Contin-

uous doesn't discretize the search space for continuous hyperparameters. As expected, it performs better than its discrete version HyperOpt Rand.

We conduct t-test on 10 replications of Stochastic ruler for different random seeds. Moreover, the function evaluation is also stochastic, thus we evaluate a system of hyperparameters 25 times and calculate t value. We use confidence level of 95%. If t value is greater than the critical t value for $n = 25$, we reject the null hypothesis of both system performing equally well. Moreover, we also report the system that performed better in the replications where null hypothesis were rejected.

5.3.1. Benchmarking Stochastic Ruler-1

Stochastic Ruler Parameters:

$M = \frac{\ln(k+10)}{\ln(5))}$

$\alpha = 1$

$a = -1, b = -0.7$

Hyperparameters:

Neurons in hidden layer 2 $\in \{2, 4, 6, 8, 10\}$

Neurons in hidden layer 1 $\in \{1, 2, 3, 4, 5\}$

Learning rate $\in \{1e-6, 5e-6, 1e-5, 4e-5, 7e-5, 1e-4, 4e-4, 7e-4, 1e-3\}$

The results are summarised in table 9.

5.3.2. Benchmarking Stochastic Ruler-2

Stochastic Ruler Parameters:

$M = \frac{\ln(k+10)}{\ln(5))}$

Table 9: Benchmarking of Stochastic Ruler (b=-0.7) vs Hyperopt tpe for different number of function evaluations

Case	Function Evaluations	Null rejected	Performance
SR and Hpt TPE	100	3/10	Hyperopt Better
	200	0/10	Both Well
	300	0/10	Both Well
SR and Hpt Rand	100	7/10	SR Better
	200	10/10	SR Better
SR and Hpt Rand Cont	100	2/10	Hpt Better
	200	0/10	Both well
SR and Mango	200	0/10	Both well

$\alpha = 1$

$a = -1, b = -0.5$

Hyperparameters are same as above. The results are summarized in table 10.

Table 10: Benchmarking of Stochastic Ruler (b=-0.5) vs Hyperopt tpe for different number of function evaluations

Case	Function Evaluations	Null rejected	Performance
SR and Hpt TPE	100	4/10	Hpt Better
	200	2/10	Hpt Better
	300	0/10	Both Well
SR and Hpt Rand	100	6/10	SR Better
	200	8/10	SR Better
SR and Hpt Rand Cont	100	4/10	Hpt Better
	200	3/10	SR Better
SR and Mango	200	2/10	Mango Better
	300	0/10	Both well

6. Conclusion

Hyperparameter Tuning of Machine Learning algorithms is a subject of interest as the accuracy and generalizability of these models strongly depends on these hyperparameters and choosing them manually is not a trivial task as their can be numerous hyperparameters and large set of possible values.

The Ranking and Selection methods provide a method with good guarantee of finding the best system though it is inefficient as it evaluates the whole search space. The Random Search methods, specifically the Stochastic Ruler method is quite efficient in terms of computational requirements and asymptotically converges to the optimal.

Moreover, we benchmarked our algorithms against Hyperopt and Mango and found KN to perform consistently better that Hyperopt Rand and Stochastic ruler to be equally efficient to tpe method of Hyperopt in most cases. And as expected, Stochastic Ruler performs significantly better than Hyperopt random search. Moreover, our implementation of Stochastic ruler is not computationally optimized at par with professional packages.

References

References

Agrawal U, Goyal M, Ramamohan V. A relaxation of the stochastic ruler method for discrete simulation optimization. arXiv preprint arXiv:201006909 2020;.

Alrefaei MH, Andradóttir S. Discrete stochastic optimization via a modification of the stochastic ruler method. In: Proceedings Winter Simulation Conference. IEEE; 1996. p. 406–11.

Bechhofer RE. A single-sample multiple decision procedure for ranking means of normal populations with known variances. The Annals of Mathematical Statistics 1954;:16–39.

Bergstra J, Bengio Y. Random search for hyper-parameter optimization. The Journal of Machine Learning Research 2012;13(1):281–305.

Bergstra J, Yamins D, Cox D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: International conference on machine learning. PMLR; 2013. p. 115–23.

Dunnett CW. A multiple comparison procedure for comparing several treatments with a control. Journal of the American Statistical Association 1955;50(272):1096–121.

Fu MC. Optimization via simulation: A review. Annals of operations research 1994;53(1):199–247.

Goldsman D, Nelson BL. Ranking, selection and multiple comparisons in computer simulation. In: Proceedings of Winter Simulation Conference. IEEE; 1994. p. 192–9.

Gupta S. On a decision rule for a problem in ranking means: Mimeograph series (150). Institute of Statistics, University of North Carolina, Chapel Hill, NC 1956;.

Gupta SS. On some multiple decision (selection and ranking) rules. Technometrics 1965;7(2):225–45.

Henderson SG, Nelson BL. Handbooks in operations research and management science: simulation. Elsevier, 2006.

Hsu JC, et al. Constrained simultaneous confidence intervals for multiple comparisons with the best. The Annals of Statistics 1984;12(3):1136–44.

Hutter F, Hoos HH, Leyton-Brown K. Sequential model-based optimization for general algorithm configuration. In: International conference on learning and intelligent optimization. Springer; 2011. p. 507–23.

Kiefer J, Wolfowitz J, et al. Stochastic estimation of the maximum of a regression function. The Annals of Mathematical Statistics 1952;23(3):462–6.

Kim SH, Nelson BL. A fully sequential procedure for indifference-zone selection in simulation. ACM Transactions on Modeling and Computer Simulation (TOMACS) 2001;11(3):251–73.

Lessmann S, Stahlbock R, Crone SF. Optimizing hyperparameters of support vector machines by genetic algorithms. In: IC-AI. 2005. p. 74–82.

Lorenzo PR, Nalepa J, Kawulok M, Ramos LS, Pastor JR. Particle swarm optimization for hyper-parameter selection in deep neural networks. In: Proceedings of the Genetic and Evolutionary Computation Conference. New York, NY, USA: Association for Computing Machinery; GECCO '17; 2017. p. 481–488. URL: https://doi.org/10.1145/3071178.3071208. doi:10.1145/3071178.3071208.

Maclaurin D, Duvenaud D, Adams RP. Gradient-based hyperparameter optimization through reversible learning. 2015. arXiv:1502.03492.

Pedregosa F. Hyperparameter optimization with approximate gradient. arXiv preprint arXiv:160202355 2016;.

Peng-Wei Chen , Jung-Ying Wang , Hahn-Ming Lee . Model selection of svms using ga approach. In: 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541). volume 3; 2004. p. 2035–2040 vol.3. doi:10.1109/IJCNN.2004.1380929.

Ressel P. De finetti-type theorems: an analytical approach. In: The Annals of Probability. volume 13; 1985. p. 898–922.

Robbins H, Monro S. A stochastic approximation method. The annals of mathematical statistics 1951;:400–7.

Sandha SS, Aggarwal M, Fedorov I, Srivastava M. Mango: A python library for parallel hyperparameter tuning. In: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE; 2020. p. 3987–91.

Snoek J, Larochelle H, Adams RP. Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems 2012;25:2951–9.

Spall JC. Simultaneous perturbation stochastic approximation. Introduction to stochastic search and optimization: Estimation, simulation, and control 2003;:176–207.

Tukey JW. The problem of multiple comparisons. Multiple comparisons 1953;.

Yan D, Mukai H. Stochastic discrete optimization. SIAM Journal on control and optimization 1992;30(3):594–612.

Yang L, Shami A. On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing 2020;415:295–316.