Sample Efficient Hyperparameter Optimization

Team 1

Korea Advanced Institute of Science and Technology, Korea

Abstract. As we have more complex machine learning algorithms, hyperparameter optimization is crucial for applying them in any machine learning domain like search-based software engineering (SBSE) domains or stochastic-based classification domains. Although off-the-shelf frameworks for search algorithms provide default hyperparamters tuned by developers, recent study has shown the impact of parameter tuning by carrying out a large empirical analysis [1]. In hyperparameter optimization problem, sample cost is very expensive. For example, in case of search algorithm, since a fitness value of a parameter set, i.e. sample, can be obtained after running a algorithm with that parameter set. In this report, we propose a sample efficient hyperparameter optimization algorithm which combines a Genetic Algorithm (GA) and Bayesian Optimization (BO) method. Our algorithm represents a chromosome as a list of hyperparameters as to have modular nature. Since GAs generally do not consider uncertainty in the search space, we exploited BO to predict the probability of improvement before actual evaluation of each individual in the current population. With the guidance of BO, GA can produce a population which has more promising individuals. We conducted experiments on two different domain of machine learning; Neural network using Convolutional Neural Network (CNN), Evolutionary algorithm using Genetic Programming (GP). Experimental results show that the combination of GA and BO helps in finding good hyperparameters with a small number of evaluations.

Keywords: Parameter Optimization, Genetic Algorithm, Bayesian Optimization

1 Introduction

In any machine learning domain, most important goal of each solution is to produce best performance to solve their problem by optimization. After we choose one machine learning algoritm as a solution for any problem, optimization is possible by tuning the hyperparameter. Generally, a proposed research algorithm has default parameter set which is tuned by the effort of researcher, however these default parameter set does not guarantee to provide optimal performance to every different problems. For example, Search based Evolutionary Algorithm such as Genetic Algorithm(GA) has many tunable parameters, e.g, crossover rate, mutation rate, population size, elitism rate, selection method, and ,to effect the performance of algorithm. To find optimal performance, the set of parameters of specific problems needs to be tuned differently by each problem and

this hypothesis is already proven by previous empirical experiments parameter tuning[1].

One of the critical barriers in hyperparameter tuning is efficiency of parameter optimizer. In case of search algorithm, since a fitness value of a parameter set, i.e. sample, can be obtained after running a algorithm with that parameter set. So, we need impossible expense in terms of computation and time to apply optimization method. Recently, many improved approaches are proposed to capture the efficiency of optimization in machine learning domains, Genetic Algorithm(GA) is one of the emerged solution for it and Bayesian Optimization(BO) is popular solution in neural network domain.

Our contribution in this report is two fold. First, we propose novel method of hyperparameter optimization using a combination from both GA and BO. GA is good solution for large search space based on the process of natural selection that belongs to the larger class of evolutionary algorithms (EA) ,however, GA generally do not consider uncertainty in the search space. BO make smart decisions based on the probabilistic model, but it works slower when number of hyperparameters increase. So we exploited BO to predict the probability of improvement before actual evaluation of each individual in the current population. We show that with the guidance of BO, GA can produce a improved population which has more promising individuals. Second, we demonstrates that our method can fit to a variety of machine learning algorithm, e.g, Neural Network(NN), Genetic Programming(GP).

In order to confirm that our approach is applicable to various machine learning fields, we evaluate to show that our approach is work with two different type of machine learning algorithms. We evaluate our method with simple Convolutional Neural Network(CNN) to verify in neural network domain using use MNIST which is a well-known large database of handwritten digits. And, for search algorithm verification, we also evaluate our method with Genetic Programming(GP) to solve fault localization problem using FLUCCS datasets which is contained feature data for 156 real world faults.

Section 2 reviews prior work in hyperparameter optimization. Section 3 describes explain the our approach. Section 4 describes comparison results of 1) GA, 2) BO,3) GA and BO combination on both CNN and GP. Section 5 closes with a conclusion and future work.

2 FOUNDATION

2.1 Genetic Algorithm for Hyperparameter Tuning

We chose the genetic algorithm as one way of parameter tuning. The basic concept of our GA is similar with general one. The one thing different is we use fitness evaluation as a result of search algorithm that use our target parameter. Therefore, each generation contains the algorithm execution process. User can choose number of population and generation and give the fitness function for evaluation. Furthermore, user gives the hyper parameter configuration to program. Each parameter option in hyper parameter configuration consists of type

of parameter and range of parameter. For example, [[int,0,4],[float,0,3.0],[int,0,5]] can be one of instance of hyper parameter configuration.

The initialize process proceeds based on hyper parameter configuration. It takes each parameter option and makes individual solution by using random function. Each random parameter keeps the given type well. Create individual solution for a given population number.

The evaluation process is done through a given fitness function with individual parameter set. Fitness function can be any search algorithm that has hyper parameter. We store each fitness value of individual solution for using selection process. Because this process has to execute the search algorithm with each parameter set, it is the most costly process in the whole GA algorithm.

In selection process, we use linear rank selection to select the parents. Because the difference in fitness value of individual solution is small, we do not choose fitness proportional selection. If the difference in fitness value is small, then selection probability of individual solution will be almost same. Then, selection pressure of GA is too low to exploit the good solution. Also, we use stochastic sampling to give solutions that are less likely to be picked. When we use only GA for parameter tuning then the number of parents equal total population * (1 elitism rate) and when we use GA with BO then number of parents are same with population number.

We choose one point crossover to make children solution set. Length of individual solution is not quite long. Therefore, one point crossover is enough to birth various children. Two parents make two children after crossover, therefore number of children are same as number of parents.

After children are created, they undergo a mutation process. Mutation can occur in each parameter of individual parameter set. We set the mutation rate 5% for each parameter. If a mutation occurs, the parameter value will reset by random function. The reset parameter value keeps the type and range of initial parameter option.

After the mutation process, new individual solutions will be chosen by elitism. When we tune the parameter with only GA then we select elites of previous generation with elitism rate and merge elites of previous generation and children. They will be next generation. We determined elites of previous generation based on fitness value. On the other hand, if we use both GA and BO then number of children is number of population. Therefore, Replace some of the children with elites from previous generations. Children to be replaced are determined via BO.

We do not set any special stopping criterion. Whole GA process is done after given number of generation finished. After the GA process is complete, return the best fitness value and its parameter set.

2.2 Bayesian Optimization

Bayesian optimization(BO) is useful to model a black box function that is hard to described analytically. BO maintains a distribution of function, then update the distribution when new data is observed. Finally it finds a point that maximize the black box function, $\arg\max_{x \in A \subset R^d} f(x)$. To find the maximal point we have

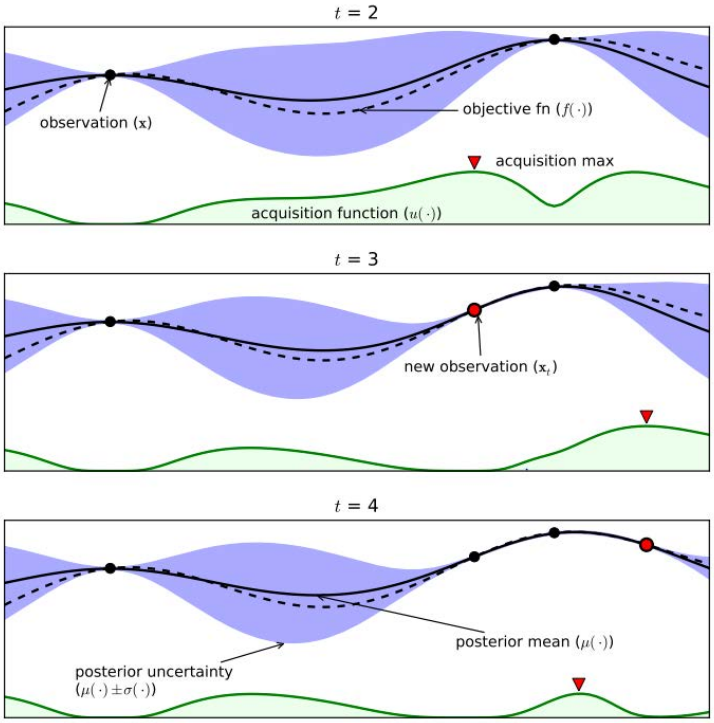

Fig. 1. Gaussian process approximation of objective function (from [4])

to search whole function domain, so it takes lots of time when the dimension of domain increases. When BO is applied to GA, the block box function to be estimated is fitness function.

To perform Bayesian optimization we used Gaussian processes with acquisition function. Gaussian processes generalize multivariate Gaussian distributions. Multivariate Gaussian distribution is defined by mean vector and covariance matrix, while Gaussian process is defined by mean function and covariance function. It regards a function as a vector of infinite dimensions.

Gaussian process predicts a value of the object function at each point x as a Gaussian distribution. When pre-observed Data D_t is given the distribution for y_{t+1} is estimated as,

$$P(y_{t+1}|D_t, x_{t+1}) = \mathcal{N}(\mu_t(x_{t+1}), \sigma_t^2(x_{t+1} + \sigma_{noise}^2)),$$
$$\mu_t(x_{t+1}) = k^T[K + \sigma_{noise}^2 I]^{-1} y$$
$$\sigma_t^2(x_{t+1}) = \mathcal{K}(x_{t+1}, x_{t+1}) - k^T[K + \sigma_{noise}^2 I]^{-1} k$$

σ_{noise}^2 is a given value, K and k are a kernel matrix and vector derived from a positive definite kernel, $\mathcal{K}$. y is the t-dimensional vector of pre-observed values. In our experiments, matern kernel is used for the kernel function.

As Gaussian processes observe new data point the variance of neighboring region of the observed point decreases (Figure 1). That means the objective function near the observed data becomes more accurate. So to efficiently update distribution of function we have to observe a point which has a high variance. While considering the variance reduction we have to find the the point that have maximum objective function value. To deal with these two metrics acquisition function is used representing the score for being selected as a next observation(Figure 1).

Two commonly used acquisition functions are expected improvement and probability of improvement. Expected improvement is defined as following. y^+ is maximum observed value

$$EI(x) = E[max(0, Y_{t+1} - y^+)|D_t, x_t + 1 = x]$$

Expected improvement is a reasonable metric But it sometimes gives zero and makes it impossible to compare two distinct observation points. So we used probability of improvement as acquisition function. It is defined as following.

$$PI(x) = P(y \geq y+) = \Phi((\mu(x) - y^+)/\sigma(x))$$

It measures the probability of our currently observed data over-performs previous best one. Since the probability never goes zero. It is possible to compare any two distinct observations. So we take the probability of improvement as acquisition function. Whenever BO algorithm selects a new sample. It finds a sample that has a highest acquisition function value.

3 Sample Efficient Hyperparameter Tuning

GA is good for exploring a very large search space. However, GA forms a generation without considering existence of similar solutions IN earlier generations. If there existed an individual in earlier generations whose fitness value was not good, it could be wasteful to evaluate the similar new-born individual. To tackle this problem, we exploit BO to predict the probability of improvement for individuals in the candidate population. Instead of direct sampling from the acquisition function of BO, we calculate the probability of fitness improvement for each individual in the generation formed by genetic operations. By removing unpromising entities in the candidate population, we can expect the population to be filled with more promising solutions.

Algorithm 1 describes the details of the proposed method. At first, the first generation is randomly initialized. After evaluating the current population P(t), the probabilistic model is updated with fitness values of individuals in P(t). After executing genetic operations to form P(t+1), the algorithm checks whether each individual in P(t+1) is promising or not using the model. After removing unpromising entities using a selection method, some elites are borrowed from P(t).

Algorithm 1 Genetic Algorithm guided by Bayesian Optimization

```
Initialize P(0) randomly
for t=0 to MaxGenerations do
    Evaluate population P(t)
    Update the probabilistic model
    Selection()
    Crossover()
    Mutate()
    Form P(t+1)
    Get probability of improvement for each individual in P(t+1)
    Remove N * r_i individuals based on probability of improvement
    Add N * r_i elites from P(t) to P(t+1) where r_i is the elitism ratio
end for
```

4 Experiments

We compared the performance of the proposed method with GA and BO on two search algorithms. Evaluation metric is the fitness of the best solution after evaluating 100 samples. We ran 10 times to beat randomness of search algorithms.

4.1 CNN Hyperparameter Tuning

Convolutional neural networks (CNN) are widely used in classifying images. We tuned CNN which is designed to classify a hand written digit dataset [2]. Tunable parameters were output size of the convolution layer, dropout rate in convolution layers, maxpooling size, number of dense layers, dropout rate in dense layers and learning rate. We used classification accuracy as a fitness value for the hyperparameter set. Figure 2 shows the quality of hyperparamters tuned by three algorithms. Overall, all algorithms provide quite good parameters mainly caused by neatness of the dataset. In other words, the choice of parameters does not have a large impact on the quality of the solution. However, as we can see in the figure, the proposed method provides slightly better parameters than others do.

4.2 GP Hyperparameter Tuning

The next experimental results came from GP hyperparmeter tuning. We implemented GP to solve software fault localization problem. The implemented basic features described in [3]. We tuned maximum tree depth, elitism size, crossover rate and mutation ratio. We defined the fitness function in terms of normalized ranking. The high fitness value means that faulty methods are highly ranked. As shown in Figure 3, the best solution comes from GA. We expect it is mainly from the randomness of the algorithm. However, the average fitness of the parameters tuned by the proposed method is higher than others.

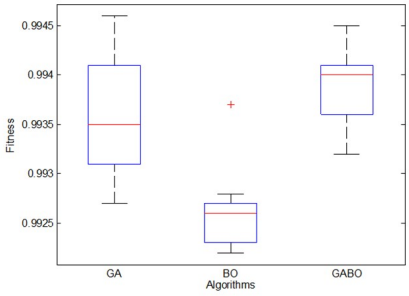

Fig. 2. Quality comparison of tuned CNN hyperparmeters

5 Conclusion

In this report, we proposed a hybrid hyperparameter tuning method which combines GA and BO. We also evaluate the performance of the proposed method for two search problems. Experimental results shows that the proposed method is sample efficient, however, we need further experiments to validate the results. Due to the sudden schedule changes, we were unable to complete experiments on EVOSUITE [5]. We are planning to improve our algorithm and evaluate the performance on other search algorithms and off-the-shelf SBSE tools including EVOSUITE.

References

1. Arcuri, Andrea, and Gordon Fraser. "Parameter tuning or default values? An empirical investigation in search-based software engineering." Empirical Software Engineering 18.3 (2013): 594-623.
2. THE MNIST DATABASE, http://yann.lecun.com/exdb/mnist/
3. Sohn, Jeongju, and Shin Yoo. "FLUCCS: using code and change metrics to improve fault localization." Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis. ACM, 2017.
4. Brochu, Eric and Cora, Vlad M and De Freitas, Nando. "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning" ParXiv preprint arXiv:1012.2599, 2010.
5. http://www.evosuite.org/

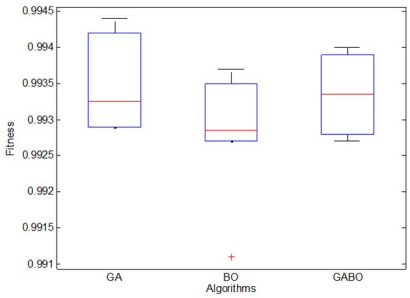

Fig. 3. Quality comparison of tuned GP hyperparmeters