



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원 저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리와 책임은 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)



공학석사 학위논문

**A Unified Approach on
Bayesian Optimization of
Deep Neural Network**

심층 신경망의 베이지안 최적화에
대한 통합적 접근법

2017년 7월

서울대학교 융합과학기술대학원

융합과학부 디지털정보융합전공

노승은

**A Unified Approach on
Bayesian Optimization of Deep Neural Network**

심층 신경망의 베이지안 최적화에
대한 통합적 접근법

지도교수 이 원 종

이 논문을 공학석사 학위논문으로 제출함

2017년 7월

서울대학교 융합과학기술대학원

융합과학부 디지털정보융합전공

노승은

노승은의 공학석사 학위논문을 인준함
2017년 7월

위 원 장	<u>이 교 구</u>	(인)
부 위 원 장	<u>서 봉 원</u>	(인)
위 원	<u>이 원 종</u>	(인)

Abstract

A Unified Approach on Bayesian Optimization of Deep Neural Network

Seungeun Rho
The Graduate School of
Convergence Science & Technology
Seoul National University

Due to the success of deep neural network (DNN) in recent years, DNN has begun to be applied in many research and application fields. At the same time, optimizing the hyper-parameters of DNNs has become an important issue because the performance of network is very sensitive to the setting of the hyper-parameters, such as the size of the convolutional filter or the number of neurons in the fully connected layer. Therefore, active research is being conducted on hyper-parameter optimization (HPO), and the Bayesian optimization method has shown promising results. However, there are some limitations with the Bayesian optimization. First, there still exists various options that need to be set in the optimization process itself; models and acquisition functions. Although the optimization varies greatly depending on the model and the acquisition function, we do not know which option is the best for a given dataset. Therefore, currently, it is all about choosing one that seems appropriate and hoping to be lucky. Second, the acquisition functions do not reflect the time-

cost, which leads to time-inefficient selections. If we can predict how much time a candidate will spend beforehand, we can make a more cost-efficient choice.

In order to solve these two problems, we propose the following method. First, we suggest a unifying approach which adapts to the problem and dataset within a single optimization process. It first explores which models and acquisition function performs well, and then converges to well performing options to exploit them. Second, we devise the acquisition function that takes time-cost into account. To do so, we must be able to predict the training time of the DNN. Hence we also propose a fast and accurate method to predict the training time of the DNN. Based on these two methods, we tried to improve Bayesian optimization, and we conducted several experiments to confirm the effect of unifying approach and time-efficient acquisition functions. Through the experiment, we achieve improvements in terms of time consumption and performance.

Keywords: Bayesian Optimization, Hyper-parameter Optimization, Deep Neural Network, Multi-armed Bandit, Acquisition Function.

Student Number: 2015-26100

TABLE OF CONTENTS

I. Introduction

1.1 The Success of Deep Neural Networks	1
1.2 Hyper-parameter Optimization.....	3
1.3 Overview of this Study.....	5

II. Bayesian Optimization

2.1 Introduction to Bayesian Optimization	7
2.2 Models	9
2.3 Acquisition Functions	12

III. Research Questions and Related Works

3.1 Research Questions	16
3.2 Related Works for Unifying Approach.....	18
3.3 Related Works for Cost-Efficient Acquisition Function.....	23

IV. Method

4.1 Pre-evaluated Look-up Table	27
4.2 Unifying Models and Acquisition Functions	30
4.3 Time Efficient Acquisition Functions.....	32

V. Experiment Results

5.1 Experimental Setup	35
5.2 Unifying Models.....	37

5.3 Unifying Models and Acquisition Functions	40
5.4 Time-Efficient Acquisition Functions	44
VI. Conclusion	
6.1 Summary	52
6.2 Contributions	53
6.3 Limitations.....	54
References	56

List of Tables

Table 1.	An example of Look-up table	28
Table 2.	Hyper-parameters list and its types and range	36
Table 3.	Time taken to reach 90%, 98%, 99% performance for models ..	39
Table 4.	Time taken to reach 90%, 98%, 99% performance for models and acquisition functions	42
Table 5.	Linear Regression Results Statistics	45
Table 6.	Time taken to reach 90%, 98%, 99% performance for time considered acquisition functions	49
Table 7.	Average number of trials in an hour	50

List of Algorithms

Algorithm 1.	Bayesian Optimization	9
Algorithm 2.	GP-Hedge	21

List of Figures

Figure 1.	The architecture of Google-Net	2
Figure 2.	Example of Gaussian Process with 3 observations.....	10
Figure 3.	Concepts of PI, EI, and UCB	12
Figure 4.	Examples of acquisition functions and their setting	14
Figure 5.	A row of slot machines in Las Vegas	19
Figure 6.	Comparison of the diverse acquisition functions including GP-Hedge.....	22
Figure 7.	Validation error on the CIFAR-10 data for different optimization strategies	26
Figure 8.	Comparison of random and Sobol sequence in 2 dimension.....	29
Figure 9.	Options for models and acquisition functions	31
Figure 10.	Batch processing times for n hyper-parameter vectors.....	33
Figure 11.	Bandits for different ways of considering EI with ET	34
Figure 12.	The architecture of modified LeNet-5	35
Figure 13.	Example images from MNIST dataset.....	36
Figure 14.	Mean and median performance over time for 3 different models and unified approach.....	38
Figure 15.	Mean and median performance over time for 7 different models and acquisition functions and unified approach	41
Figure 16.	Linear Regression Results on Training Time.....	46
Figure 17.	Mean and median performance over time for 4 different ways to consider time cost and unified approach.....	48

Chapter 1. Introduction

1.1 The Success of Deep Neural Networks

Recently, the deep neural network (DNN) has achieved remarkable results in a variety of fields. Especially, it showed good performance in the field of image classification, speech recognition, machine translation, which traditional machine learning algorithms suffered. This lead to the active research in relevant academia and development of application in practitioners. But, another important obstacle emerged regarding training deep neural networks: choosing proper hyper-parameters. For successful training of network, we have to set a lot of parameters such as number of neurons in fully connected layers, filter size of convolutional layers, pooling window size, striding size, etc. Furthermore, the performance of the neural networks are highly sensitive to the setting of hyper-parameters, so we need to choose appropriate hyper-parameters. The problem of choosing proper hyper-parameters already existed in the past, but it is much more difficult in DNN for the following two reasons.

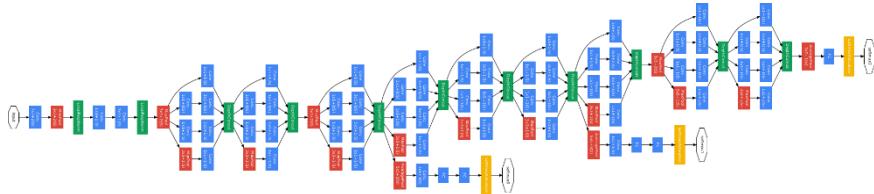


Figure 1. The architecture of GoogLe-Net (Figure from [15]).

The first reason is that as the network becomes more and more complex, the number of hyper-parameters becomes larger. Deep neural networks, as the name suggests, are heavily layered, so there exist a lot of hyper-parameters that need to be set. For example, Figure 1 shows an overview of GoogLe-Net which is composed of hundreds of layers. Each box has its own hyper-parameters. Other recently developed networks such as VGG net [19], residual-net [20], and inception-res-net[21] face similar situation.

The second reason is that measuring the performance of single hyper-parameter configuration is prohibitively expensive. To verify how a certain hyper-parameter configuration performs, the neural network has to be trained first. Training a DNN is heavily time consuming process which typically requires hours to days. After training is finished, the trained model is evaluated on the validation set and finally we can check the accuracy of the hyper-parameter vector. Furthermore, this process is just about evaluating the performance of a single hyper-parameter configuration. Numerous iterative evaluations of hyper-parameter

configurations are required to find best performing one. This leads the total time to be typically between weeks to months.

Due to the two reasons presented above, we cannot find a good hyper-parameters by traditional approach such as brute force search or grid search. Therefore, a more sophisticated method is required, and the related research is called hyper-parameter optimization.

1.2 Hyper-parameter Optimization

The process of hyper-parameter optimization (HPO) can be divided into two types, search approach and fine-tuning approach. Although it is difficult to strictly distinguish between the two, roughly speaking, search approach does not fix the dimension of the target hyper-parameter space. It dynamically manipulates the dimensions, and explores the search space. It aims to find well-performing architecture from unbounded, huge space. Currently, reinforcement learning based methods (B. Zoph et al [16] and B. Baker et al [22]) and evolution algorithms based methods (E. Real et al [27]) showed promising results for solving search problems. On the other hand, fine-tuning approach deals with a fixed, and bounded space. The range of each hyper-parameter is given, and the objective is to find the best performing set of hyper-parameters within the range. In summary, the hyper-parameter set and the range of each hyper-parameter are fixed for fine-tuning approach, and nor for the

search approach.

Here, the ranges of hyper-parameters are important key to utilizing prior knowledge. If there exists some prior knowledge about architecture, we can exploit a fine-tuning approach to focus on performance gain within a pre-defined range of hyper-parameters. On the contrary, if there is no prior knowledge, we need to find a plausible architecture in a bottom-up way. We can apply the search approach here, but our research topic will be limited to the fine-tuning approach among HPO. From now on, HPO refers to the fine-tuning approach of hyper-parameter optimization in this study.

The HPO problem can be formalized as follows according to [13] :

1. Given a machine learning algorithm A .
2. A has hyper-parameters $\lambda_1, \lambda_2, \dots, \lambda_n$ with respective domains $\Lambda_1, \Lambda_2, \dots, \Lambda_n$.
3. We define hyper-parameter space as $\Lambda = \Lambda_1 \times \dots \times \Lambda_n$
4. For each hyper-parameter setting $\lambda \in \Lambda$, we use A_λ to denote the learning algorithm A using this setting.
5. We further use $l(\lambda) = \mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ to denote the validation loss(e.g., misclassification rate) that A_λ achieves on data \mathcal{D}_{valid} when trained on \mathcal{D}_{train} .
6. The hyper-parameter optimization problem is then to find $\lambda \in \Lambda$ minimizing $l(\lambda)$.

Here, n is the number of hyper-parameters. HPO is a problem of finding a point with a maximum(or minimum) value of non-convex function \mathbf{l} in n -dimensional space. The problem becomes more complicated as n becomes larger, because of the curse of dimensionality. Furthermore, total network must be trained for single evaluation of \mathbf{l} , hence it is computationally expensive and consumes massive time.

Since conventional machine learning algorithms such as Random Forest and Support Vector Machines have relatively small number of hyper-parameters to set, a simple method like grid search was standard practice for HPO. However, J. Bergstra et al. suggests that if some of these hyper-parameters are not sensitive to performance, random search is more effective than grid search [17]. After that, manual tuning based approach and random search were standard practice but they cannot handle DNN well. Therefore, finally the algorithms based on Bayesian Optimization are proposed which is proper for optimizing DNN. The detail of this approach will be covered in chapter 2.

1.3 Overview of this Study

This paper can be roughly divided into two parts. The first part, Chapters 2-3, is about research motivation. Chapter 2 will cover various models and acquisition functions of Bayesian Optimization to help understand this study. Chapter 3 presents research motivations, research questions, and preceding research. The second part, Chapters 4-6, is about research

methodology and results. We will explain the methodology in chapter 4 and discuss actual experimental results in chapter 5. Finally, chapter 6 will present the contribution and conclusion of this study.

Chapter 2. Bayesian Optimization

In this chapter, we will explain the process of Bayesian optimization. Subsequently, we will discuss model and acquisition functions, which are important factors of Bayesian optimization.

2.1 Introduction to Bayesian Optimization

The search for global maxima of a black-box function f within a given range has been ongoing research for a long time. The black-box function f is not expressed in a closed form, so we cannot obtain the derivative of f , and we have no information about whether f is convex. Thus it is hard to apply general convex optimization method. What we can do is the evaluation of $f(x)$ for any input vector x in given range. Here, evaluating f once is very costly in terms of time and computation. Bayesian optimization is known to be a suitable methodology for this type of problems.

Bayesian optimization is a sequential algorithm. At first, it evaluates a few x 's which are randomly picked. Then it assumes smoothness prior to the space of x . Smoothness means that if the input x changes slightly, the output $f(x)$ also changes slightly. Under smoothness prior

assumption, we can expect that if a candidate vector $x_{candidate}$ is close to certain pre-evaluated x , than the value of $f(x_{candidate})$ will be similar with $f(x)$, and variance will be small. On the contrary, if $x_{candidate}$ is far from any of x 's, then we have very little knowledge about $f(x_{candidate})$, so the variance will be large. According to Bayesian rule, we can strictly induce the posterior distribution of f based on pre-evaluated x 's and the smoothness prior.

The posterior distribution of f has expected mean and variance. Through this posterior distribution, we can calculate the utility of evaluating $x_{candidate}$. This utility function is called as **acquisition function**. Although there exists several types of acquisition functions, they usually have larger value when both mean and variance are large, since mean corresponds to expectation and variance corresponds to uncertainty. High score for uncertainty facilitate exploration. In section 2.3, three canonical examples of acquisition functions will be introduced. Next, $x_{candidate}$ which maximizes the acquisition function is actually evaluated by f . Then a new data point $(x_{n+1}, f(x_{n+1}))$ is added to pre-evaluated dataset. Then, we can update posterior distribution with extra data points. This whole process is repeated. The algorithm can be expressed as follows.

Algorithm 1. Bayesian Optimization from [1]

```
1: for  $t = 1, 2, \dots$  do
2:   Find  $\mathbf{x}_t$  by optimizing the acquisition function over the GP:  $\mathbf{x}_t = \operatorname{argmax}_{\mathbf{x}} u(\mathbf{x} | \mathcal{D}_{1:t-1})$ .
3:   Sample the objective function:  $y_t = f(\mathbf{x}_t) + \epsilon_t$ .
4:   Augment the data  $\mathcal{D}_{1:t} = \{\mathcal{D}_{1:t-1}, (\mathbf{x}_t, y_t)\}$ .
5: end for
```

A prominent application of Bayesian optimization is the hyper-parameter optimization of DNN [2]. As suggest in section 1.2, it is important to determine the appropriate hyper-parameters of DNN because the performance of DNNs varies very sensitively to the hyper-parameters setting. However, evaluating the performance of a DNN for a hyper-parameter vector is very costly, so we should carefully determine the hyper-parameter vectors to evaluate. This is the case when Bayesian optimization is well-suited. In practice, J. Snoek et al. [3] succeeded in optimizing CNN architecture with Bayesian optimization, and the resulting hyper-parameter set's performance surpasses state-of-art performance architecture found by human in 2012.

2.2 Models

In Bayesian optimization, there are several ways of modeling $f(x)$. The most common method is to use Gaussian Process [4]. Gaussian Process assumes each observation and each candidate is the result of sampling

from GP, and objective function follows multivariate Gaussian distribution.

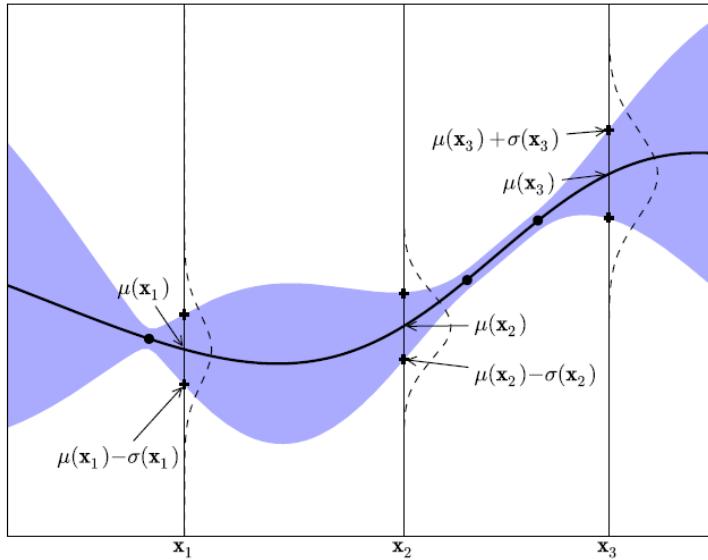


Figure 2. Example of Gaussian Process with 3 observations (figure from [1]).

As shown in the Figure 2, there is a correlation between the value of an evaluated point and the expected value of the candidate, and this correlation is stronger as the two points becomes close to each other. Therefore, the variance is small near the measured points, and the farther the point is, the larger the variance becomes. The specific mean and variance values can be calculated through the marginalization property of the multivariate Gaussian distribution. Gaussian Process is usually known to well-perform when the dimension of x is small and continuous.

J.Snoek et al. [3] present the algorithm in a form of software library called Spearmint.

Another common method is SMAC(Sequential model-based optimization for general algorithm configuration) [5]. SMAC uses random forests instead of Gaussian processes to predict the output of candidates. Since some data points which have been already evaluated are needed to fit the random forest, points are picked uniform randomly for the first few iterations. Based on these handful evaluated points, random forest regression is performed. Then we can predict the value of candidates hyper-parameter vectors. More specifically, we obtain a single value expectation for each tree in the random forest, and we can calculate the mean and variance of the value by integrating them. Since the fundamental of SMAC is the decision tree, it is known to handle categorical variables better than Spearmint.

The last method is TPE (Tree Parzen Estimator) [6]. If the previous two methods were modeled from well-defined concepts which are GP and Random Forest, this is rather heuristic approach. Similar to SMAC, the TPE also requires some evaluated points, and these points are picked randomly. Then, TPE uses certain thresholds to divide the evaluated points into two groups; good performing group and bad performing group. Then, by parzen estimator [18] within each group, calculate the probabilities that any candidate belongs to a high function value group and to a low function value group. The utility of this point is calculated through the ratio of two probability values. The highest point of the

utility is set to next trial point. According to J.Bergstra [6], this utility function is equivalent to calculating Expected Improvements.

2.3 Acquisition Functions

We've introduced methodologies to model the mean and variance of $f(x)$. In this section, we will discuss acquisition functions, a mapping function from candidate point to the utility of evaluation. Acquisition function uses mean and variance to the utility. Three typical acquisition functions are PI [7], EI [8], and UCB [9]. Figure 3 shows the concept of three acquisition functions.

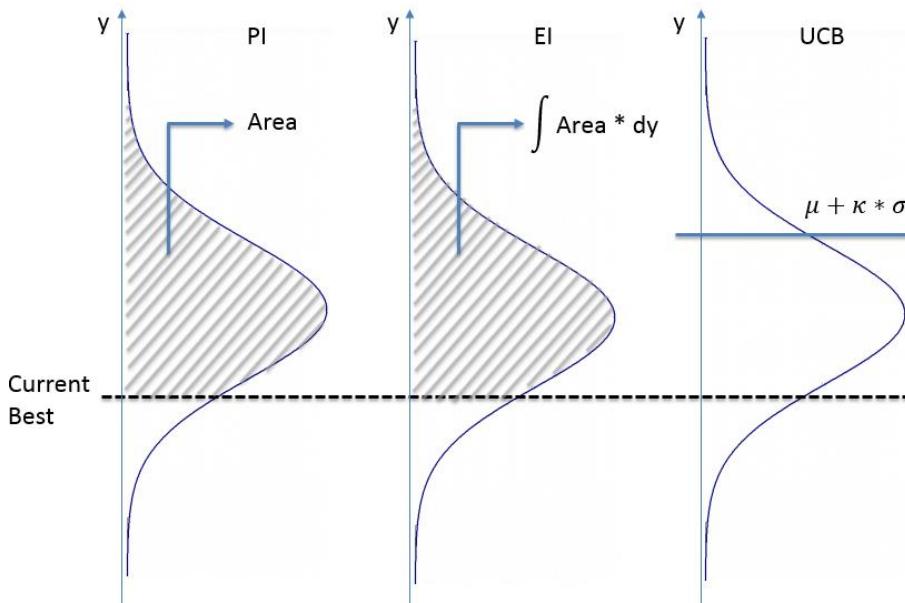


Figure 3. Concepts of PI, EI, and UCB.

- Probability of Improvement (PI)

$$a_{PI}(x ; \{x_n, y_n\}, \theta) = \Phi(\gamma(x)), \quad \gamma(x) = \frac{\mu(x ; \{x_n, y_n\}, \theta) - f(x_{best})}{\sigma(x ; \{x_n, y_n\}, \theta)}.$$

PI determines the probability that the output of the candidate will be higher than $f(x_{best})$, which is the largest value achieved by already evaluated hyper-parameter vectors.

- Expected Improvement (EI)

$$a_{EI}(x ; \{x_n, y_n\}, \theta) = \sigma(x ; \{x_n, y_n\}, \theta) \left(\gamma(x) \Phi(\gamma(x)) + \mathcal{N}(\gamma(x); 0, 1) \right)$$

The disadvantage of PI is that it can only utilize the probability of increasing the function value but not utilize how much it will increase. EI considers both probability and magnitude of improvements. EI is calculated by integrating probability and increment.

- Upper Confidence Bound (UCB)

$$a_{UCB}(x ; \{x_n, y_n\}, \theta) = \mu(x ; \{x_n, y_n\}, \theta) + \kappa \sigma(x ; \{x_n, y_n\}, \theta)$$

UCB is simpler. The mean and variance are considered together in the

simplest way of weighted sum.

The proposed acquisition functions select different candidates as the next trial. Therefore, overall Bayesian optimization follows a very different path depending on which acquisition function is used. Figure 4 shows three different acquisition functions with same posterior distributions. The results of maximizing acquisition function values differ a lot.

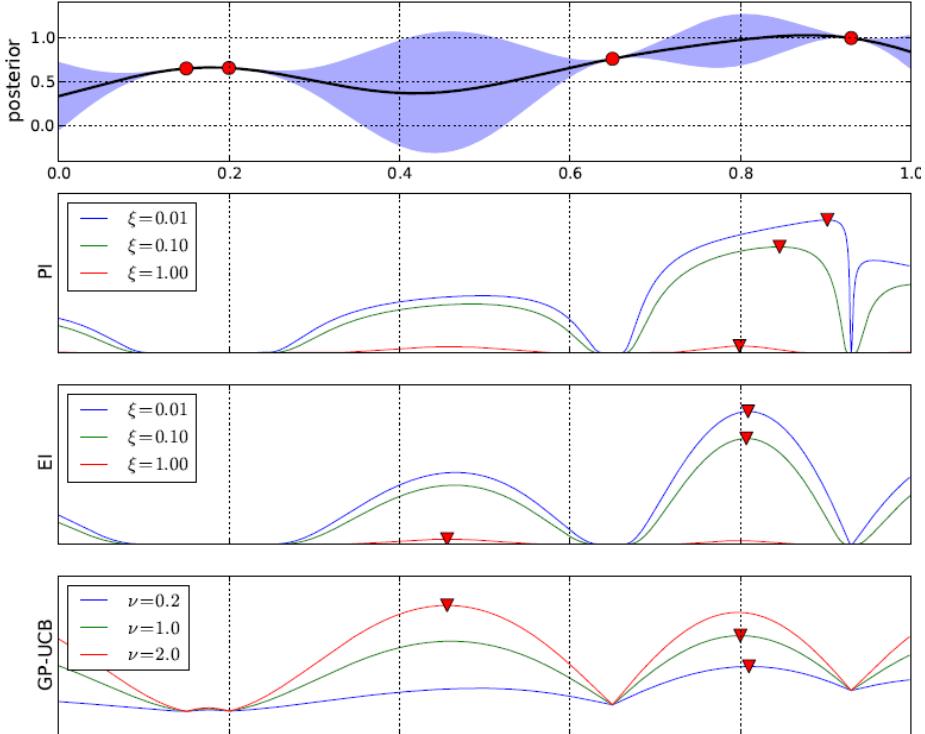


Figure 4. Examples of acquisition functions and their settings (Figure from [1]).

Since the behavior of each acquisition function is different, performance of each optimization process is different, and thus we have to choose the right acquisition for each optimization problem.

Chapter 3. Research Questions and Related Works

3.1 Research Questions

In section 2.2 and section 2.3, we've discussed various models and acquisition functions of Bayesian optimization. In practice, we have to choose a specific model and a specific acquisition function from the aforementioned ones. However, there is no optimal choice in general. The best model and the best acquisition function depends on dataset and problem, and we have limited information about which model and acquisition function might fit well to a given problem. Moreover, it requires additional choices from human. A natural question is how to unify existing models and acquisition functions into a single, well-performing one. We want this unified algorithm to adapt to the problem during optimization process, and this is the first research motivation.

Next, the second research motivation stems from the efficiency of acquisition functions. Currently, acquisition function does not consider the time cost. Figuratively speaking, it is similar with buying a product by seeking only utility without considering the price. If someone has inexhaustible money, this might be a rational strategy. Our time however, is limited and the best performance given time is one of the important metric for our goal. Furthermore, full process of Bayesian optimization consumes massive time. Since the single evaluation of the objective

function is very time-consuming as described above, total time consumption for optimization requires a great deal of time. Therefore, reducing the total time required is a very important issue. So, we plan to modify the acquisition function to take time cost into account. The basic idea is "Let's try a candidate that takes less time if it shows a similar utility." If this idea works well, HPO could achieve equal performance in a shorter time, or better performance in an equal time. To do this, we first need to predict the time of function evaluation before evaluating the function. Fortunately, in the case of a DNN, the time required for training can be predicted in advance. The specific method will be presented in consecutive sections.

To sum up, the purpose of this study can be summarized into 2 parts. The first step is to unify the existing models and acquisition functions to create a single adaptive strategy. The second step is to design more cost-efficient acquisition function. Hence we propose the following two research questions.

- RQ 1. How can we unify existing models and acquisition functions into a single adaptive strategy?
- RQ2. How can we make acquisition function to consider time cost of each evaluation?

If we can successfully answer these two research questions, we will ultimately achieve a better performing Bayesian optimization method.

Here, better performing means for given time, better performance is achieved, or for given performance, shorter time is required.

Since there are two research questions, section 3.2 describes the related work for the unified acquisition function, which deals with the first research question. Section 3.3 describes the related work on the cost-effective acquisition function.

3.2 Related Works for Unifying Approach

M. Hoffman et al. [11] suggested an idea about unifying the three acquisition functions EI, PI, and UCB, by applying a method used in multi-armed bandit problems. In section 3.2.1, we will explain the problem setting of multi-armed bandit and known algorithms. Next, we will introduce the research of M. Hoffman et al.

3.2.1. Multi-armed bandit

Multi-armed bandit(MAB) problem [10] settings are as follow. There are several slot machines in the casino, and players can play one slot machine at a time. The total number of games is set to n . Also, the reward of the slot machine is not known until actually running a bandit machine, because it is a random variable that is drawn from unknown probabilistic distribution. Then how the player could maximize the total sum of

reward from n games? What strategy should the player follow?



Figure 5. A row of slot machines in Las Vegas

We can play n games on one slot machine. But then we do not know if the reward of the other slot machines is better or worse. Thus we have to explore the other machines as well. On the other hand, we can play n games on different slot machines, but it will cost too much to get information of reward distributions of slot machines. Thus we have to exploit some slot machines which returns high rewards. In summary, we have to spend a few plays to find slot machines with moderately good reward, and we have to maximize our reward by playing continuously on some profitable slot machines. This problem reveals a trade-off between exploration and exploitation. Choosing the best slot machine based on experiences is called exploitation, and choosing other slot machines that have not been tried yet or understood well is called exploration.

Representative examples of multi-armed bandit problems include clinical trials and A/B testing. In the clinical trial, patients are asked to confirm the effectiveness of the treatment by establishing a treatment method. At this time, if only the good treatment is used, new treatments cannot be explored. On the other hand, if we explore various treatments, some patients need to suffer from rather ineffective treatments. Since the number of patients is fixed, we must find a balance between exploration and exploitation.

Analogous story holds for the A/B test. During the A/B test, web service shows different user interface (UI) for each user, and measure the effect of each UI. Here exists trade-offs between trying a various UIs and adhering to fairly well designed UI found through experiences.

Various approaches to solve MAB problems are known. Epsilon greedy [28], decaying epsilon greedy, Thompson sampling [29], Poker algorithm [30], and optimistic initialization are some of them. The details of epsilon greedy will be introduced in section 4.2. Suggested algorithms usually spend early stages on exploration, focusing on exploitation in later stages.

3.2.2. Choosing acquisition function as MAB

M. Hoffman et al. [11] suggest a method to unify acquisition functions into single one. The method is called hedge, which is a portfolio based strategies used in multi-armed bandits. The idea behind it is simple; any

single acquisition function cannot generally perform best in all cases. Therefore, like the bandits in MAB, different acquisition functions are chosen for every trial of Bayesian optimization according to hedge strategy. Then the chosen acquisition function results in reward, and the probability is adjusted with the reward it got. If it obtained a large reward, probability of being chosen is increased, and vice versa. This process is repeated until performance converges. In [11], they used 3 slot machines for EI, PI and UCB, which are canonical acquisition functions. The algorithm details are as follows.

The reward is the mean of Gaussian regression, and the reward is recorded for every trial. Recorded rewards form a reward vector. Each acquisition function has its own reward vector, and the probabilities of being chosen are proportional to the sum of the reward vector. That is, a function with a larger reward is selected with a higher probability. When the acquisition function selection is finished, then the algorithm actually evaluate a point which maximize the selected acquisition function. Then, update the reward vector again. Exact algorithm is presented in Algorithm 2.

Algorithm 2. GP-Hedge from [11]

-
- 1: Select parameter $\eta \in \mathbb{R}^+$.
 - 2: Set $g_0^i = 0$ for $i = 1, \dots, N$.
 - 3: **for** $t = 1, 2, \dots$ **do**
 - 4: Nominate points from each acquisition function: $\mathbf{x}_t^i = \text{argmax}_{\mathbf{x}} u_i(\mathbf{x} | \mathcal{D}_{1:t-1})$.
 - 5: Select nominee $\mathbf{x}_t = \mathbf{x}_t^j$ with probability $p_t(j) = \exp(\eta g_{t-1}^j) / \sum_{\ell=1}^k \exp(\eta g_{t-1}^\ell)$.
 - 6: Sample the objective function $y_t = f(\mathbf{x}_t) + \epsilon_t$.
 - 7: Augment the data $\mathcal{D}_{1:t} = \{\mathcal{D}_{1:t-1}, (\mathbf{x}_t, y_t)\}$.
 - 8: Receive rewards $r_t^i = \mu_t(\mathbf{x}_t^i)$ from the updated GP.
 - 9: Update gains $g_t^i = g_{t-1}^i + r_t^i$.
 - 10: **end for**
-

Figure 6 shows the results of Bayesian optimization using the existing EI, PI, UCB acquisition functions and GP-hedge acquisition function, respectively. Experiments were conducted on 10-dimensional, 20-dimensional, and 40-dimensional input vectors. In many cases, the GP-Hedge algorithm outperforms any of choice.

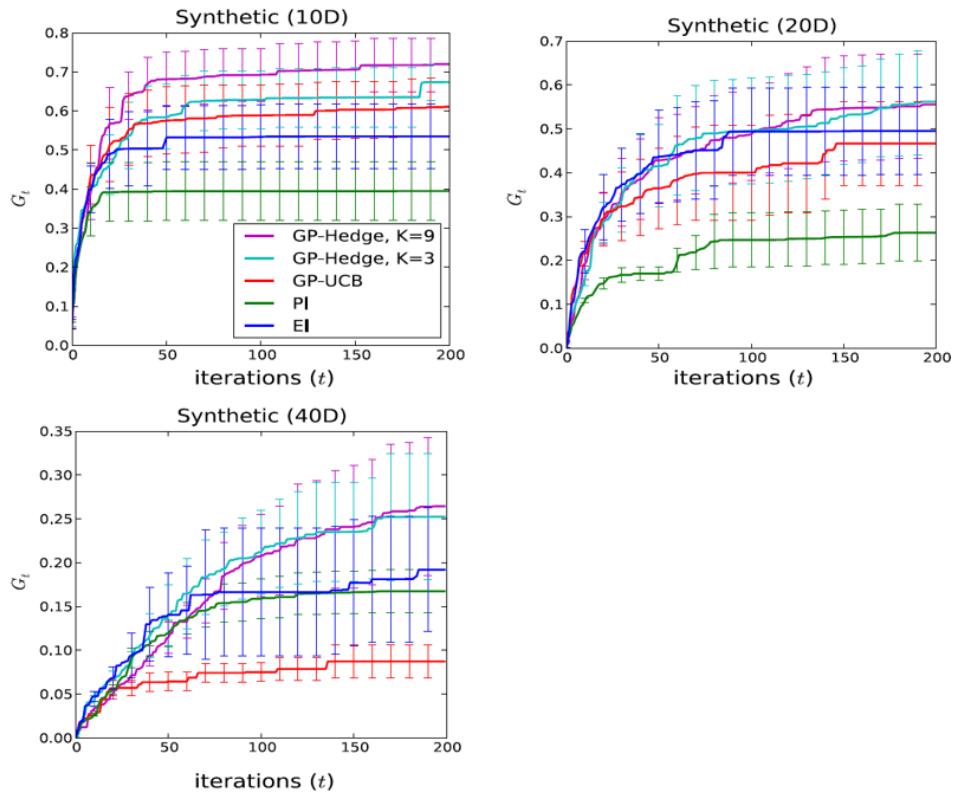


Figure 6. Comparison of the diverse acquisition functions including GP-Hedge. [11]

This study gave us a hint on unifying acquisition functions with MAB

approach. This study also shows better performance than each of bandit. However, despite the greater diversity of the models' behavior, this study does not address the selection of the model issue. We might also consider the model and the acquisition function jointly.

3.3 Related Works for Cost-Efficient Acquisition Function

Due to massive time consumption of Bayesian optimization, various researches have been conducted to make optimization process faster. T. Domhan et al [23] suggests a method of early stopping. In the early part of training, it decides whether to keep training or not by extrapolating the learning curve. Another research by J. Snoek et al [24] introduce neural-net based modeling method which is linearly proportional to the number of observations, while Gaussian Process is cubically proportional. In addition, A. Klein et al [25] reduced time consumption by utilizing only small portion of given dataset during optimization process. Also, M. Feurer et al [26] presents a method to speed up the optimization process by exploiting configurations that worked well in similar datasets. Although there have been many efforts related to time as presented here, a study by J. Snoek et al [3] was the only study that mentioned that the acquisition function should take time into consideration.

J. Snoek et al [3] introduced Bayesian optimization approach for HPO of

machine learning algorithms. In the research, they devised Spearmint, a Bayesian optimization software based on Gaussian process. Although the method itself is not a novel approach, this study suggests an idea about solving practical issues such as the selection of kernel function in Gaussian process, and the method of utilizing parallel processing. And they succeeds in finding a DNN architecture that actually surpasses the performance of DNN architecture which is tuned by human expert. In addition, they suggest that the acquisition function should consider not only the utility but also the cost of evaluating the function. They designed an acquisition function that takes time-cost into consideration. There are many ways to incorporate the calculated time-cost into the acquisition function. In this paper, they used division.

$$\frac{EI(x)}{CT(x)}$$

Here, $CT(x)$ stands for cost of time. It means expected time consumption for evaluating $f(x)$. They used Gaussian process to model the time-cost, which is exactly same way of modeling the output of the function. If the time-cost of x becomes large, the value of the acquisition function becomes small, and the probability that the corresponding candidate vector is selected becomes relatively small.

However, the approach on modifying acquisition has several limitations. Figure 7 shows the performance of different optimization strategies. The line with “GP EI per second” is the result of time-cost considered acquisition functions. As can be seen from the results, the performance

difference between the case with and without time-cost is insignificant. This may be due to imprecision in predicting the time cost, or the naïve methodology for modifying the acquisition function. Since the result of independent experiment is not presented in the paper, it is not clear which element is responsible for the result.

In summary, this paper presents the idea that time cost should be taken into consideration, but it has limitations in that it does not give specific methodology on how to accurately predict the time cost. Also, there exists numerous ways to consider EI with CT, but this paper does not provide the result of other approaches. Through this, we were able to think about how Bayesian optimization could further develop the method of considering the time cost.

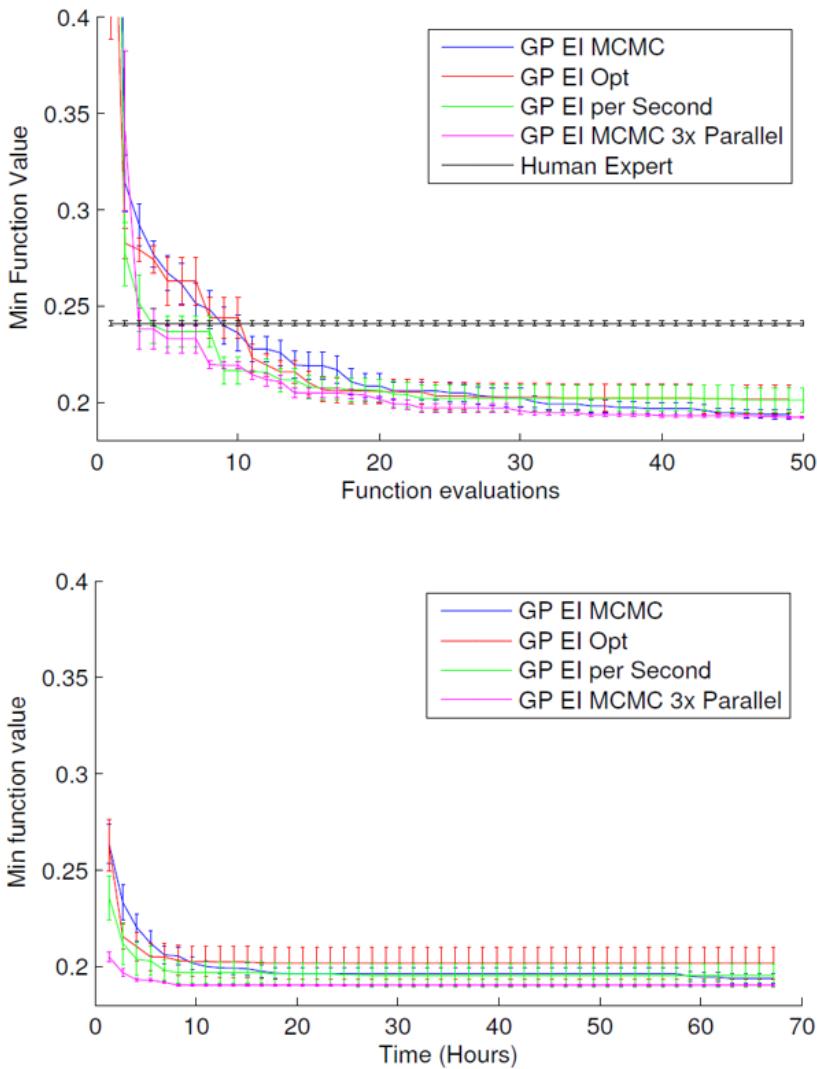


Figure 7. Validation error on the CIFAR-10 data for different optimization strategies. [3]

Chapter 4. Method

In this chapter, we will discuss the methodology that has been applied to the two research objectives presented in chapter 3. First, in section 4.1, we will explain how efficient experiment can be made with pre-evaluated look-up table. Next, we will describe what algorithms were used to unify the models and the acquisition functions in section 4.2. Finally, in section 4.3, we will introduce how to estimate the time cost with high precision, and how to apply it to acquisition functions.

4.1 Pre-evaluated Look-up Table

The total process of a Bayesian optimization is composed of iterative training of deep neural networks, so it is a very time consuming(days to weeks) task. Furthermore, given algorithms (Spearmint, SMAC, TPE) occasionally perform disastrously, and this leads to huge variance of performance. Therefore, repeated experiments are required for credible results. Hence we decided to pre-evaluate roughly uniformly distributed 20,000 points from hyper-parameter space. All the optimization experiments from now on will be processed within 20,000 points. This does not harm generality, because in fact, the internal implementation of the Spearmint library actually optimizes 20,000 candidates in exactly the

same way. The pre-evaluated 20,000 points form a look-up table. Example Look-up table is suggested in Table 1.

Table 1. An example of Look-up table

Num	λ_1	λ_2	λ_n	Duration(s)	Accuracy
1	32	2	0.70	557.9	0.973
2	46	3	0.41	448.4	0.112
3	15	2	...	0.73	0.965
4	16	6	0.26	321.0	0.891
⋮					
k-1	55	5	0.66	493.4	0.920
k	48	5	0.60	263.2	0.102

The look-up table records the time spent in evaluating the corresponding function for all 20,000 hyper-parameter vectors and the validation accuracy achieved by the model. Then, in the Bayesian optimization experiment, the output of the corresponding function can be searched in the look-up table without having to evaluate each function. Therefore, the evaluation cost is infinitesimal, and it is possible to perform repeated experiments within a short time. The optimization will then be made within 20,000 candidate points. To approximate uniform distribution of candidate points, points are extracted from sobol-sequence [12]. This is also a equivalent method from Spearmint library. Here, Sobol-sequence refers to a low discrepancy sequence in unit hypercube. Figure 8 shows how Sobol sequence is more uniformly distributed than randomly

generated sequence in 2 dimension space.

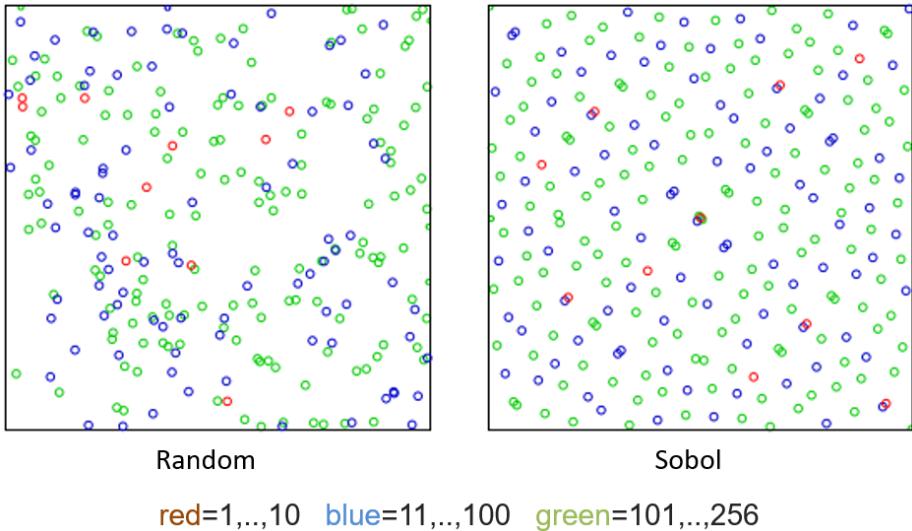


Figure 8. Comparison of random and Sobol sequence in 2 dimension.

Note that evaluating these 20,000 points means training 20,000 different neural networks, since each point correspond to each hyper-parameter vector of a neural network. It takes a long time in itself, but then it is possible to experiment quickly and efficiently. Twenty-five days were spent using four Titan-x graphic processor unit to evaluate 20,000 points.

4.2 Unifying Models and Acquisition Functions

To be concrete, our research goal is to eliminate the concerns of users on which models and acquisition functions to use. This requires a

methodology that adapts to the problem, since different models work differently on different problems and the best option is dependent on the specific problem and dataset. In an optimization episode, such as a multi-armed bandit, the early stage is used to explore several models and acquisition functions, and the second half is converged to a well-functioning model to exploit. Therefore, we can utilize methodology known to work well in multi-armed bandits. We decided to use the decaying ϵ -greedy algorithm because of its simplicity.

The decaying ϵ -greedy algorithm works on a very simple principle, but is known as a very powerful algorithm. Even complex algorithms often fail to surpass the performance of this simple algorithm. The principle is as follows. Considering the average of the rewards that each bandit has acquired so far, select the bandit that has the highest value. But explore with probability of ϵ . That is, one of the remaining choices is randomly selected except for the best option with probability of ϵ . Here the magnitude of ϵ decides the weight of exploration. Larger ϵ leads to more exploration. This is the ϵ -greedy algorithm. Decaying ϵ -greedy algorithm continues to decrease ϵ as the step progresses. It reflects the intention to gradually reduce the opportunities for exploration when the best options become clear to some extent.

With decaying ϵ -greedy algorithm, we can unify models and acquisition functions. Reward determined as the validation accuracy of trained model with each option. We propose a stepwise method to finally select a model and an acquisition function at the same time. In experiment 1.1, we will try to unify the models. In experiment 1.2, we will then jointly

unify the models and the acquisition functions together. Within single episode of optimization process, we hope ϵ -greedy algorithm finds best models, and finally converges to the model (and acquisition function). This is illustrated in Figure 9. One of each box is correspond to bandit, and a bandit is selected for each trial.

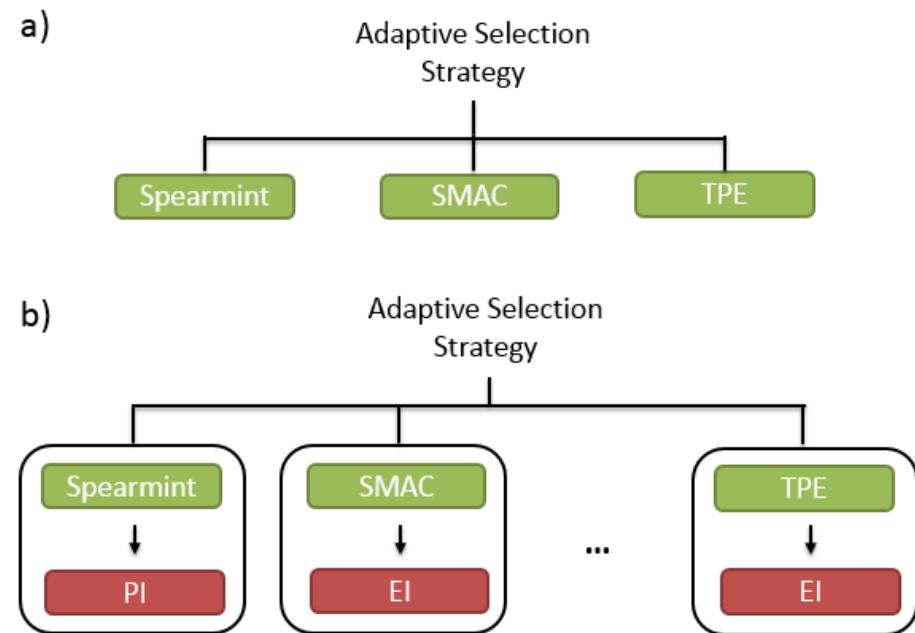


Figure 9. Options for models and acquisition functions. Adaptive selection strategy is decaying ϵ -greedy algorithm. a) corresponds to model unification, and b) corresponds to joint unification of model and acquisition functions.

4.3 Time Efficient Acquisition Functions

In this section, we will introduce how training time of DNN can be estimated, and how acquisition function can consider training time.

4.3.1. Training Time Prediction

Training of deep neural networks is time consuming and the acquisition function must take into account the training time of the candidates. However, since there is no information about the training time of the candidate, we need to create a model to predict it. If we can predict the training time of networks, more time-efficient acquisition function can be designed.

The question is how can we predict the training time of a network before actually training it. We focus on the fact that total training time is proportional to a batch processing time because network training process consists of millions of batch processing. Batch processing is an extremely iterative process. Here, the processing time of single batch is very short (several seconds), which makes it easy to measure. Therefore, if we measure the processing time of several batches, and make a prediction model between hyper-parameter vectors and batch processing time, we can assess the training time of a model. The reason why hyper-parameters have close relationship with training time is that, some of hyper-parameter decides the computational complexity of model. For example, number of neurons in fully connected layers and size of

convolutional filter have intimate relationship with computational complexity. Computational complexity is highly correlated with training time. Hence we can make a regression model between hyper-parameter vectors and training time of several batches in supervised manner. We decided to measure 10 batch processing times for n hyper-parameter vectors and regress the total training time through the supervised framework. Figure 10 shows the concept of supervised framework.

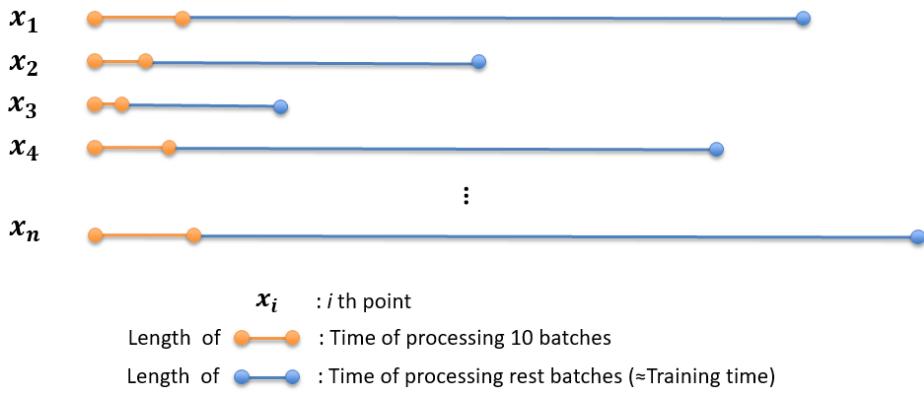


Figure 10. Batch processing times for n hyper-parameter vectors. They are proportional to total training time.

Note that this training time prediction experiment is independent from HPO experiment. Before full-scale optimization begins, this experiment is performed. And it give us a quite accurate model for predicting training time of any hyper-parameter vectors of neural network.

4.3.2. Cost-efficient Acquisition Function

Next natural question is how to apply time estimates into acquisition function. In case of using EI as an acquisition function and time estimates as ET, there are many ways to consider EI and ET together, such as

$$\frac{EI}{ET}, \frac{EI}{\log(ET)}, EI - 0.3ET, \dots$$

Analogous to unifying model and acquisition function, We decided to use the MAB approach to decide which of these options to choose. The outline is shown in Figure 11 below.

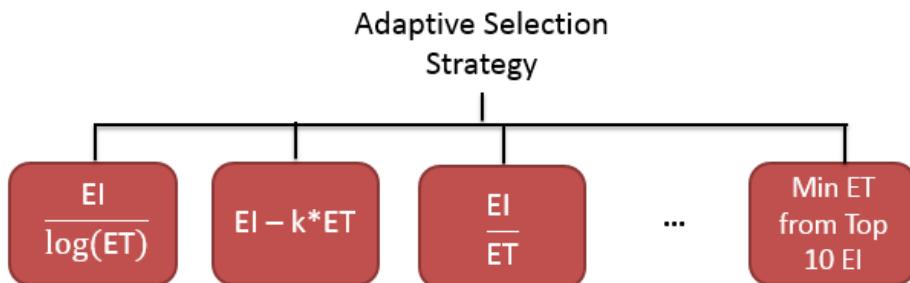


Figure 11. Bandits for different ways of considering EI with ET.

Likewise, the decaying epsilon greedy algorithm was used. Among the various methods, it converges toward the well-performing options.

Chapter 5. Experimental Results

In this chapter, we will first introduce the overall settings, such as architecture and dataset used in experiments. We then explain the unification of model and acquisition function experiments, and finally introduce the experiments about time-considered acquisition function.

5.1 Experimental Setup

5.1.1. Objective Architecture and Hyper-parameters

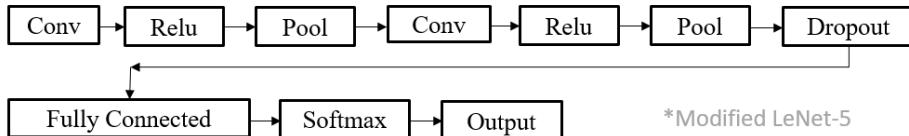


Figure 12. The architecture of modified LeNet-5.

We first need to determine the high-level architecture of the neural network that we want to optimize. As shown in Figure 12, we used a slightly modified architecture of the LeNet-5 [14] model, which is simple and prominent model from image classification. The modified parts are : uses REctified Linear Unit (Relu) as an activation function, and a dropout layer is added to prevent overfitting. Regarding hyper-parameter lists and its types and ranges used for optimization are shown in the Table 2 below.

Table 2. Hyper-parameters list and its types and range

Hyper-parameter Name	Type	Range
# of filters in conv1	Integer	1~350
Pooling 1 window size	Integer	2~3
# of filters in conv2	Integer	1~350
Pooling 2 window size	Integer	2~3
Conv Filter size	Integer	2~10
# of neurons in fully connected layer	Integer	1~1024
Learning rate	Float	0.0001~1.0
Regularization Param	Float	0.0~1.0
Dropout rate	Float	0.1~1.0

5.1.2. Dataset



Figure 13. Example images from MNIST dataset.

We have to set dataset for training and validation. We used MNIST dataset, which is the most widely used toy data in machine learning. As shown in Figure 13, the MNIST dataset consists of hand written digit from 0 ~ 9. Training data composed of 55,000 images, and Validation data composed of 5,000 images.

5.2 Unifying Models

We proceeded experiment to unify the models with acquisition function is fixed to EI. As introduced in section 2.2, there are three possible choices for the model: Spearmint based on Gaussian Process (GP), SMAC based on Random Forest (RF), and TPE. 100 times repeated for each experiment. The unifying algorithm used decaying epsilon greedy to select one of three trials for each trial. Unifying experiments repeated 40 times. The initial value of epsilon was set to 0.6 and designed to be reduced in inverse proportion to the log (step). Figure 14 represent the result of experiments at the aspect of performance in given time. Curve cornered on left upper side indicates good performance, since it means higher performance in given time. We presented both median and mean graph since mean is sensitive to outliers.

As can be seen in Figure 14, the most promising model is based on Gaussian Process. On the other hand, Random Forest and TPE do not work well. Note that the unified algorithm curve is similar to Gaussian process curve rather than that of RF and TPE. This shows that the unified

approach has successfully converged to quite plausible option during the optimization process.

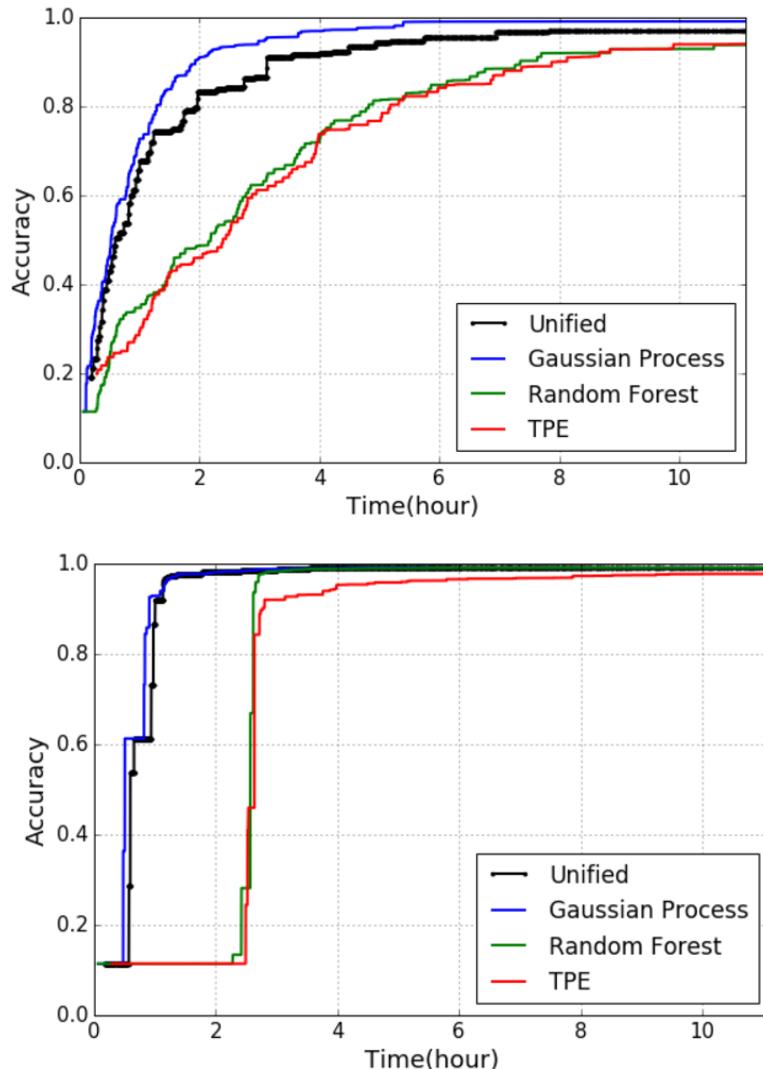


Figure 14. Mean (up) and Median (down) performance over time for 3 different models and Unified approach.

On the other hand, we presented the result at the aspect of the time taken to reach high performance in Table 3. In Table 3, "+" stands for the lower bound of performance since part of the repetitive experiment is forcibly terminated because the performance could not be reached within given time. The value is calculated with the cases that target performance is actually reached.

Table 3. Time taken to reach 90%, 98%, 99% performance for models.

	Mean			Median			Converge nce Ratio
	90%	98%	99%	90%	98%	99%	
Unified	1.5	2.5	5.7	1.0	1.8	5.8	
GP	1.2	2.1	4.9	0.9	1.9	4.3	36%
RF	3.8	4.3	7.3	2.6	2.8	5.6	64%
TPE	4.1	19.3	149+	2.8	12.4	151+	0%

In consistent with the results in Figure 12, the GP reaches 99% accuracy the fastest overall. However, here, RF reaches 98% and 99% within small amount of time, hence GP and RF are good, but TPE is disastrously bad. It is worth noting here that the unified algorithm reach median performance of 98% within shorter time than the GP. Although the gap is small, this shows adaptation actually happens and contributes to performance even though unified approach has a handicap: it exhausts early stage trials for distinguish poor performance algorithms, such as

TPE. In order to confirm this, we investigated which option decaying epsilon algorithm finally converged to. It converged to GP 14 out of 40 times, 26 out of 40 times to RF, and 0 times to TPE. The fact that there is no convergence to the TPE means that the unified algorithm adapt well to avoid the worst case, but it failed to converges on GP, the best option. It might be due to the design of reward system. We will discuss more about this issue on section 6.3.

5.3 Unifying Models and Acquisition Functions

Next, we created seven options(=bandits) jointly considering the acquisition function and model. Note that TPE fits with only one acquisition function because it cannot select any other acquisition function except for EI. GP and RF each have three options of acquisition functions. The mean and median of the performance over given time are presented in Figure 15.

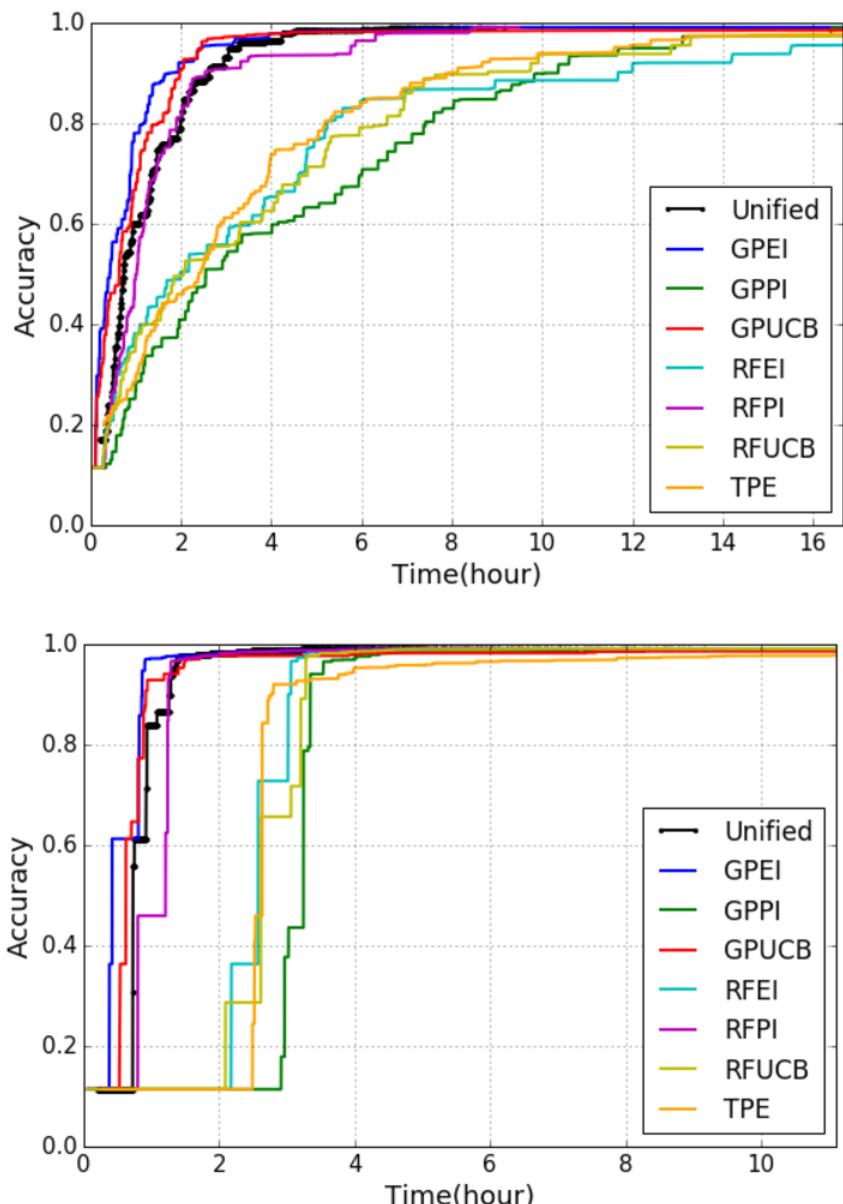


Figure 15. Mean (up) and Median (down) performance over time for 7 different models and acquisition functions and Unified approach.

Here, the seven algorithms could be divided into two groups. The first group quickly achieves high accuracy with GPEI, GPUCB, and RFPI, while the second group, RFEI, RFUCB, GPPI, and TPE, consume a lot of time to reach certain level of accuracy, especially in relatively low accuracy area. The unified algorithm successfully matches with the first well-performing groups. To be concrete, we presented time consumption to reach three accuracy levels – 90%, 98%, and 99%. Table 4 summarizes the results. The results are summary statistic from 50 repetitive experiments for all options.

Table 4. Time taken to reach 90%, 98%, 99% performance for models and acquisition functions. Green numbers indicates shortest time consumption.

	Mean			Median			Convergence Ratio
	90%	98%	99%	90%	98%	99%	
Unified	1.4	2.0	4.1	1.3	1.9	3.9	
GPEI	1.2	2.2	5.2	0.8	1.9	4.3	10%
GPPI	4.7	5.3	8.5	3.3	4.3	7.8	4%
GPUCB	1.2	5.9	21+	0.9	4.0	21+	4%
RFEI	4.3	4.8	7.3	3.0	3.3	5.5	32%
RFPI	1.8	2.3	4.7	1.2	2.0	4.5	40%
RFUCB	4.4	4.9	7.4	3.2	3.5	6.4	10%
TPE	4.1	19.3	149+	2.8	12.4	151+	0%

Based on the results in Table 4, we confirmed a very promising result. Unified algorithm surpassed all options for time consumption to reach 98% and 99%, although it is just a combination of the existing seven options. The results are consistent for both mean and median. Among the seven options, even inefficient algorithms for finding high performance such as GPUCB and TPE are included. Note that GPUCB belonged to the good performance group in Figure 13 above. This means that GPUCB quickly reaches fairly good accuracy in the early stages, but the ability to exploit to reach the best performance declines in later stages. The unified approach, according to the graph, behaves similar to GPUCB in the beginning, but converges to a well-suited algorithm for best performance, which means it only takes the advantage of existing options. The reason why unified algorithms achieve robust result might be found in the gap between mean and median. Unlike existing 7 options, the gap between mean value and median value is relatively small for unified algorithm. This implies that unified algorithm seldom produce outlier, which means that the case of disastrous failure rarely occurs in unified algorithm. Perhaps it is due to quick adjustment of the selection probability to avoid negative options, before it falls into disastrous failure. Nevertheless, it requires additional experiment to strictly confirm. Finally, we checked which option decaying epsilon greedy algorithm actually converges. According to mean values, the best performing (achieves 99% in shortest time) three algorithms are GPEI, RFPI, and

RFEI. The convergence ratio to one of these three options is 82% of the total, which is stable result. We can confirm exactly same results based on median. On the other hand, the worst performing (achieves 99% in longest time) three algorithms are TPE, GPUCB, GPPI, and the convergence ratio to one of these three options is 8%. Therefore, we also confirmed here that unified algorithm efficiently avoids bad options, and sticks to good options.

5.4 Time-Efficient Acquisition Functions

In this section, we will introduce how the acquisition function was modified to consider time cost. First, Section 5.4.1 introduces the results of the experiment about training time prediction of DNN, and the adaptation result of the acquisition function that reflects the time cost is explained in Section 5.4.2.

5.4.1. Training Time Prediction

We first performed the training time regression, the idea presented in section 4.3.1. The processing time of 10 batches was recorded for uniform randomly picked 500 points to minimize the time spent on training time regression itself. We used 300 points for training and 200 points for testing. At first, we only used the parameters related to the

model architecture among the hyper-parameters, then only the parameters related to the optimization were used. Finally, we tried to confirm the difference by using the parameters that combine the two. The mean squared error and R squared values for each case are shown in Table 5.

Table 5. Linear Regression Results Statistics

	Specification	Mean Squared Error	R^2
1	Architecture param only	0.019	0.766
2	Learning param only	0.084	-0.025
3	Total	0.020	0.758

From the results, it can be confirmed that the best result is obtained when only the architecture parameter is used. R squared value was 0.766. Figure 16 shows a visualization of the results.

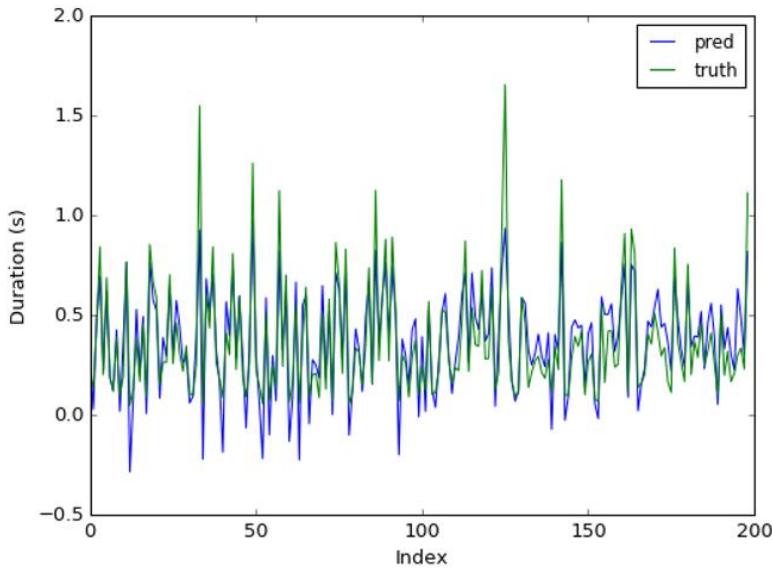


Figure 16. Linear Regression Results on Training Time

Only 300 points total can be used for training. It takes about 140 seconds to collect 300 points, which is negligible compared to total training time. Note that this result is a prediction of the training time of 10 batches, so to compute the training time of the actual millions of batches, the prediction output should be proportionally increased.

5.4.2. Time Cost into Acquisition Function

Now we can predict the training time even before training actual DNN with hyper-parameter vector \mathbf{x} from candidate set. Let's call this expected value as Expected Time (ET). An adaptive selection approach with four options has been implemented to consider EI and ET together. The four

ways we considered are:

- EI-0.3ET
- Min ET from top 10 EI
- EI/ET
- ET/log(ET)

Here, ‘Min ET from top 10 EI’ is an algorithm that reflects the intuition to select an option that is likely to take the shortest time among the candidates with sufficiently high EI. So, among top 10 candidates from EI, select the one with the smallest ET. These four options are chosen from 9 total options to ensure the different characteristic from each other. Model was fixed to the GP to all the cases, and acquisition function was fixed to EI. EI without time consideration is also performed as the benchmark. The performance within given time for each case are

summarized in Figure 17.

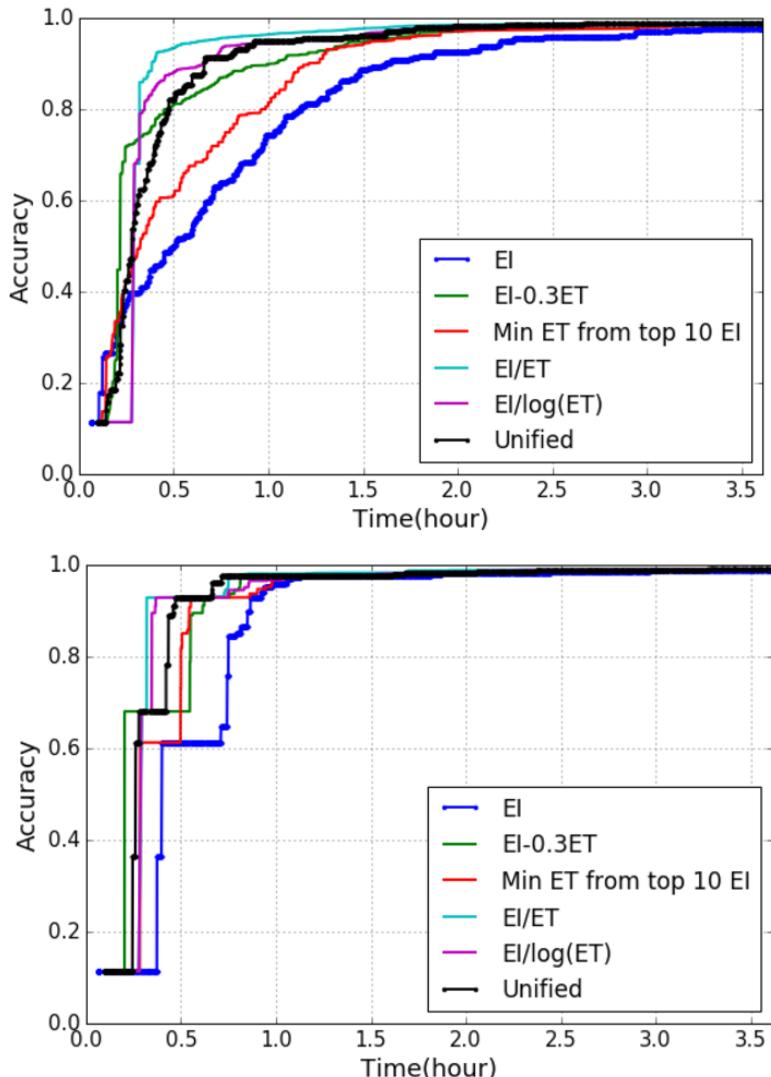


Figure 17. Mean (up) and Median (down) performance over time for 4 different ways to consider time cost and unified approach.

The result is a summary statistic from 100 repetitive experiments for all options, and 50 repetitive experiments for unified algorithm. According to Figure 17, the benchmark algorithm (EI curve) is always at the bottom, while all options considering time cost possess upper side. The unified algorithm also belongs to upper part, but it is not the top of one. This implies that considering time cost is mostly beneficial, until reaching quite high accuracy. But we cannot assure that it is always beneficial, since time goes after 3.5 hours, six curves are so close to each other. To confirm this, we presented the times results in Table 6.

Table 6. Time taken to reach 90%, 98%, 99% performance for time considered acquisition functions

	Mean			Median			Convergence Ratio
	90%	98%	99%	90%	98%	99%	
Unified	0.6	1.7	4.8	0.4	1.5	4.4	
EI	1.1	2.1	4.7	0.9	1.9	4.1	
EI-0.3ET	0.9	1.9	4.4	0.7	1.7	4.1	34%
Min ET from top 10 EI	0.8	1.9	4.7	0.6	1.7	4.3	18%
EI/ET	0.7	1.7	5.0	0.4	1.5	4.4	12%
EI/log(ET)	0.6	1.7	5.0	0.4	1.5	4.4	36%

Table 6 summarizes median and mean of the time it takes to reach high performance (90%, 98%, and 99%). With Table 6, we confirmed that until HPO reaches 98%, every time-considered acquisition functions

surpassed the benchmark (EI). In other words, time-considered acquisition functions more quickly achieve fairly good performances, and this is consistent with Figure 17. However, on the case of 99% accuracy, it is hard to say which one is better. For the mean, EI-0.3 ET is the best, while EI is the best for the median. As for unified algorithm, it ranks at first place for both 90% and 98%. In summary, the unified algorithm works properly, since it surpassed any other time-considered acquisition functions. Also, we confirmed that every time-considered acquisition functions has clear advantages in finding fairly good performance such as 90% and 98%. It drastically reduce the time consumption from 25% to 50%. However, we could not confirm that time-considered acquisition functions is also better at achieving very high performance points.

Table 7. Average number of trials in an hour

Algorithms	Unified	EI	EI-0.3ET	Min ET from top 10 EI	EI/ET	EI/ $\log(EI)$
Avg. # of trials in an hour	17.2	10.9	13.4	14.9	16.8	17.1

The performance boost of time-considered acquisition functions are stem from more frequent evaluation within equal amount of time, because it penalize heavy time-consuming hyper-parameter vectors. To examine

this idea, we recorded average number of trials in an hour for each of acquisition functions. Table 7 presents the result. It can be seen that algorithms with time concepts evaluate much more points from 30% to 70%.

Chapter 6. Conclusion

6.1 Summary

As deep neural networks are widely used, finding a well-performing hyper-parameter configuration has become an important issue, since the performance of the algorithm is sensitive to hyper-parameters. Hence, many HPO researches have actively conducted to solve problem. The most widely used method is the Bayesian optimization methodology. In Bayesian optimization, however, there exists too many options available for model and acquisition functions, and human must participate to choose one of them. It is very luck-dependent procedure because usually have any information about which optimization algorithm is proper for given problem. Therefore, we proposed a unified approach. Instead of using single model and single acquisition function during whole optimization process, it is a methodology that adapts to model and acquisition function to fit to the problem by selecting model and acquisition function at each trial. Decaying epsilon greedy algorithms is used for adaptation. As the results, there was no case of convergence to the worst case, and even it frequently surpass the performance of best single bandit. It indicates that unified approach could be used as a robust standard for model and acquisition function. In addition, the acquisition function was modified to take time cost into consideration. As the result,

the accuracy up to some level (90%, 98%) reached much faster than the benchmark algorithm, but at the level of highest accuracy (99%), we could not confirm the performance improvement.

6.2 Contributions

In our study, we believe there exist three contribution points for Bayesian Optimization of deep neural network.

- Proposal of unifying method for both models and acquisition functions
- Suggestion of method to accurately predict the training time of deep neural network
- A time-efficient Bayesian optimization method by allowing the acquisition function to take time-cost into consideration

Specifically, the unifying method that we designed first adapts according to the dataset and algorithm, and converges to the optimization method that works well among the various options, so choosing models and acquisition functions by user is no longer needed. Reducing additional choices within HPO is important because HPO's original intent is to avoid difficult intuition-based choices.

Secondly, we devised a method to accurately measure the training time of the DNN with a small amount of time, which is expected to have a

positive effect in subsequent experiments with DNN.

Finally, the acquisition function takes time cost into account, which shorten the time required to reach an appropriate level of performance.

6.3 Limitations

The limitations of this study are as follows. First, there exists limitation regarding designing reward. Currently, reward is the average accuracy of the bandit, which is a naïve approach. In fact, the purpose of HPO is to improve the accuracy, so the amount of improvement must be reflected in the reward. In the current reward system, the decaying epsilon greedy algorithm can avoid the worst case, but it cannot guarantee convergence to the optimal choice, and was partly confirmed in the experimental results as well. However, if the magnitude of improvement is applied as it is, the improvement of the reward in the initial search is too large. Therefore, it is necessary to design a reward system that can deal with the initial improvement and the later improvement equally.

Second, the multi-armed bandit assumes that the probabilistic distribution of the bandit does not change. However, Bayesian optimization changes the probability density function (pdf) of the reward according to the history of the points so far evaluated. In a small number of steps, it is a quasi-static state that maintains the original pdf to some extent. Currently, we do not consider this, but considering this, we can design a more accurate adaptation algorithm.

Finally, this study confirmed the results only for MNIST data, so it was not verified whether the insight found here could be transferred to other dataset. For generalization of our results, further experiments with other datasets is necessary.

References

- [1] E. Brochu, V.M. Cora, N. De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv:1012.2599, 2010.
- [2] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl. Algorithms for Hyper-Parameter Optimization. Advances in Neural Information Processing Systems 24, 2011.
- [3] J. Snoek, H. Larochelle, R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. Advances in Neural Information Processing Systems 25, 2012.
- [4] C. E. Rasmussen. Gaussian processes for machine learning. MIT Press, 2006.
- [5] F. Hutter, H. H. Holger, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. International Conference on Learning and Intelligent Optimization, 2011.
- [6] J. Bergstra, D. Yamins, and D. D. Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. International Conference on Machine Learning, 2013.
- [7] H. J. Kushner. A new method of locating the maximum of an arbitrary multipeak curve in the presence of noise. J. Basic Engineering, 86:97-106, 1964.
- [8] J. Mockus, V. Tiesis, and A. Zilinskas. Toward Global Optimization, volume 2, chapter The Application of Bayesian Methods for Seeking the Extremum, pages 117-128. Elsevier, 1978.

- [9] Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In Proc. International Conference on Machine Learning, 2010.
- [10] M. N. Katehakis, A. F. Veinott. The Multi-Armed Bandit Problem: Decomposition and Computation. Mathematics of Operations Research. 12 (2): 262–268, 1987.
- [11] M. Hoffman, Eric Brochu, Nando de Freitas. Portfolio Allocation for Bayesian Optimization. Conference on Uncertainty in Artificial Intelligence, 2011.
- [12] I. M. Sobol, Distribution of points in a cube and approximate evaluation of integrals. Zh. Vych. Mat. Mat. Fiz. 7: 784–802, 1967.
- [13] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, K. Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. NIPS workshop on Bayesian Optimization in Theory and Practice, 2013.
- [14] Y. LeCun, L. Bottou, Y. Bengio, and P Haffner. Gradient-Based Learning Applied to Document Recognition. Proc. of the IEEE, November, 1998.
- [15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. Conference on Computer Vision and Pattern Recognition, 2015.
- [16] B. Zoph, Q. V. Le. Neural Architecture Search with Reinforcement Learning, International Conference on Learning Representations, 2017.
- [17] J. Bergstra, Y. Bengio. Random search for hyper-parameter optimization. Journal of Machine Learning Research, 2012.

- [18] E. Parzen. On the estimation of a probability density function and the mode. *Annals of Math. Stats.*, 33:1065-1076, 1962.
- [19] K. Simonyan, A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015.
- [20] K. He, X. Zhang, S. Ren, J. Sun. Deep residual learning for image recognition. *Computer Vision and Pattern Recognition*, 2015.
- [21] C. Szegedy, S. Ioffe, V. Vanhoucke, A. Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *Computer Vision and Pattern Recognition*, 2016.
- [22] B. Baker, O. Gupta, N. Naik, R. Raskar. Designing Neural Network Architectures using Reinforcement Learning. *arXiv:1611.02167*, 2017.
- [23] T. Domhan, J. T. Springenberg, F. Hutter. Speeding up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves. *International Joint Conference on Artificial Intelligence*, 2015.
- [24] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, Mr Prabhat, R. Adams. Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:2171-2180, 2015.
- [25] A. Klein, S. Falkner, S. Bartels, P. Hennig, F. Hutter. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. *arXiv:1605.07079*, 2016.
- [26] M. Feurer, J. T. Springenberg, F. Hutter. Initializing Bayesian Hyperparameter Optimization via Meta-Learning. *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [27] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, A. Kurakin. Large-Scale Evolution of Image Classifiers.

arXiv:1703.01041, 2017.

- [28] R. S. Sutton, A. G. Barto. Reinforcement learning: an introduction. Cambridge, MA: MIT Press, 1998.
- [29] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285–294, 1933.
- [30] J. Vermorel, M. Mohri. Multi-armed bandit algorithms and empirical evaluation. In European Conference on Machine Learning, Springer, pp. 437–448, 2005.

국문 초록

심층 신경망 (Deep Neural Network)이 근래에 많은 연구 분야와 응용 분야에서 큰 성취를 거두며 동시에 심층 신경망의 하이퍼 파라미터를 최적화하는 것이 중요한 문제로 대두되었다. 컨볼루션 필터의 크기나 개수, 신경망의 깊이 등 여러 하이퍼 파라미터가 어떻게 설정 되는지 여부에 따라 신경망 알고리즘의 성능이 매우 민감하게 변하기 때문이다. 따라서 하이퍼 파라미터 최적화에 대한 연구가 활발히 진행되었으며 그 중에서 베이지안 최적화 방법론이 유망한 결과를 보여주었다. 그러나 베이지안 최적화에는 몇 가지 한계가 내재되어 있다. 첫째, 베이지안 최적화는 본래의 가치가 “선택의 자동화”에 있음에도 불구하고, 최적화 과정 안에서 모델링 알고리즘과 애퀴지션 함수 등 추가적으로 사람이 설정해줘야 하는 선택지들을 지니고 있다. 문제와 데이터셋에 가장 적합한 모델링 알고리즘과 애퀴지션 함수는 매번 다르며 어느 선택지가 좋은지 알 수는 없어서 이 선택이 어려운 문제이며, 현재는 문제에 따라 적절해 보이는 선택지를 적당히 고르고 운이 좋기를 희망하는 것이 전부이다. 둘째로, 애퀴지션 함수에는 시간 비용이 반영되지 않아 비효율적인 선택으로 이어진다. 후보군에 포함된 신경망의 학습에 필요한 시간을 미리 예측할 수 있다면 보다 비용 효율적인 선택을 할 수 있다.

이 두 가지 문제를 해결하기 위해 본인은 다음과 같은 방법을 제안하였다. 첫째, 기존의 모델과 애퀴지션 함수를 하나로 통합하여, 최적화 과정 안에서 문제와 데이터 셋에 적응하는 방식을 제안한다. 최적화 과정의 초기에 어떤 모델과 획득 기능이 잘 수행되는지 탐색 한 다음, 그 중에 좋은 성능을 보인 선택지로 수렴하는 방법론을 사용하였다. 둘째로 시간 비용을 고려한 애퀴지션 함수를 고안하였다. 이를 위하여 심층 신경망의 학습 시간을 예측할 수 있어야 하기 때문에, 심층 신경망의 학습 시간을 예측하는 빠르고 정확한 방법을 함께 제시하였다. 이 두 가지 방법을 바탕으로 베이지안 최적화를 개선하기 위해 노력했으며, 개선의 효과를 확인하기 위해 MNIST 데이터를 바탕으로 몇 가지 실험을 수행하였다. 실험을 통해 본인이 제안한 방법론이 두 가지 측면, 즉 시간에 따른 성능과 특정 성능에 도달하기 까지 걸리는 시간의 측면에서 개선됨을 확인하였다.

Keywords: 베이지안 최적화, 하이퍼 파라미터 최적화, 심층 신경망, 멀티암드 밴딧, 애퀴지션 함수

Student Number: 2015-26100