ACCELERATED BAYESIAN OPTIMIZATION FOR DEEP LEARNING

Ayahiko Niimi[1] and Kousuke Sakamoto[2]

[1]Faculty of Systems Information Science, Future University Hakodate,
2-116 Kamedanakano, Hakodate, Hokkaido 041-8655, Japan
[2]School of Systems Information Science, Future University Hakodate,
2-116 Kamedanakano, Hakodate, Hokkaido 041-8655, Japan

ABSTRACT

Bayesian optimization for deep learning has extensive execution time because it involves several calculations and parameters. To solve this problem, this study aims at accelerating the execution time by focusing on the output of the activation function that is strongly related to accuracy. We developed a technique to accelerate the execution time by stopping the learning model so that the activation function of the first and second layers would become zero. Two experiments were conducted to confirm the effectiveness of the proposed method. First, we implemented the proposed technique and compared its execution time with that of Bayesian optimization. We successfully accelerated the execution time of Bayesian optimization for deep learning. Second, we attempted to apply the proposed method for credit card transaction data. From these experiments, it was confirmed that the purpose of our study was achieved. In particular, we concluded that the proposed method can accelerate the execution time when deep learning is applied to an extremely large amount of data.

KEYWORDS

Deep Learning, Bayesian Optimization, Activation Function, Real Dataset

1. INTRODUCTION

In this paper, we propose a deep-learning automatic parameter-tuning method using an improved Bayesian optimization.

Deep learning is a new approach, which has recently attracted considerable attention in the field of machine learning. It considerably improves the accuracy of abstract representations by reconstructing deep structures, such as the neural circuitry of the human brain. Moreover, deep learning algorithms have been honored in various competitions, such as the International Conference on Representation Learning.

However, a problem with deep learning is that its performance cannot be optimized unless multiple parameters are appropriately tuned. Manual tuning of the parameters is performed based on the experience and intuition of experts. To address this problem, many studies have investigated automatic parameter tuning. Furthermore, it has been proposed that the learning algorithm's parameters can be determined by automatic parameter tuning. However, this requires the original machine-learning algorithm to be repeated multiple times. If applied to a heavy learning model, such as deep learning, this process becomes time-consuming.

Natarajan Meghanathan et al. (Eds) : ICAITA, Signal, SAI - 2017
pp. 01– 13, 2017. © CS & IT-CSCP 2017 DOI: 10.5121/csit.2017.71301

Therefore, in this study, we propose a deep-learning automatic parameter tuning method using an improved Bayesian optimization, which is one of the methods used in automatic parameter-tuning algorithms. The proposed method aims at accelerating the execution time of Bayesian optimization for deep learning. This method focuses on the relation between output values and the accuracy of each neuron. The output value of each neuron is obtained by substituting the total input into the activation function (Relu). If this value is 0, then the feature value is also 0, indicating that the neuron could not successfully extract the feature. In other words, the probability that learning does not proceed increases as the number of 0's in the output value of each neuron increases. From this fact, when learning is performed using this method, the number of 0s in the output value of each neuron excessively increases and learning stops. By performing learning using the following parameters, the execution time of Bayesian optimization is increased.

The novelty of this research is that Bayesian optimization is applied to deep learning. Bayesian optimization has been applied to light learning models, but there have been no studies applied to heavy learning models like deep learning. In this research, we explore the possibility of deep learning using Bayesian optimization through experiments with actual credit card data.

This study is summarized as follows: first, in section 2 and 3, we describe deep learning and Bayesian optimization. In section 4, we propose the improved Bayesian optimization for deep learning. In section 5 and 6, we describe the experiment and results of the study, respectively. Finally, in section 7, we discuss our conclusions and future work.

2. DEEP LEARNING

Deep learning is a new approach, which has recently attracted considerable attention in the field of machine learning.

Deep learning is a generic term for multilayer neural networks that have been studied over several years [1]-[3]. Multilayer neural networks reduce the overall calculation time by performing calculations on hidden layers. Therefore, these networks are prone to excessive overtraining because an intermediate layer is often used to approximate each layer.

Nevertheless, technological advancements have addressed the overtraining problem, while the use of GPU computing and parallel processing has increased the tractable number of hidden layers.

A sigmoid or tanh function has been commonly used as the activation function (Equations (1) and (2)). However, recently, the maxout function has also been used (section 2.1) and the dropout technique has been implemented to prevent overtraining (section 2.2).

$$h_i(x) = \text{sigmoid}(x^T W_{\cdots i} + b_i) \tag{1}$$

$$h_i(x) = \tanh(x^T W_{\cdots i} + b_i) \tag{2}$$

2.1. Maxout

The maxout model is simply a feed-forward architecture such as a multilayer perceptron or deep convolutional neural network that uses a new type of activation function, the maxout unit [3].

In particular, given an input $x \in \mathbb{R}^d$ (x may be either v or a hidden layer's state), a maxout hidden layer implements the function

$$h_i(x) = \max_{j\in[1,k]} z_{ij} \quad (3)$$

where $z_{ij} = x^T W_{\dots ij} + b_{ij}$, $W \in \mathbb{R}^{d\times m\times k}$ and $b \in \mathbb{R}^{m\times k}$ are learned parameters. In a convolutional network, a maxout feature map can be constructed by taking the maximum across \$k\$ affine feature maps (i.e., pool across channels, in addition to spatial locations). When training with the dropout, in all cases, we perform element-wise multiplication with the dropout mask immediately prior to multiplication by weights and the inputs not dropped to the max operator. A single maxout unit can be interpreted as a piecewise linear approximation of an arbitrary convex function. In addition to learning the relationship between hidden units, maxout networks also learn the activation function of each hidden unit.

The maxout approach abandons many of the mainstays of traditional activation function design. Even though the gradient is highly spare, the representation produced by maxout is not sparse at all, and the dropout will artificially sparsify the effective representation during training. Although the maxout unit may saturate on one side or another, this is a measure zero event (so it is almost never bounded from above). Because a significant proportion of the parameter space corresponds to the function delimited from below, maxout learning is not constrained. Moreover, the maxout function is locally linear almost everywhere, in contrast to many popular activation functions that demonstrate significant curvature. Considering all these deviations from thestandard practice, it may seem surprising for the maxout activation functions to work however, we find that they are very robust, easy to train with dropout, and achieve excellent performance.

$$h_i(x) = \max_{j\in[1,k]} Z_{ij} \quad (4)$$

$$z_{ij} = X^T W_{\dots ij} + b_{ij} \quad (5)$$

2.2. Dropout

Dropout is a technique that can be applied to deterministic feedforward architectures that predict an output y given an input vector v [3].

In particular, these architectures contain a series of hidden layers $h = \{h(1), \cdots, h(L)\}$. Dropout trains an ensemble of models comprising a subset of the variables in both v and h. The same set of parameters θ are used to parameterize a family of distributions $p(y \mid v;\ \theta, \mu)$, where $\mu \in \mathcal{M}$ is a binary mask determining which variables to include in the model. For each example, we train a different submodel by following the gradient $\log p(y \mid v; \theta, \mu)$ for a different randomly sampled μ. For many parameterizations of p (usually for multilayer perceptrons) the instantiation of the different submodels $p(y \mid v;\ \theta, \mu)$ can be obtained by element-wise multiplication of v and h with the mask μ.

The functional form becomes important when the ensemble makes a prediction by averaging the submodels' predictions. Previous studies of bagging averages used the arithmetic mean. However, this is not possible with the exponentially large number of models trained by dropout. Fortunately, some models can easily yield a geometric mean. When $p(y \mid v;\ \theta) = \mathrm{softmax}(v^T W + b)$, the predictive distribution defined by renormalizing the geometric mean of $p(y \mid v;\ \theta, \mu)$ over $\mathcal{M}$ is simply given by $\mathrm{softmax}(v^T W/2 + b)$. In other words, the average exponential prediction for many submodels can be computed simply by running the full model with the weights divided by two. This result holds exactly in the case of a single layer softmax model. Previous work on dropout applies the same scheme to deeper architectures, such as multilayer perceptrons, where the $W/2$ method provides only an

approximation of the geometric mean. While this approximation is not mathematically justified, it performed well in practice.

3. BAYESIAN OPTIMIZATION

The automatic parameter-tuning method automatically determines the machine-learning parameters. Parameters are considerably important in machine learning, and the accuracy significantly varies depending on how the parameters are set. However, a great amount of knowledge and experience is required to determine appropriate parameters. Automatic parameter tuning can complement knowledge and experience and make machine learning easier to handle.

Grid search is one of the automatic parameter-tuning techniques. In this technique, multiple parameter combinations are made to determine the parameter with the best accuracy. However, the problem with grid search is that there is a possibility that some parameter combinations which were not tested may actually be the best parameters. However, if the number of combinations of the tested parameters is increased, the execution time will be prolonged.

Another automatic parameter-tuning technique is Bayesian optimization. In this technique, several parameters are tested and the parameter combinations are determined by predicting the accuracy of parameter combinations that have not been tested. This method determines the parameters by predicting a combination of multiple parameters and a parameter having a high probability of being the most accurate one based on multiple parameters and their precision.

1) Predict the prior distribution of parameter accuracy in the Gaussian process.

2) Predict the posterior distribution from prior distribution by Bayesian estimation.

3) Search the combinations of parameters with the highest accuracy from posterior distribution using the Markov chain Monte Carlo (MCMC) method.

4) Repeat steps 1)–3) to select the most accurate parameter.

In the Gaussian process, the prior distribution was assumed to be a normal distribution. From the obtained data, the probability that data is obtained under certain parameters was defined as likelihood. The Gaussian process uses the fact that the likelihood approaches the posterior distribution. To find parameters with high probability of high accuracy from the accuracy of multiple models, there are many methods. In this paper, we use Metropolis-Hastings (MH) method.

In such a procedure, it is possible to predict a combination of parameters with the highest accuracy and high probability. The problem in Bayesian optimization is that the combination of the predicted parameters may not be the best parameter.

This method is a method for predicting accuracy and parameter prior distribution. The method is performed in the following procedure. This method predicts the accuracy and parameter prior distribution. It has the following procedure:

1) Find the accuracy of multiple parameters.

2) Assume that the accuracy of the obtained parameters follows the normal distribution and predict the prior distribution.

By implementing this procedure, prior distribution can be predicted.

Bayesian estimation is one of the methods used for Bayesian optimization. This method predicts the accuracy and posterior distribution of parameters. In this method, the posterior distribution is obtained using the Bayes' theorem from the accuracy of the prior distribution, which is obtained using the Gaussian process and multiple parameter combinations. Using Bayesian estimation, the posterior distribution can be predicted.

The MCMC method is one of the methods used for Bayesian optimization [4]. This method searches a parameter combination with the highest accuracy in the posterior distribution. This method is a combination of the Markov chain and Monte Carlo method, each of which is explained as follows:

- Markov chain: The next state is determined by the previous state; however, it is not affected by the previous state.

- Monte Carlo method: This method generates random numbers from various probability distributions.

The MCMC method involves the use of various methods, such as the Guinness sampling and Metropolis–Hastings (MH) method. The MH method is generally used in this study and has the following procedure:

1) Determine the transition destination of the point where multiple parameters obtained by the Gaussian process are combined.

2) If the transition destination point has a higher probability of higher accuracy than the transition point before the transition, the transition is rejected with a certain probability when the probability of high accuracy is low. The reason for rejection is that it can enter a different mountain by transitioning to a point with lower precision and can get off the mountain of the local solution.

By repeating such a procedure, it is possible to search parameter combinations with a high probability of high accuracy in the posterior distribution.

The difference between grid search and Bayesian optimization is whether or not to consider parameters that have not been tested. In grid search, parameters are tested at regular intervals; however, because there are parameters that are not tested, it is possible that parameters with the best accuracy exist among these parameters. Bayesian optimization compensates for the defects of grid search in order to estimate the parameters with the highest accuracy among all the parameters from the accuracy of multiple and select parameters. However, as the estimated parameter accuracy is not necessarily the highest, there is a possibility of choosing a wrong parameter if the trial count is small.

This study deals with one of the methods used to accelerate Bayesian optimization by aiming to speed up the MCMC method by locally creating an approximate posterior distribution [5]. The MCMC method creates a posterior distribution from multiple parameters and precision values, determines the next transition destination, and obtains the accuracy of the transition destination parameter through the model; however, this operation is time-consuming. Therefore, speeding it up will also speed up the operation of the MCMC method.

4. PROPOSED METHOD FOR ACCELERATING THE MCMC METHOD

In this research, the proposed method aims to accelerate the execution time of Bayesian optimization for deep learning. It focuses on the relation between output values and the accuracy of each neuron. The output value of each neuron is obtained by substituting the total input into the activation function (Relu). If this value is 0, the feature value is also 0, indicating that the neuron was not successful in extracting the feature. In other words, the probability that learning does not proceed increases as the number of 0s in the output value of each neuron increases. Consequently, when learning is performed, the number of 0's in the output value of each neuron excessively increases and learning is stopped. By performing learning with the following parameters, the execution time of Bayesian optimization is increased.

The feature of the proposed method is to assess when learning becomes excessive and then stop it. Normal Bayesian optimization follows the procedure summarized below:

1) Find the accuracy of the deep-learning model with randomly determined parameters.

2) Find parameters with high probability of high accuracy from the accuracy of multiple models.

3) Calculate the accuracy of the deep-learning model with the obtained parameters.

4) Accurately compute parameters with high accuracy by adding the obtained parameters and accuracy.

5) Repeat steps 3) and 4) multiple times.

6) Select the parameter with the highest accuracy.

In this method, low-accuracy learning is performed for a number of learning times, which is time-consuming. The proposed method solved this problem and follows the procedure summarized below:

1) Find the accuracy of the deep-learning model with randomly determined parameters
.
2) Find parameters with high probability of high accuracy from the accuracy of multiple models.

3) Calculate the accuracy of the deep-learning model with the obtained parameters. At this time, the output of the first and second layers of the model is obtained; learning which has not advanced is stopped.

4) Accurately compute parameters with high accuracy by adding the obtained parameters and accuracy.

5) Repeat steps 3) and 4) multiple times.

6) Select the parameter with the highest accuracy.

At the point where step 3) is different from normal Bayesian optimization, by stopping learning that has not advanced by performing this procedure, the proposed method can increase the execution time.

We used Python as the implementation language. The reason for using Python is because it can use TensorFlow and because it contains libraries that allow fast calculation of NumPy arrays.

In addition, a program for deep-learning modeling called TensorFlow was used because it allows calculating only the necessary parts when obtaining the output value of the neuron necessary for implementing the proposed method. This is because the speed can be increased.

Implementation was performed according to the following procedure.

1) Create a program to develop deep-learning models for deep learning.

2) Prepare a program to calculate the number of neuron output value 0 from the deep-learning model.

3) Create a program that can stop the model to learn from the number of neuron's 0 output values.

4) Create a Bayesian optimization program.

5) Combine a program that can stop the model to learn from the number of neuron's 0 output values and a program of Bayesian optimization.

Development was conducted in the following environment:

Table 1. The Development Environment

OS	**Memory**	**# of CPU**	**Python**	**TensorFlow**
Amazon Linux	1GB	1	Anaconda 3-4.2.0	0.11.0

TensorFlow is an open-source software library for numerical calculation using a data flow graph. As it was developed for machine-learning applications and deep-learning research, it comes with many functions essential for deep learning. TensorFlow was utilized due to its suitability to the intended purpose of this study [6].

TensorFlow can speed up calculation time by performing only the necessary calculations. We will explain the procedure when we want to obtain the output y(x) of the intermediate layer of the three-layer neural network, as shown in Fig.1. Calculations are made according to the following procedure.

1) First, define the number of neurons in each layer, weight (w), bias (b), weight adjustment method (e.g., gradient descent method), activation function, and calculation formulae.

2) Assign training data and real values, such as weight w, to the definition.

3) We used weight (w), bias (b), and input (x) from the output y(x) calculation formula.

4) Construct the minimum necessary model from the judged contents. In other words, create a model with layers other than the output layer.

By design, TensorFlow makes it possible to assess which calculations are essential and perform those calculations only.

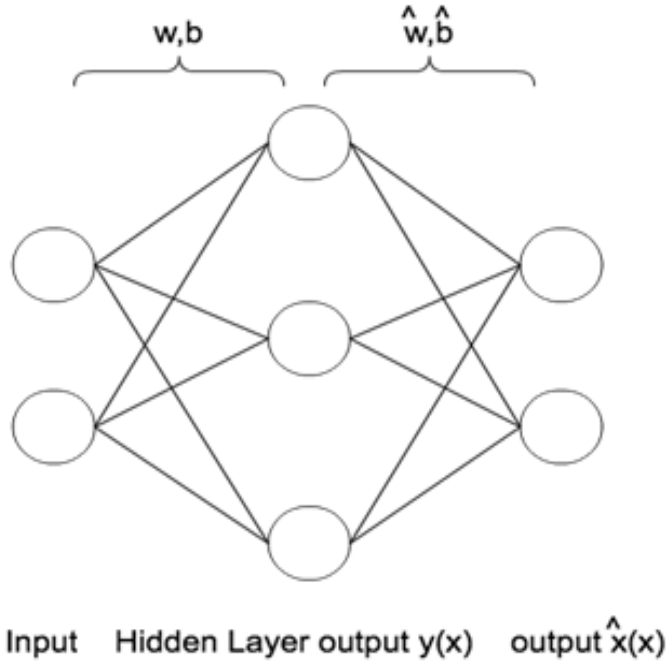

Figure 1. Auto Encoder

5. EXPERIMENT 1: IRIS DATASETS

To verify the effectiveness of the proposed method, we used the Iris dataset to conduct an experiment [7]. The dataset contained 3 classes of 50 instances each, where each class referred to a type of an iris plant. (Table 2)

Table 2. The Iris Dataset

# of data	# of training	# of test	# of attributes	# of classes
150	120	30	4	3

Bayesian optimization and the proposed method were performed 5 times. The parameters determined by Bayesian optimization are the Neuron number (1 - 30), the learning coefficient of the gradient descent method (0.1, 0.2, and 0.3), and the number of learning (2 - 1001). The experimental results are summarized in Tables 3 and 5.

Table 3. The Result of Bayes Oprimization (Iris)

# of learning	# of neurons	Accuracy	Calculation time(s)
28206	772	0.9833	153.7447
27202	782	0.9833	154.5802
27248	797	0.9833	153.6453
26004	796	0.9833	151.6539
27259	805	0.9833	154.4179

Table 4. Result of the Proposed Method (Iris)

# of learning (Original)	# of learning (Actual)	# of learning (Reduction)	# of Models (Low Accuracy)	# of total nurons	Accuracy	Calculation time(s)
27316	16256	11060	17	831	0.9916	145.9402
27212	16195	11017	21	795	0.9833	145.6388
29067	23083	5984	10	760	0.9833	150.9196
27035	19124	7911	13	820	0.9916	145.0893
26590	18446	8144	13	822	0.9833	146.4700

Based on the results, a speed increase of approximately 9s without loss in precision was observed. However, because the dataset and the decrease in the time needed were small, we considered testing our method using a larger dataset; therefore, we conducted experiments with credit card transaction data.

6. EXPERIMENT 2: CREDIT CARD TRANSACTION DATASET

The datasets for credit card transactions are as follows:

1) Credit approval dataset

2) Card transaction dataset

6.1. Credit approval dataset

For each user submitting a credit card application, a record of the decision to issue the card or reject the application is maintained. This is based on user attributes in accordance with general usage trend models.

However, to reach this decision, it is necessary to combine multiple models, each referring to a different clustered group of users.

6.2. Credit card transaction data

In actual credit card transactions, the data is complex, constantly changing, and continuously arriving online as follows:

i. Data of approximately one million transactions arrive daily.
ii. Each transaction takes less than 1 s to complete.
iii. Approximately 100 transactions are done per second during peak time.
iv. Transaction data arrive continuously.

Therefore, it is accurate to consider credit card transaction data as a stream. However, even if we use data mining for such data, an operator can only monitor approximately 2,000 transactions per day. Thus, suspicious transaction data should be effectively detected by analyzing less than 0.02% of the total number of transactions. In addition, fraud detection from the analysis of massive amounts of transaction data is extremely low because real fraud occurs at an extremely low rate, i.e., within a range of 0.02% - 0.05% of the amount of all analyzed transaction data.

In a previous study, the transaction data in CSV format were described as being attributed to a time order [8]. Credit card transaction data have 124 attributes, of which 84 are transactional data, including an attribute used to indicate fraudulent activity. The remaining data are behavioral data and are relevant to credit card usage. The inflow file size is approximately 700 MB per month.

Mining the credit card transaction data stream involves difficulties as it requires performing efficient calculations on an unlimited data stream with limited computing resources; therefore, many stream-mining methods seek an approximate or probabilistic solution instead of an exact one. However, as actual unauthorized credit card use is very less, these imprecise solutions do not appropriately detect frauds.

We are currently researching how to apply credit card data to ordinary deep learning [9], [10].

We apply the method proposed in this paper to credit card transaction data. As experiments cannot be conducted at universities due to security issues, we collaborate with non-academic researchers.

6.3. Credit card transaction data (Random Dataset)

In this study, the proposed method was also validated using a large credit card transaction dataset. We constructed this dataset from the actual credit card transaction dataset, which contained 129 attributes with random values within the same range specified for each attribute. The dataset contained approximately 32,000 transactions, including approximately 218 instances of illegal usage. While pre-processing deleted attributes, including null value and character string attributes, the number of attributes changed from 129 to 64 attributes. Because this dataset contained random values, it cannot be used to evaluate accuracy. Instead, experiments were conducted to confirm whether the calculation time could be reduced using the proposed method.

The percentage of fraudulent transactions in the dataset was considerably low. In the experiment, we used all illegal activity occurrences (218 instances) and a sample of normal usage activity (218 instances). The results of the experiments are summarized in Tables 5 and 6.

Table 5. Result of Bayes Oprimization (Credit Card)

# of learning	# of neurons	Accuracy	Calculation time(s)
14942	1526	0.5481	208.9575

Table 6. Result of the Proposed Method (Credit Card)

# of learning (Original)	# of learning (Actual)	# of learning (Reduction)	# of Models (Low Accuracy)	# of total nurons	Accuracy	Calculation time(s)
16174	11657	4517	7	1568	0.5298	188.7191

Based on the experimental results, our method could speed up the execution time by approximately 20s while maintaining the same precision. This result showed that it is highly probable that the proposed method can speed up the execution time if the amount of data is large.

6.4. Credit card transaction data (Real Dataset)

In this paper, we apply the proposed method for real transaction dataset from real system.

Because all the data is enormous, sampled data was used. The use samplinged dataset as follow (see in Table 7)

Table 7. Credit Transaction Dataset

Number of Instances:	120,000
Number of Attributes	125
Number of Instance for Training:	20,000
Class Distribution for Training:	illegal : 382 (1.91%), legal : otherwise.
Number of Instance for Test:	100,000
Class Distribution for Test:	illegal : 1148 (1.148%), legal : otherwise.

Verification was conducted while changing the number of sampling.

In the experiment, we use the following environment.

i. OS Linux (VM on Windows7 64bit)
ii. CPU Intel i303229 3.30 GHz
iii. Memory 4GB
iv. Disk 500GB

We used the following indexes for evaluation.

i. Correct answer rate: Among the transactions deemed to be illegal, the ratio of transactions that were illegal

ii. Detection rate: The ratio of transactions judged to be invalid among all illegal transactions

Table 8 shows the experimental results.

As for the tuning (sampling 1), since there was a date to distinguish all test data as illegal, the result rate was low. For tuning (sampling 2), the detection rate was lower than Deep Learning alone, but the accuracy rate slightly improved. Since the number of trials is small, it is necessary to repeatedly verify in order to obtain a general-purpose result.

Table 8. Result of the Proposed Method (Real Credit Card)

	Deep Learning (10 times learning)	**Deep Learning (20 times learning)**	**Auto-Tuing (sampling 1)**	**Auto-Tuning (sampling 2)**
# of Illegal Judgment	2986	1862	21887	1473
# of True Illegal in Judgement	567	574	608	466
Correct Answer Rate	18.99%	30.83%	2.78%	31.63%
Detection Rate	49.39%	50.00%	52.96%	40.59%

7. CONCLUSIONS

In this paper, we proposed a deep-learning automatic parameter-tuning method with improved Bayesian optimization.

Deep learning is a new approach, which recently attracted considerable attention in the field of machine learning. However, a problem with deep learning is that it cannot demonstrate satisfactory performance unless multiple parameters are appropriately tuned. Bayesian optimization for deep learning is time-consuming because it involves several calculations and parameters. To solve this problem, this study aimed to accelerate the execution time by focusing on the activation function's output that is strongly related to accuracy. In this study, we developed a technique to accelerate the execution time by stopping the learning model so that the activation function of the first and second layers would become zero. Two experiments were conducted in order to confirm the effectiveness of the proposed method. We first used the Iris dataset, implemented the proposed method, and compared its execution time with the execution time of Bayesian optimization. Based on the experimental results, the proposed method could accelerate the execution time by approximately 9s while maintaining the same precision. Therefore, we were successful in accelerating the execution time of Bayesian optimization for deep learning. Second, we applied our proposed method to analyze the credit card transaction data. Based on the experimental results, our method could accelerate the execution time by approximately 20s while maintaining the same precision. The results of the experiments demonstrate that the purpose of this study was achieved. In particular, we concluded that the proposed method can achieve a faster acceleration when deep learning is applied to an extremely large amount of data.

In the future, we will apply the proposed method to analyze the actual credit card transaction data and verify the acceleration effect with real large-scale data. We will also consider improving the Bayesian estimation algorithm.

ACKNOWLEDGEMENTS

The authors would like to thank Intelligent Wave Inc. for their comments on credit card transaction datasets.

This work was supported by JSPS KAKENHI Grant Number JP17K00310.

REFERENCES

[1] Y. Bengio, "Learning deep architectures for AI," Foundations and Trends in Machine Learning, vol. 2, no. 1, pp. 1–127, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1561/2200000006

[2] Q. Le, "Building high-level features using large scale unsupervised learning," in Acoustics, peech and Signal Processing (ICASSP), 2013 IEEE International Conference on, May 2013, pp. 8595–8598.

[3] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout Networks," ArXiv e-prints, Feb. 2013.

[4] S. Chib, "Markov chain monte carlo methods: computation and inference," Handbook of econometrics, vol. 5, pp. 3569–3649, 2001.

[5] P. R. Conrad, Y. M. Marzouk, N. S. Pillai, and A. Smith, "Accelerating Asymptotically Exact MCMC for Computationally Intensive Models via Local Approximations," ArXiv e-prints, Feb. 2014.

[6] Tensorflow. (Access Date: 6 April, 2017). [Online]. Available: https://www.tensorflow.org/

[7] M. Lichman, "UCI machine learning repository," 2013, (Access Date: 6 April, 2017). [Online]. Available: http://archive.ics.uci.edu/ml

[8] T. Minegishi and A. Niimi, "Proposal of credit card fraudulent use detection by online-type decision tree construction and verification of generality," International Journal for Information Security Research (IJISR), vol. 1, pp. 229–235, 2013.

[9] A. Niimi, "Deep learning for credit card data analysis," in World Congress on Internet Security (WorldCIS-2015), Dublin, Ireland, Oct. 2015, pp. 73–77.

[10] A. Niimi, "Deep learning with large scale dataset for credit card data analysis," in Fuzzy Systems and Data Mining II, Proceedings of FSDM 2016, ISO Press, Macau, China, Dec. 2016, pp. 149–158.

AUTHORS

Ayahiko Niimi is Associate Professor of Future University Hakodate, and Kousuke Sakamoto is master student of its University.