Simplified Influence Evaluation of Additional Training on Deep Neural Networks

Naoto Sato, Hironobu Kuruma, Yuichiroh Nakagawa, and Hideto Ogawa
Research & Development Group, Hitachi, Ltd.
{naoto.sato.je, hironobu.kuruma.zg, yuichiroh.nakagawa.hk, hideto.ogawa.cp}@hitachi.com

***Abstract*—During operation of a system including a deep neural network (DNN), new input values to the DNN that were not included in the initial dataset retained during development may be given. In such a case, the DNN may be additionally trained with the new input values; however, that additional training may reduce the accuracy of the DNN in regard to the initial dataset. It is therefore necessary to evaluate the influence of the additional training on the accuracy for the initial dataset. However, evaluation by testing all the input values included in the initial dataset takes time. In this paper, we newly propose a method for quickly evaluating the influence on the accuracy for the initial dataset. As for the proposed method, the gradient of the parameter values (such as weight and bias) for the initial dataset is extracted by running the DNN before the additional training. Then, after the additional training, its influence on the accuracy with respect to the initial dataset is calculated from the gradient and update differences of the parameter values. To show the feasibility of the proposed method, results of experiments with the MNIST dataset are presented, Accordingly, it is confirmed that the calculation amount of the proposed method after the additional training depends on the number of parameters; therefore, even if the number of input values included in the initial dataset is enormous, the influence of additional training can be evaluated quickly.**

***Index Terms*—machine learning, neural networks, regression analysis**

I. Introduction

In recent years, the introduction of machine-learning technologies in various industrial fields has been advancing. Among those technologies, deep neural networks (DNNs) are being popularly applied. In addition to replacing human tasks, in some fields, DNNs are demonstrating better ability than people. DNNs are trained with a dataset. A dataset is composed of pairs of input values to DNNs and their corresponding expected output values. Part of the dataset is used as training dataset for training DNNs, and the rest of the dataset is used as test dataset to evaluate the trained DNNs. In the training of DNNs, when input values included in the training dataset are input into DNNs, values of parameters such as weights and bias are adjusted so that the possibility of obtaining the expected output values increases. After the training is completed, the test dataset is used to measure the probability of obtaining the expected output value as expected. This probability—called accuracy—is an indicator showing the validity of the developed DNNs. However, the accuracy measured during development is only that with respect to the initial dataset retained at that time. That is to say, when input values that are not included in the initial dataset are given, the output values are not exclusively those expected; consequently, the accuracy of the DNN during operation is lower than that during its development.

When the accuracy of DNNs decreases, the DNNs may be subjected to additional training during operation [1] [2]. As for additional training, operating DNNs are trained by using input values and their expected values newly acquired during their operation. In this paper, a set of pairs of input values used for additional training and their expected output values is called an *additional dataset*. By additional training, parameter values (such as weight and bias) are adjusted so as to improve the accuracy concerning the additional dataset. However, The result of the adjustment also influences the accuracy of the initial dataset. Accordingly, it is necessary to evaluate not only the accuracy for the additional dataset but also the accuracy for the initial dataset. As for the result of that evaluation, in the case that the accuracy with respect to the initial dataset decreases, either training hyperparameters are changed and the additional training is repeated, or rollback to the DNNs before additional training is executed. Since it is assumed that additional training is performed during operation of DNNs, it is preferable to be able to evaluate the results of the additional training in as short a time as possible. To grasp the influence of additional training on the accuracy for the initial dataset, it is sufficient to run the DNNs additionally trained with all the input values included in the initial dataset and acquire their accuracy again. However, when the number of input values in the initial dataset is huge, it takes a long time to execute that test. In other words, the accuracy of the DNNs cannot be evaluated quickly.

In this paper, a method for quickly evaluating the influence of additional training on the accuracy for the initial dataset—after completion of the additional training—is proposed. In regard to the proposed method, the gradient of the parameter values is extracted by executing the DNNs with input values in the initial dataset before the additional training. Then, after the additional training, its influence on the accuracy for the initial dataset is calculated from the gradient and the update differences of parameter values resulting from the additional training. The results of experiments using the proposed method with the MNIST dataset confirmed that the amount of calculation to be performed after additional training does not depend on the number of input values in the initial dataset; instead, it depends on the number of parameters used in the DNN. As a result, even if the number of input values in the initial dataset is huge, applying the proposed method makes it possible to evaluate the result of additional training quickly. Moreover, the result of evaluating the influence of additional training by the proposed method was compared with the result of evaluating it by test execution. On the basis of the results of that comparison, the utility of the proposed method was also confirmed.

II. Background

A. Deep Neural Networks

For an arbitrary DNN, denoted as N, handling a classification problem of class $c(c > 1)$, I is taken as the number of neurons making up N, and each neuron (in any layer) is denoted as $n_i(1 \leq i \leq I)$. Also, J_i is taken as the number of parameters used to calculate the value of n_i, and the parameter itself is denoted as $[w_{i,1}, ..., w_{i,j}, ..., w_{i,J_i}]$. Note that if n_{i_1} and n_{i_1} are included in different layers, J_{i_1} and J_{i_2} can be different. Then, a vector obtained by combining the parameters of all the neurons is defined as $W = [w_{1,1}, ..., w_{1,J_1}, w_{2,1}, ..., w_{2,J_2}, ..., w_{I,1}, ..., w_{I,J_I}]$. In the case of a multilayer perceptron, for example, these parameters correspond to weights and biases. Note that the formula for calculating the value of each neuron by using these parameters is not prescribed in this paper.

For an arbitrary input value x^m, the expected output value corresponding to x^m is expressed as $t(x^m)$. $t(x^m)$ represents the identifier of the classification class to which x^m belongs. Moreover, when x^m is input to N, the output value returned by N is expressed as $y(x^m)$, which corresponds to the c-dimensional vector $[y_1^m, ..., y_k^m, ..., y_c^m]$. The y_k^m value for each dimension represents the probability that the input value x^m belongs to class k. If y_k^m has the largest value in $[y_1^m, ..., y_k^m, ..., y_c^m]$, class k is denoted as $fst(y(x^m))$. That is, $\forall y_k^m \cdot y_k^m \leq y_{fst(y(x^m))}^m$ holds. When $fst(y(x^m)) = t(x^m)$ holds, it is said that N can correctly classify x^m. Likewise, if y_k^m has the second-largest value, class k is denoted as $snd(y(x^m))$. That is, $\forall y_k^m \cdot y_k^m \neq y_{fst(y(x^m))}^m \Rightarrow y_k^m \leq y_{snd(y(x^m))}^m$ holds.

B. Additional Training

The flow of additional training assumed in this paper is shown in Fig. 1.

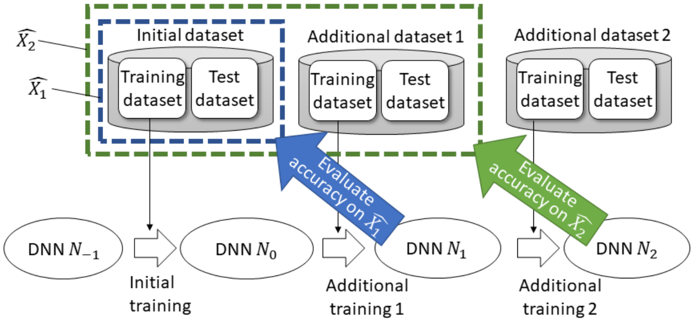

Fig. 1. Flow of additional training

In this flow, untrained DNN N_{-1} is trained first by using the training dataset of initial dataset. The result of that training is evaluated by using test dataset of the initial dataset. After that, the resulting DNN, N_0, starts to be used. During its use, additional dataset 1 is newly acquired. Then, N_0 is additionally trained using the training dataset that is a subset of additional dataset 1, and the result is evaluated in the same way using the corresponding test dataset. The resulting DNN, N_1, achieves sufficient accuracy for additional dataset 1. However, N_1 does not always maintain sufficient accuracy for the initial dataset. Accordingly, it is necessary to evaluate the influence of the additional training on the accuracy for the initial dataset. That necessity holds for the following additional trainings. As for additional training $s(1 \leq s)$, if a dataset whose influence should be evaluated is expressed as $\hat{X}_s$, $\hat{X}_s$ is defined as follows:

Definition 1:

$$\hat{X}_s = \begin{cases} ID & (s = 1) \\ ID \cup \bigcup_{s'=1}^{s-1} AD_{s'} & (otherwise) \end{cases}$$

where ID and $AD_{s'}$ represent the initial dataset and additional dataset of additional training s' respectively.

III. Proposed Method

Additional training 1 explained in Section II-B is used as a running example.

A. Positive and Negative Gradients

As for the proposed method, prior to additional training 1, preprocessing is performed on DNN N_0. It is supposed that N_0 deals with a classification problem with class $c(c > 1)$. Hereafter, as for the expected output value $t(x^m)$ for an arbitrary input value x^m to N_0, *positive supervisor* $ps(t(x^m))$ and *negative supervisor* $ns(t(x^m))$ are defined as follows:

Definition 2:

$$\begin{aligned} ps(t(x^m)) &= t(x^m) \\ ns(t(x^m)) &= \begin{cases} snd(y(x^m)) & (fst(y(x^m)) = t(x^m)) \\ fst(y(x^m)) & (otherwise) \end{cases} \end{aligned}$$

where $y(x^m)$ represents the output value returned by N_0 when x^m is input to N_0. At that time, as for all input values included in arbitrary-input-value set $X_1 \subseteq \hat{X}_1$, *positive loss* that is loss under the assumption that positive supervisor is the supervisory signal of the training, and *negative loss* that is loss under the assumption that negative supervisor is the supervisory signal of the training are calculated. Average positive loss for X_1 is denoted as $PL(X_1)$. Similarly, average negative loss is denoted as $NL(X_1)$. For an arbitrary loss function expressed as $L(y(x^m), t(x^m))$, $PL(X_1)$ and $NL(X_1)$ are defined as follows:

Definition 3:

$$\begin{aligned} PL(X_1) &= \frac{1}{M}\sum_{x^m}^{X_1} L(y(x^m), ps(t(x^m))) \\ NL(X_1) &= \frac{1}{M}\sum_{x^m}^{X_1} L(y(x^m), ns(t(x^m))) \end{aligned}$$

where M represents the number of input values included in X_1. Next, the gradient of parameter $W_0 = [w_{1,1}, ..., w_{i,j}, ..., w_{I,J_I}]$ of N_0 with respect to $PL(X_1)$ (obtained as previously described) is calculated. The gradient of $PL(X_1)$, called *positive gradient*, is expressed by $\nabla PL(X_1)$. Similarly, the gradient of $NL(X_1)$, called *negative gradient*, is expressed by $\nabla NL(X_1)$. $\nabla PL(X_1)$ and $\nabla NL(X_1)$ are defined as follows:

Definition 4:

$$\begin{aligned} \nabla PL(X_1) &= \left[\frac{\partial PL(X_1)}{\partial w_{1,1}}, ..., \frac{\partial PL(X_1)}{\partial w_{i,j}}, ..., \frac{\partial PL(X_1)}{\partial w_{I,J_I}}\right] \\ \nabla NL(X_1) &= \left[\frac{\partial NL(X_1)}{\partial w_{1,1}}, ..., \frac{\partial NL(X_1)}{\partial w_{i,j}}, ..., \frac{\partial NL(X_1)}{\partial w_{I,J_I}}\right] \end{aligned}$$

B. Influence of Additional Training

After $\nabla PL(X_1)$ and $\nabla NL(X_1)$ are created for N_0, additional training 1 is executed, and DNN N_1 is created. Parameter W_1 of N_1 is compared with parameter W_0 of N_0, and the update difference of the parameter $\Delta W = W_1 - W_0$, is acquired. Then, positive influence $PI(X_1, \Delta W)$ and negative influence $NI(X_1, \Delta W)$ given to X_1 by updating the parameter is calculated as follows:

Definition 5:

$$\begin{aligned} PI(X_1, \Delta W) &= (-\nabla PL(X_1)) \cdot (\Delta W) \\ NI(X_1, \Delta W) &= (-\nabla NL(X_1)) \cdot (\Delta W) \end{aligned}$$

$PI(X_1, \Delta W)$ approximates the amount by which $PL(X_1)$ is decreased by updating parameter W_0 to W_1. Likewise, $NI(X_1, \Delta W)$ corresponds to the approximate value of the decrease in $NL(X_1)$. If ΔW is denoted as $[\Delta w_{1,1}, ..., \Delta w_{i,j}, ..., \Delta w_{I,J_I}]$, $PI(X_1, \Delta W)$ corresponds to $\sum_{i,j}\left(-\frac{\partial PL(X_1)}{\partial w_{i,j}} \times \Delta w_{i,j}\right)$. For example, in the case that loss function L is cross entropy, the relation between the value of the decrease of in $PL(X_1)$ due to updating $w_{i,j}$ by $\Delta w_{i,j}$, and its approximate value $-\frac{\partial PL(X_1)}{\partial w_{i,j}} \times \Delta w_{i,j}$ is shown in Fig. 2.

Here, it is assumed that L is the loss function used for the initial training to create N_0 from N_{-1}. The initial training aims to reduce the loss calculated based on L (called training loss hereafter), in which the expected output value is the supervisory signal of the training. As a result, it is assumed that the accuracy of the DNN in regard to X_1 is improved. In that case, the value of the decrease in training loss is considered to be an index for evaluating the change of accuracy with respect to X_1. According to Definition 2, since the positive supervisor is the same as the expected output value, $PL(X_1)$ coincides with the training loss. It can therefore be assumed that $PI(X_1, \Delta W)$, which is an approximate value of the decrease in $PL(X_1)$, can be taken as an index for evaluating the change in accuracy with respect to X_1.

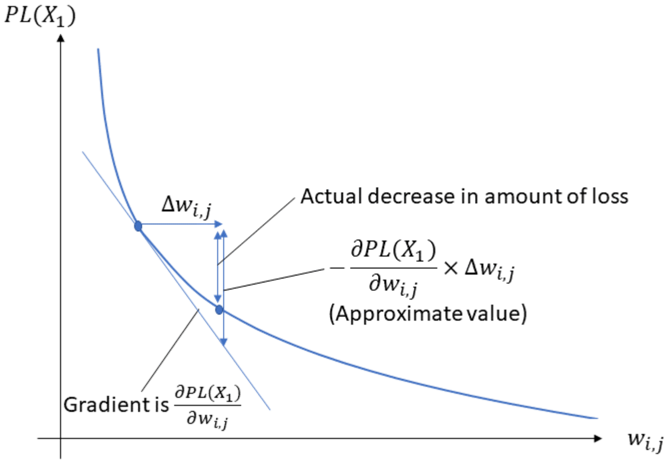
$PL(X_1)$
$\Delta w_{i,j}$
Actual decrease in amount of loss
$-\frac{\partial PL(X_1)}{\partial w_{i,j}} \times \Delta w_{i,j}$
(Approximate value)
Gradient is $\frac{\partial PL(X_1)}{\partial w_{i,j}}$
$w_{i,j}$

Fig. 2. Decrease in $PL(X_1)$ and its approximate value

It can be said that the value of the decrease in $PL(X_1)$ functions as an index indicating ”How close the output value is to the expected output value.” However, after training has progressed to a certain extent and the output value sufficiently approaches the expected output value, it is considered that not only ”How close the output value is to the expected output value” but also ”How far it has deviated from an erroneous output value” will influence accuracy. Accordingly, as for the proposed method, the ”dimension with the maximum value except for the dimension indicated by the expected output value” among the elements of $y(x^m)$ (which is the output value of N_0) is focused on. Therefore, the output value indicating that dimension is defined as negative supervisor. Since the loss $NL(X_1)$ is calculated based on the negative supervisor, the decrease in the loss $NL(X_1)$ represents ”how close the output value is to the erroneous output value.” Hence, $NL(X_1)$ is considered to be related to the decrease in accuracy. Since $NI(X_1, \Delta W)$ corresponds to the approximate value of the decrease in $NL(X_1)$, it can therefore be assumed that $NI(X_1, \Delta W)$ also serves as an index for evaluating the change in accuracy. Based on the above considerations, the following $INF(X_1, \Delta W)$ can be used as an index for evaluating change in accuracy with respect to X_1.

Definition 6:

$$INF(X_1, \Delta W) = PI(X_1, \Delta W) - NI(X_1, \Delta W)$$

Among the formulas in Definitions 2 to 6 for determining $INF(X_1, \Delta W)$, the formulas in Definitions 2 to 4 are calculated after the end of the initial training and before the start of additional training 1. Although the formulas in Definitions 5 and 6 are calculated after additional training 1, since the amount of calculation depends on the number of elements of W_0 (not the number of input values contained in X_1), even if the size of X_1 is enormous, change in accuracy can be evaluated in a short time. In the case of a multilayer perceptron, since the number of elements of W_0 is g $O(I^2)$ for number of neurons I constituting N_0, the calc order of the proposed method after additional traini given as $O(I^2)$.

IV. Experiment

The results of experimentally applying the proposed to the MNIST dataset [3] (which is a handwritten containing the digits 0 to 9) are presented in the fol In the following sections, arguments concerning IN are omitted as long as there is no misunderstanding .

A. Setup

In this experiment, it is assumed that 50 additional tr will be conducted after the initial training. It is also a that 46,666 image data, which equals two-thirds of the 70,000 image data provided as the MNIST dataset, are retained as initial dataset. It is supposed that the remaining 23,334 data are acquired equally between additional trainings. Namely, the number of data in each additional dataset is given by $23,334/50 \fallingdotseq 466$. Moreover, the ratio of the training dataset to the test dataset in each dataset (initial, additional 1, and so on) is set to 6:1 (as in the MNIST dataset). The DNN used is a multilayer perceptron with two hidden layers (composed of 1,000 neurons each) in addition to the input and output layers.

We evaluate the influence of additional training for each classification class in this experiment. Accordingly, in additional training $s(1 \leq s \leq 50)$, a set of input values with the same classification class $k(0 \leq k \leq 9)$ is created from $\hat{X}_s$ as X_s^k. The proposed method is then applied for each X_s^k and $INF(X_s^k, \Delta W)$ was calculated. Moreover, to evaluate the effectiveness of INF, the difference value of actual accuracy in regard to X_s^k between before and after the additional training is measured. Furthermore, the following times were calculated: (i) the time taken to evaluate the influence by applying the proposed method and (ii) the time taken to evaluate the influence by executing the DNN with all the input values included in $\hat{X}_s$. Note that in the case of the proposed method, the formulas to be calculated after the additional training are those in Definitions 5 and 6. In the initial training and all subsequent additional trainings, cross entropy is used as loss function L. The experiment was performed on a Windows 10® PC equipped with two Intel® Core™ i7-8700 3.2-GHz processor with 6 cores, 16-GB memory.

B. Results

Time taken to apply the proposed method and time taken to run the DNN with all the input values contained in $\hat{X}_s$ are plotted in Fig. 3.

Fig. 3 indicates that calculation time of test execution increases as number of additional training since the number of input values in $\hat{X}_s$ increases. On the contrary, in the case of the proposed method, it is confirmed that the evaluation can be performed in almost the same time regardless of the increase in number of data. It is evident from the fact that the calculation order of the formulas in Definitions 5 and 6 is given as $O(I^2)$ as mentioned in Section III-B. The amount of calculation corresponding to the formulas in Definitions 2 to 4 increases in accordance with number of data; even so, those calculations are carried out before additional training, so their results are not included in the time for evaluating the results of additional training.

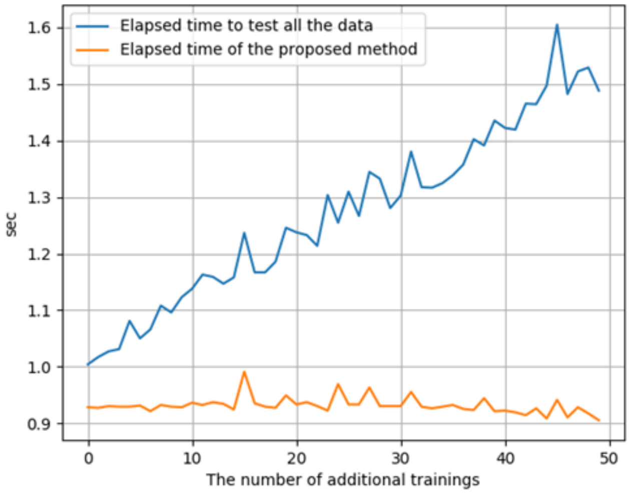

Fig. 3. Calculation times

Values of INF obtained as a result of the experiment (on the left y-axis) and the difference value of actual accuracy (right) in regard to X_s^k between before and after the additional training are plotted in Fig. 4.

It is clear from Fig. 4 that the INF value and the difference value of actual accuracy have similar waveforms. In other words, it can be said that when the INF value increases or decreases as compared with its previous value, it is highly likely that the difference value of accuracy also increases or decreases as compared with its previous time. It can be supposed from this result that it is possible to utilize INF as an index for evaluating the change in accuracy. For example, for the graphs with $k = 0$, the value of the second INF acquired in the additional training is smaller than that of the first additional training. In that case, it can be predicted that the difference value of the accuracy of the second additional training becomes smaller than that of the first additional training. (However, whether it increases by a smaller value than the first additional training or it turns into a decrease cannot be predicted.) For all classification classes, relative change (increase or decrease) of INF was compared with relative change (increase or decrease) of difference value of accuracy, and they agreed with a probability of about 89.2%.

V. Evaluation and Related Work

According to the above-presented experimental results, the feasibility of the proposed method is confirmed. Since it is confirmed that the proposed method does not depend on the number of retained input values, the more the number of retained input values is, the more useful the proposed method

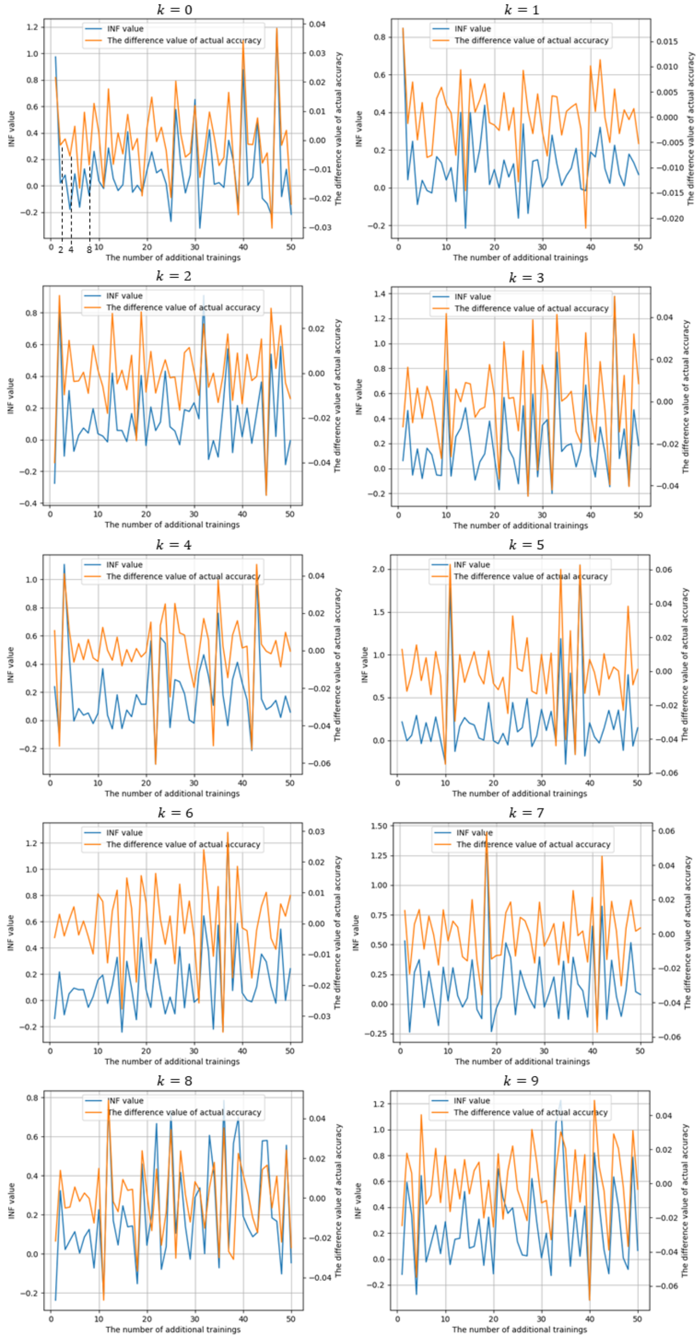

Fig. 4. INF value for each classification class and difference value of actual accuracy between before and after the additional training

is to evaluate the results of additional training than testing all the retained input values.

As for the proposed method, by examining whether INF increases or decreases compared with its previous value, it is possible to evaluate whether the difference value of accuracy also increases or decreases compared with its previous value. Moreover, because INF value and the difference value of actual accuracy have similar waveforms, how much accuracy changes after additional training can be roughly predicted from the waveform of INF. For example, if INF increases more than it previously increased, the difference value of accuracy will also increase more than it previously increased. However, the result of the evaluation by the proposed method is not always accurate. Also, with the proposed method, it is not possible to evaluate the precise value of accuracy after additional training.

In view of these advantages and disadvantages, it is thought that the proposed method will be useful in a system that repeats additional training of DNNs to find a better DNN during operation. In the case of such a system, it should be promptly decided whether the system adopts the DNN additionally trained, tries another additional training, or performs rollback to DNNs before the additional training. Examples of such systems include rescue robots and stock trading systems. These systems tend to prefer adapting to changes in the environment as soon as possible by repeating additional training not to lose opportunities to save people or gain profits. Otherwise, even if the precise value of accuracy is evaluated by running on all the data, using the proposed method in advance is advantageous in that ”triage” becomes possible. For example, it is possible to execute tests with input values from classification classes that are highly likely to degrade accuracy.

Because the number of layers, neurons, and parameters are generalized in the definition of DNNs and types of activation functions are not specified, the proposed method can be applied to any neural networks of multilayer perceptron topology. For the same reason, it is thought that the proposed method gives similar results when applied to a convolutional neural network.

In the graphs for $k = 0$ shown in Fig. 4, as for the 4th and 8th additional trainings, accuracy drops due to the additional training. In that case, it is considered that PL increases, so INF representing the decrease in PL should be a negative value. However, as shown in Fig. 4, INF may have a positive value, the reason for which is explained below. As explained in Section III-B, PI is an approximate value of decrease in PL. First, it is supposed that $\Delta w_{i,j}$ is added to $w_{i,j}$ (which is an element of parameter W) and that the decrement value of PL in that case is represented by ΔPL. When $\Delta w_{i,j} > 0$, $PI > 0$ and $\Delta PL > 0$; however, due to the property of the cross entropy function used as L, $\Delta PL < PI$ holds. On the contrary, if $\Delta w_{i,j} < 0$, although $PI < 0$ and $\Delta PL < 0$, for the same reason, $\Delta PL < PI$. That is, regardless of the updating direction of $\Delta w_{i,j}$, PI becomes larger than ΔPL. For the above reasons, the value of PI tends to be greater than the actual value of the decrease in PL. Similar errors in NI also occur. However, the additional training is conducted so that PL decreases, so it is conceivable that decrease in PL tends to be larger than that in NL in many cases. In that case, the error caused by PI is larger than that caused by NI. As a result, the value of INF tends to be larger than the actual value of the decrease in PL.

To the author’s knowledge, no research results on a method for evaluating additional training results have been published. However, in a similar manner to our research presented in this paper, Kirkpatrick et al. [4] have focused on decrease in accuracy in multitask learning [5] [6]. For example, when training for task 2 is carried out after training for task 1, it is a problem that the performance of the previously trained task (task 1) is catastrophically reduced. In response to that problem, they proposed a method of identifying the parameters (weights and biases) important in regard to task 1, and training task 2 in a manner that change those parameters (important in regard to task 1) as little as possible. Accordingly, it is possible that their method can be applied to additional training by associating the dataset of task 1 with dataset $\hat{X}_s$ and making the dataset of task 2 correspond to additional dataset s. However, even by applying this method, it is difficult to completely eliminate the influence of additional training on the accuracy for the initial dataset. Therefore, even when their method is applied, the proposed method is still useful.

VI. Conclusion

Regarding the problem that the accuracy of the initial dataset is decreased by the additional training of DNNs, a method for quickly evaluating the influence of additional training is proposed. The computational complexity of the proposed method depends on the number of parameters of DNNs (such as weight and bias), namely, not on the amount of data in the initial dataset. Therefore, even if the amount of data included in the initial dataset is enormous, applying the proposed method makes it possible to evaluate the result of additional training quickly. Moreover, it is clear from the results of an experiment with the MNIST dataset that the method is useful in the case of a system that requires quickness—rather than accuracy—of evaluation, such as when additional training of DNNs is performed during operation of the system and further additional training is performed according to the evaluation results. As for future work, the proposed method will be evaluated using datasets other than MNIST. Moreover, improving the means of creating INF will help in the search for more accurate evaluation.

References

[1] D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan: The Missing Piece in Complex Analytics: Low Latency, Scalable Model Management and Serving with Velox, Conference on Innovative Data Systems Research (2015).

[2] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang: Error-driven incremental learning in deep convolutional neural network for large-scale image classification, In Proceedings of the 22nd ACM international conference on Multimedia, pp.177-186 (2014).

[3] Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[4] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell: Overcoming catastrophic forgetting in neural networks, Proceedings of National Academy of Sciences (PNAS) (2017).

[5] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell: Policy distillation, arXiv preprint arXiv:1511.06295 (2015).

[6] E. Parisotto, J. L. Ba, and R. Salakhutdinov: Actor-mimic: Deep multitask and transfer reinforcement learning, arXiv preprint arXiv:1511.06342 (2015).