# Characterizing Adaptive Optimizer in CNN by Reverse Mode Differentiation from Full-Scratch

**Ruo Ando, Yoshihisa Fukuhara, Yoshiyasu Takefuji**

***Abstract**: Recently, datasets have been discovered for which adaptive optimizers are not more than adequate. No evaluation criteria have been established for optimization as to which algorithm is appropriate. In this paper, we propose a characterization method by implementing backward automatic differentiation and characterizes the optimizer by tracking the gradient and the value of the signal flowing to the output layer at each epoch. The proposed method was applied to a CNN (Convolutional Neural Network) recognizing CIFAR-10, and experiments were conducted comparing and Adam (adaptive moment estimation) and SGD (stochastic gradient descent). The experiments revealed that for batch sizes of 50, 100, 150, and 200, SGD and Adam significantly differ in the characteristics of the time series of signals sent to the output layer. This shows that the ADAM optimizer can be clearly characterized from the input signal series for each batch size.*

***Keywords**: Characterization of Optimizers, Adaptive Optimizer, Reverse Mode Differentiation, CNN*

## I. INTRODUCTION

Researchers were missing an effective way to train deep neural networks until the late 2000s. At that time, neural networks working in a shallow with only at most two layers of representations. Main problem of neural networks is called the gradient vanishing problem. That is, the feedback signal for training neural networks would weaken and fade away with the increase of the number of layers.

Several methods have been proposed for this problem as algorithmic improvements.

- More sophisticated activation functions for neural layers [12].
- Better optimizers such as RMS Prop and Adam

We can handle deep neural networks with ten or more layers by adopting these methods. Furthermore, batch normalization method [15] in 2014 and residual networks [5] in 2015 were added to the set of deep learning technologies for helping gradient propagation. Nowadays, we can cope with scratch models with thousands of layers.

**Manuscript received on 30 June 2023 | Revised Manuscript received on 15 June 2023 | Manuscript Accepted on 15 June 2023 | Manuscript published on 30 June 2023.**

*Correspondence Author(s)
**Ruo Ando***, National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan. Email: ruo@nii.ac.jp, ORCID ID: 0000-0002-9279-8333
**Yoshihisa Fukuhara**, Musashino University, Department of Data Science, 3-3-3 Ariake, Koto-Ku, Tokyo, Japan. ORCID ID: 0000-0003-3458-7463
**Yoshiyasu Takefuji**, Musashino University, Department of Data Science, 3-3-3 Ariake, Koto-Ku, Tokyo, Japan. ORCID ID: 0000-0002-1826-742X

However, in [1], Wilson et al. found that there are data sets for which optimal adaptive methods do not provide general-purpose performance. Therefore, there is a need for further exploration and improvement of adaptive optimization methods. In this article, we propose a characterization method of adaptive optimizers by tracing the final input values to the output layer. We have implemented automated reverse mode differentiation by C++ from full scratch.

## II. RELATED WORK

The back-propagation algorithm, which is originally proposed by Rumelhart et al. [6] enables chain-rule for backward flow through the network for calculating the gradient. Caffe [16] adopts the approach of symbol-to-number differentiation. Theano [9], Pytorch[2] and Tensorflow [3] adopts the approach with a computational graph where additional nodes represent a symbolic description of the expected derivatives. Random filters have a good performance in convolutional networks.

CNNs which Le Cun first proposed [18], adopts a grid-like topology, which makes CNN a specialized kind of neural network. In [11], some theoretical guidance concerning coping with the pooling layer. Another essential feature of CNNs is the pooling layer. For instance, a clustering algorithm for each image's various pooling regions is presented in [10].

Boris Polyak proposes momentum optimization using terminal velocity [7]. Further, Nesterov Momentum Optimization (NAG) is presented by Yurii Nesterov in 1983. NAG applies a cost function by gradient. In NAG, the cost function is calculated before the momentum direction is determined. RMS Prop [8] improves Ada Grad [13] by calculating the gradients from the latest iterations. Adam [17] is a kind of hybrid version of both RMS Prop and optimizing momentum. In Adam, the average of the past squared gradient and an average of previously calculated gradients which is exponentially decaying are held during the learning process. Hy Adam C [19] is improved version of Adam in CNN. Recursion algorithm of batch normalization is discussed in [4]. Detailed implementation of deep neural network is presented in [14].

## III. OPTIMIZERS: SGD AND ADM

Adam [17], which stands for Adaptive Moments, is one of the optimization algorithms for holding track of the average exponential decay of past gradients. Like RMS Prop, it holds an exponentially decaying

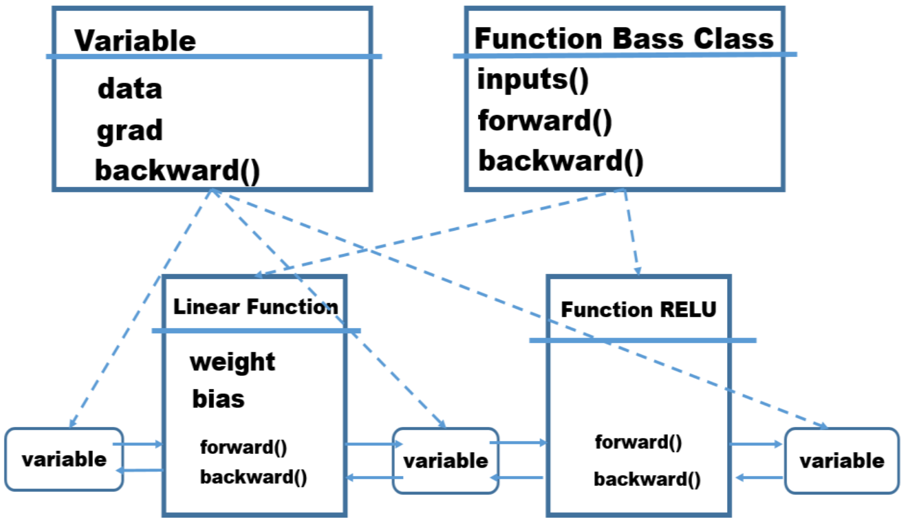


**Fig. 1. Deviation of Linear fucion and activate fuction (RELU) in CNN**

average of past squared gradients. Adam copes with the first-order moment of the gradient as exponential weighing.

Straightforwardly, Adam uses momentum with the rescaled gradients. Applying momentum with rescaling is not based on clearly-defined theoretical motivation.

Also, Adam considers the bias corrections to the estimations. The estimations are figured out as the first-order and second-order moments.

Concerning momentum, RMS Prop figures out an estimation about the second-order moment. However, unlike Adam, the second-order moment of RMS Prop is estimated as high bias.

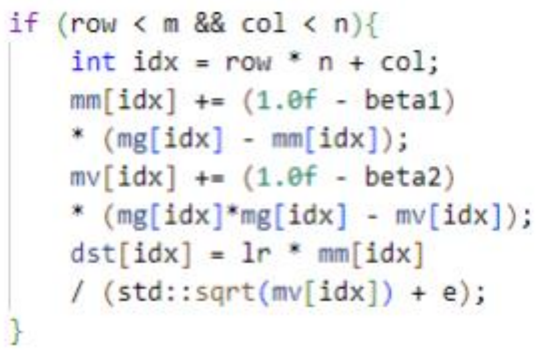
Adam is often fairly robust to the perturbation of hyper parameters, as the learning rate of Adam often require to be modified from the suggested result.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) * g_t \qquad (1)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) * g_t^2 \qquad (2)$$

Finally, the weight update is as follows:

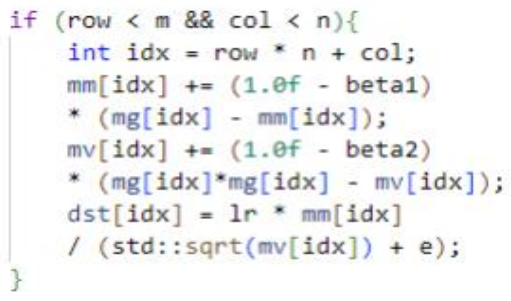
$$\Delta W_t = \Delta W_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \qquad (3)$$



```
if (row < m && col < n){
    int idx = row * n + col;
    mm[idx] += (1.0f - beta1)
    * (mg[idx] - mm[idx]);
    mv[idx] += (1.0f - beta2)
    * (mg[idx]*mg[idx] - mv[idx]);
    dst[idx] = lr * mm[idx]
    / (std::sqrt(mv[idx]) + e);
}
```

**Fig. 2. Adam kernel: C++ code of equation (1) (2)**

The figure shows the code for updating weights by Adam. Here, mm is the first-order moment of gradient (mean). And mv is the second-order moment which is variance. The variables of beta1 and beta2 are hyper parameters.



```
if (row < m && col < n){
    int idx = row * n + col;
    mm[idx] += (1.0f - beta1)
    * (mg[idx] - mm[idx]);
    mv[idx] += (1.0f - beta2)
    * (mg[idx]*mg[idx] - mv[idx]);
    dst[idx] = lr * mm[idx]
    / (std::sqrt(mv[idx]) + e);
}
```

**Fig. 3. Adam kernel: C++ code of equation (3)**

## IV. IMPLEMENTATION

### A. Linear function

The reverse mode differential of the loss function is denoted as follows:

$$Z^l = f^l(U^{(l)}) \qquad (4)$$

$$U^{(l+1)} = W^{(l)} * Z^l + b^l \qquad (5)$$

Here $f^l$ is the activation function, for example, the sigmoid and RELU (Rectified Linear Unit). $U^l$ is the input of l-th layer. Equation (4)(5) shown in Figure 2 represents the linear activation with weight $w^{(l)}$ and bias $b^l$. We apply stochastic gradient descent with the incremental change of $w^{(l)}$ where T is the mini-batch size.

Figure 3 depicts the code for calculating the gradient according to the minimization in the equation of (6)(7). By adopting the chain rule, we can update the gradient $\frac{\partial E_p}{\partial w^{(l)}}$from the output layer to the input layer I n back-propagation. At lines 16 to 19, $\Delta^l$, which is represented the left side of the equation (6)(7), is calculated.

$$\Delta^{(l)} = (W^{(l+1)T} * \Delta^{l+1}) \odot f^{(l)'}(U^{(l)}) \qquad (6)$$

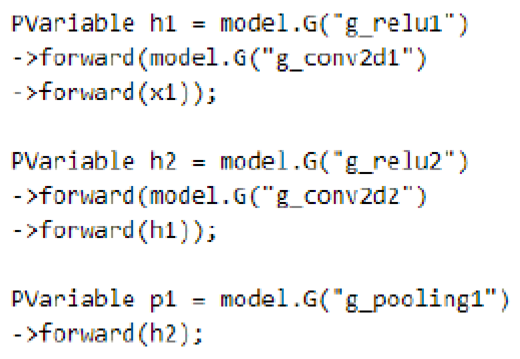
$$\frac{\partial E_p}{\partial W(l)} = \Delta^l * Z^{(l-1)T} \qquad (7)$$

## B. CNN

The behavior of the human visual cortex inspired Convolutional Neural Networks (CNNs) designed to recognize objects. Recently CNNs have had overwhelming performance in the task of image classification. On the other hand, it is often pointed out that deep learning algorithms, including CNNs, are block boxes. The process of the algorithms is almost impossible to be translated into a human-readable representation.

This is also the case with convolutional neural networks. The weight matrix calculated by convolutional neural networks is highly amenable to image classification tasks because CNNs have good representations of visual concepts. Currently, the prevailing view is that there are no clear criteria for selecting an optimizer, based on the assumption that deep learning is a ``black box".

As we know, optimization is a critical step in deep learning algorithms. Unfortunately, we have yet to determine which algorithm one should choose so far. However, with a lot of open-source of CNNs available on the Internet website such as GitHub, we can modify those codes to track every variable in every phase of the process of learning. By leveraging the open source, we present a novel method of full gradient tracing of adaptive moment estimation optimizer (Adam and Adagrad) in CNNs.



```
PVariable h1 = model.G("g_relu1")
->forward(model.G("g_conv2d1")
->forward(x1));

PVariable h2 = model.G("g_relu2")
->forward(model.G("g_conv2d2")
->forward(h1));

PVariable p1 = model.G("g_pooling1")
->forward(h2);
```

**Fig. 4. C++ code of RELU**

## C. *RELU*

Figure 4 depicts C++ code of RELU (Reflective Linear Unit). In this implementation, the variable of weight and bias in each function is freed and erased after the function is called. Therefore we should save the result of the variable and weight in graph class. In Figure 4, the program derives Function Linear->forward and Function RELU->forward with the value of weight and bias. The framework in Figure 4 is defined as follows:

$$f(x) = \psi * (\sum_{i=0}^{n} w_i * x_i + b) \qquad (8)$$

In equation (8), $ \psi $ is an activation function, for example, Tanh and RELU. FunctionLinear is implemented as a C++ class in the lower-left side of Figure 4. FunctionLinear is noted as of summasion in equation (8). For example, FunctionLinear outputs f(x), which is equal to the right hand of equation (8), and is passed to the Graph class of RELU. RELU is shown in code of Figure 4.

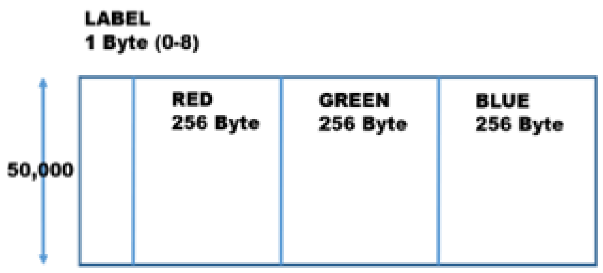


**Fig. 5. CIFAR-10 binary format**

## V. EXPERIMENTAL RESULTS

In the experiment, we ran our program on a workstation equipped with a CPU of Intel(R) Xeon(R) E5-2620 v4 (2.10GHz) and 252G RAM.

## A. CIFAR-10

CIFAR-10 is a standard dataset for benchmarking of image classification in computer vision and machine learning. CIFAR images are more eleborated than those in MMIST, which is another popular benchmark dataset. CIFAR is color with three channels for representing dynamic variation, while MNIST images are all grayscale and centered objects. Figure 5 shows CIFAR-10 binary formats. The CIFAR-10 dataset has 32x32 color images. In addition, color images are classified into 10 classes, and CIFAR-10 has 6000 images. In detail, CIFAR-10 has 5000 images for the train and 1000 images for the test. CIFAR-10 adopts two kinds of image format: Python pickle style and original binary format. We chose the original binary format. Figure 5 depicts the binary format of CIFAR-10. For example, The first 1 byte in Figure 5 is several kinds of labels: airplane, automobile, bird, cat, deer, frog, horse, ship, and truck. The following 1024 bytes have a 32*32 pixel value of red color.

## B. Numerical outputs

Figure 6 plots the change in the value of output values to the final layer. The epoch size is **, and the batch size is varied between 50, 100, 150, and 200. It is found that there is a significant difference in the characteristics of the transition of output value values to the final layer between SGD and Adam, depending on the batch size. In both SGD and Adam, there are batch sizes where the lower bounds of the data are stable (2, 4, 6, 7). For side, the lower bounds are stable at 1 and 4, and for Adam, at 6 and 7. In the case of batch 50, SGD is stable, but ADAM is not. This may be due to the need for more information on past gradients in the case of ADAM. In contrast, SDG is not stable when the batch size is 200. This may be due to the batch size needing to be bigger. On the other hand, batch sizes 3 and 5 are also relatively unstable, but the cause is unclear. In any case, the experiment revealed that ADAM is not stable at batch sites 50 and 200.

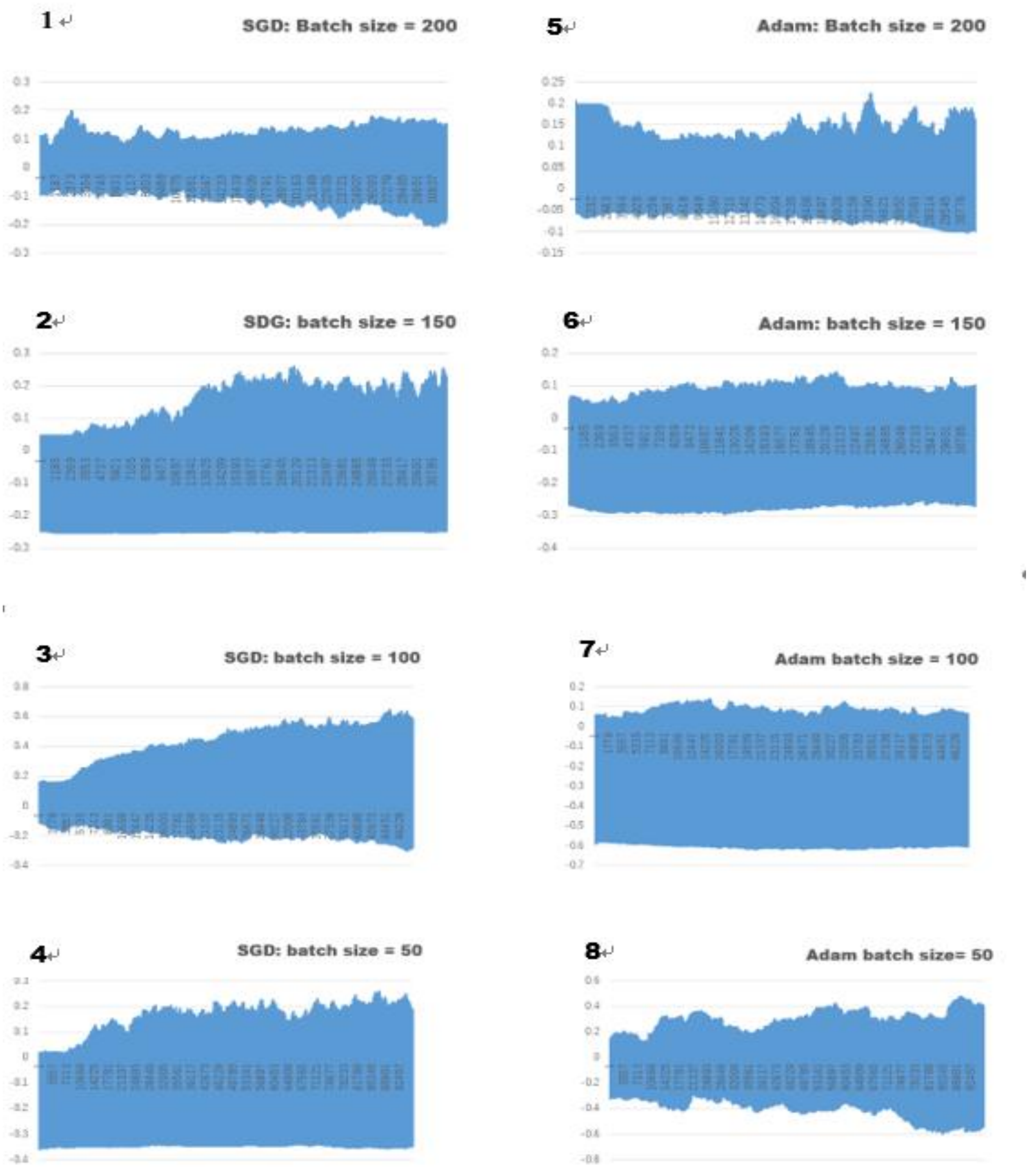


**Fig. 6. Experimental result of ADAM and SGD**

## VI. DISCUSSION

As mentioned in the introduction, cases have been reported where adaptive optimization methods have failed to achieve satisfactory generalization performance on some data sets. This can be attributed to several optimization challenges in neural networks.

- Many believe that most optimization difficulties in neural networks are due to extrema. In high-dimensional spaces, it is complicated to be sure that minima matter. This is because it requires a detailed scrutiny of the gradient norm.
- It is known that for many high-dimensional nonconvex functions, there are many saddle points (points where the gradient is zero) and very few minima; Dauphin points out that the loss function of a high-dimensional neural network contains many computationally awkward saddle points.

- In some cases, the problem may be that almost all optimization methods in existence use temporary partial derivatives. Second-order partial derivative optimization methods have also been proposed but are very difficult to implement regarding memory constraints.

## VII. CONCLUSION

There is still much that is not known about the optimization of current neural networks. Even concerning adaptive optimizers, cases have been pointed out in which they do not provide adequate generalization performance. In this paper, we have implemented the backward automatic differentiation of neural networks from scratch and characterized adaptive optimizers. The proposed method tracked the input signal transition to the output layer and observed the optimizer's behavior by varying the batch size. The experimental results show that the behavior of regular SGD and ADAM, the adaptive optimizer, is significantly different in the cases of batch sizes of 50, 100, 150, and 200. Specifically, the time series of the input signal to the output signal was found to be unstable for batch sizes of 100 and 200 for SGD and 50 and 150 for ADAM. The proposed method is independent of the optimization algorithm and can be applied to optimizers such as momentum-based NAGs

## DECLARATION

| | |
|---|---|
| Funding/ Grants/ Financial Support | No, I did not receive. |
| Conflicts of Interest/ Competing Interests | No conflicts of interest to the best of our knowledge. |
| Ethical Approval and Consent to Participate | No, the article does not require ethical approval and consent to participate with evidence. |
| Availability of Data and Material/ Data Access Statement | Not relevant |
| Authors Contributions | All authors having equal contribution for this article. |

## REFERENCES

1. Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, Benjamin Recht: The Marginal Value of Adaptive Gradient Methods in Machine Learning. CoRR abs/1705.08292 (2017)
2. Pytorch. https://github.com/pytorch/pytorch
3. Martin Abadi et al.: ``TensorFlow: A System for Large-Scale Machine Learning", OSDI 2016: 265-283
4. Ando, R. and Takefuji", Y, "A constrained recursion algorithm for batch normalization of tree turctured lstm", https://arxiv.org/abs/2008.09409
5. Andreas Veit, Michael J. Wilber, Serge J. Belongie: Residual Networks Behave Like Ensembles of Relatively Shallow Networks. NIPS 2016: 550-558
6. David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, Learning representations by back-propagating errors, Nature volume 323, pages533-536 (1986) [CrossRef]
7. B.T. Polyak, Some methods of speeding up the convergence of iteration methods, USSR Computational Mathematics and Mathematical Physics Volume 4, Issue 5, 1964, Pages 1-17 [CrossRef]
8. Geoffrey Hinton Neural Networks for machine learning online course. \\ https://www.coursera.org/learn/neural-networks/home/welcome
9. Frdric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, Yoshua Bengio: Theano: new features and speed improvements. CoRR abs/1211.5590 (2012)
10. Y-Lan Boureau, Nicolas Le Roux, Francis R. Bach, Jean Ponce, Yann LeCun: Ask the locals: Multi-way local pooling for image recognition. ICCV 2011: 2651-2658 [CrossRef]
11. Y-Lan Boureau, Jean Ponce, Yann LeCun: A Theoretical Analysis of Feature Pooling in Visual Recognition. ICML 2010: 111-118
12. Brownlee, J, "a gentle introduction to the rectified linear unit (relu)", Machine Learning Mastery, 2021
13. Duchi, J., Hazan, E.Singer et al. "adaptive subgradient methods for online learning and stochastic optimization", Journal of Machine Learning Research, 2121-2159
14. Frosst, N. and Hinton, G. , ``distilling a neural network into a soft decision tree", https://arxiv.org/abs/1711.09784
15. Ioffe, S. and Szegedy, C, ``batch normalization: Accelerating deep network training by reducing internal covariate shift","arXiv:1502.03167. 2015
16. Jia, Shelhamer, Donahue, Karayev, Long, Girshick, Guadarrama. "caffe: Convolutional architecture for fast feature embedding", CoRR abs/1408.5093
17. Kingma, D.~P. ,Ba, "adam: A method for stochastic optimization", ICLR (Poster), 2015.
18. Yann LeCun, Lawrence D. Jackel, Bernhard E. Boser, John S. Denker, Hans Peter Graf, Isabelle Guyon, Don Henderson, Richard E. Howard, Wayne E. Hubbard: Handwritten digit recognition: applications of neural network chips and automatic learning. IEEE Commun. Mag. 27(11): 41-46 (1989) [CrossRef]
19. Kyung Soo Kim, Yong Suk Choi: HyAdamC: A New Adam-Based Hybrid Optimization Algorithm for Convolution Neural Networks. Sensors 21(12): 4054 (2021) [CrossRef]

## AUTHORS PROFILE

**Ruo Ando** received Ph.D. from Keio University in 2006. He is now associate professor by special appointment of National Institute of Informatics since 2016. Before joining NII, he worked as senior researcher of National Institute of Information and Communications Technology since 2006. His research interests focus on network security, information security and big data mining technologies. He received Outstanding Leadership Award in the 8th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC-09) at China in 2009. He is the member of Trusted Computing Group JRF (Japan Regional Forum) in 2008-2015. He worked in project "Next Generation Security Info-Security R&D" METI (FY2008-10). He was engaged in project "Unknown malware detection using incremental malware detection" MEXT FY(2012-2015).

**Yoshihisa Fukuhara** is is an associate professor of Musashino University since 2019. He has received Ph.D from Keio University in 2003. His research area in Graduate school of Keio University was a complexity and chaos in artificial neural networks. Before joining Musashino University, He has engaged in the activity of Shiseido Child Foundation in Japan. He has received Keio University Principle Prize in 1999. He is now working in the department of Data Science of Musashino University. His research interests are neural networks, machine learning and artificial intelligence. He has received Student Encouragement Awards from IPSJ (Information Processing Society of Japan) in 2022 and 2023.

**Yoshiyasu Takefuji** is a professor of Musashino University and was a tenured professor on faculty of environmental information at Keio University from April 1992 to March 2021, and was on tenured faculty of Electrical Engineering at Case Western Reserve University since 1988. Before joining Case, he taught at the University of South Florida for two years and the University of South Carolina for three years. He received his BS (1978), MS (1980), and Ph.D. (1983) from Electrical Engineering from Keio University. He received the National Science Foundation/Research Initiation Award in 1989 and received the distinct service award from IEEE Trans. on Neural Networks in 1992 and has been an NSF advisory panelist.