International Journal of Advanced Smart Convergence Vol.6 No.4 67-72 (2017)
http://dx.doi.org/10.7236/IJASC.2017.6.4.10

IJASC 17-4-10

Improvement of the Convergence Rate of Deep Learning by Using Scaling Method

Jiacang Ho*, Dae-Ki Kang†

*Department of Ubiquitous IT, Graduate School, Dongseo University
†Department of Computer Engineering, Dongseo University
*ho_jiacang@hotmail.com, †dkkang@dongseo.ac.kr

Abstract

Deep learning neural network becomes very popular nowadays due to the reason that it can learn a very complex dataset such as the image dataset. Although deep learning neural network can produce high accuracy on the image dataset, it needs a lot of time to reach the convergence stage. To solve the issue, we have proposed a scaling method to improve the neural network to achieve the convergence stage in a shorter time than the original method. From the result, we can observe that our algorithm has higher performance than the other previous work.

Keywords: *Deep learning, neural network, convergence rate, scaling method.*

1. Introduction

Due to the high speed of the internet that we currently have today, there are many precious data we can keep into a storage and then we can use them for some purposes in the future. These data that have stored in the storage we called them as the dataset. Due to the reason that keeps storing the new data can make the dataset becomes very big. For example, image dataset, speech dataset, etc., are big in term of the size and attributes. We know that it is difficult to execute a big dataset with a simple machine learning algorithm such as linear regression, Euclidean distance, etc. Therefore, deep learning [1] has been proposed and it is one of the best methods that can operate with a huge dataset effectively and efficiently.

Deep learning, also known as deep structured learning, is part of a broader family of machine learning methods based on learning data representations. Deep learning architectures such as recurrent neural networks [2], deep neural networks [3], and deep belief networks [4] have been applied to fields including computer vision, bioinformatics, speech recognition, natural language processing, audio recognition, etc., where they have generated results comparable to and in some cases greater to human experts.

Due to the complexity of the dataset (refers to the multidimensional properties), to the best of our knowledge, deep learning is the most appropriate method among the others to perform the big dataset's experiment. With the integration of the deep learning technique and a machine, the machine can learn some behaviours effectively and accurately from the given data. However, there is an open problem in the deep learning technique that remains unsolved. The issue of the deep learning technique is that it needs to consume a lot of time to train a dataset if the neural network structure consists of many layers. Figure 1

Manuscript Received: Nov. 10, 2017 / Revised: Nov. 23, 2017 / Accepted: Dec. 5, 2017
Corresponding Author: dkkang@dongseo.ac.kr
Tel: +82-51-320-1724, Fax: +82-51-327-8955
Department of Computer Engineering, Dongseo University, Korea

explains the simple neural network (left) and deep learning neural network (right) structures. As we can see from Figure 1, the simple neural network has only 1 hidden layer but deep learning neural network has 4 hidden layers. In the reality, the structure of the deep learning neural network can be more which will reach to 20 or more hidden layers. The more the number of hidden layer, the slower the training will be completed.

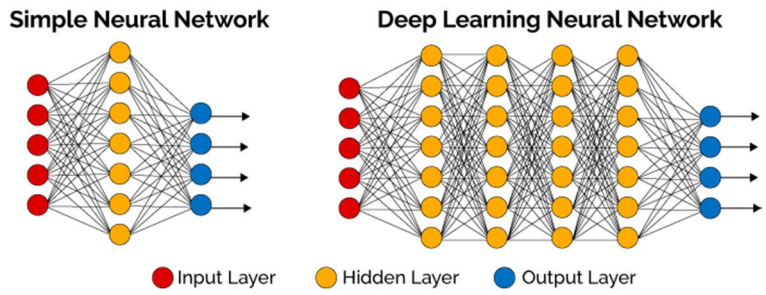


Figure 1. Simple neural network and deep learning neural network structure

Hence, to solve the issue, we study a way to shorten the convergence rate of the neural network. There are several researchers have studied on this work and they have proposed a few methods such as Momentum [5], Nesterov Momentum [6], Adagrad [7], RMSprop [8], Adam [9] etc., which these methods have shown the significant improvement in the convergence rate.

From the previous work, commonly, in order to reduce the convergence rate of the neural network, the researchers study more on the update rule in the back-propagation process. In contrast, we work on the equation in the feed-forward process to scale down the time of the convergence. We propose a scaling method that improves the convergence rate of the neural network which namely scaled neural network (Scaled-NN). Scaled-NN is a simple improvement algorithm and it speeds up the convergence stage of a neural network. We elaborate more of our proposed method in Section 2.

Sections 3 and 4 describe the experimental method and the results of the experiment, respectively. The final section (Section 5) concludes several remarks.

2. Scaled neural network

Scaled-NN leads the fast convergence stage and obtains the same performance in term of the accuracy of the original neural network structure during the training phase. Scaled-NN is a simple modification of the original equation (Eq. 1) and does not consume extra memory, but only has to add a few operations in the computation process. Before we explain the insight of the Scaled-NN, we briefly explain the basic equation of the neural network.

The common equation that we have known in the neural network is shown as follow:

$$\hat{Y} = W^T X + b \tag{1}$$

where $\hat{Y}$ is the predicted output, W is the weight, X is the input and b is the bias. In our Scaled-NN method, we add an extra variable, k, in front of the equation on the left-hand side of Eq. 1 which the final modification equation formed as follow:

$$\hat{Y} = \boldsymbol{k} \cdot (W^T X + b) \tag{2}$$

where $\hat{Y}$ is the predicted output, k is the scaling factor, W is the weight, X is the input and b is the bias. One of the important roles that the k can be used is the regularization process. Regularization is a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting. Noted that we use Eq. 2 only in the last layer of the neural network but before the activation function (usually use softmax and cross entropy loss function in the output layer) layer (shown in Eq. 4). After applying the activation function in Eq. 1 and 2, they can be a new form as shown in Eq. 3 and 4, respectively.

$$\hat{Y} = a(W^T X + b) \tag{3}$$

$$\hat{Y} = a(k \cdot (W^T X + b)) \tag{4}$$

where $a(\cdot)$ is an activation function.

2.1 Step to produce the k

The variable k is not a random scalar value but it has several steps to generate it. We explain the detail of the steps in generating the variable k in the following paragraphs.

During the training phase, first, we generate the usual output, Y_1, as presented in Eq. 1 (shown in Eq. 5). Then, we apply the softmax activation function to the generated output, Y_1 in order to produce a new output, Y_2 (shown in Eq. 6). Next, we compute the cross product between Y_2 and the original label (class), Y. This will return a scalar value according to the correct label. Finally, we sum the cross product value with one value (shown in Eq. 7). Hence, the final value of k will be in between 1 and 2, inclusively. The reason we add one value in Eq. 7 is to prevent the zero value which is possible to be generated from Eq. 6. In other words, the value of the k is either equal to or more than 1 if the predicted value is incorrect, or the value of the k is more than 1 or equal to 2 if the predicted value is correct. Assume that if we allow the k to be zero, then the final predicted output will be zero because of the multiplication in the equation.

$$Y_1 = W^T X + b, -\infty \leq Y_1 \leq \infty \tag{5}$$

$$Y_2 = softmax(Y_1), 0 \leq Y_2 \leq 1 \tag{6}$$

$$k = ({Y_2}^T Y) + 1, 1 \leq k \leq 2 \tag{7}$$

To provide a clear description of the entire proposed method's design, we present the pseudocode for this method in Algorithm 1

Algorithm 1 Pseudocode for Scaled Neural Network (Scaled-NN)

Input: MNIST dataset ($x_1, x_2, \ldots, x_n \in X$)
Initialize all the parameters: learning rate (α), weights (W), bias (b)
  while W, b not converge **do**
    $y \leftarrow WX + b$
    $k \leftarrow softmax(y) \times Y + 1$
    $\hat{Y} \leftarrow softmax\_cross\_entropy(k \cdot y)$
    update the weights and bias by using back-propagation method
  end while
  return W, b
Output: W, b

3. Experimental method

In this paper, we perform only MNIST dataset in our experiment. The MNIST dataset is a big dataset of handwritten digits. It is usually used for training various image processing systems. In this dataset, it consists of 60,000 training images and 10,000 testing images. In term of the programming language, we use TensorFlow which is written in Python programming language. TensorFlow is widely used nowadays due to the reason that it is an open-source software library for dataflow programming across a range of tasks and it is developed by the Google Brain team.

During the training phase, we initialize the batch size to 100. This means that it will run about 600 iterations per epoch. In all experiment, we only execute 50 epochs and then record the accuracy value of the testing data in each epoch. Each accuracy value will be illustrated in Figure 2 in the next section.

In this paper, we test four neural network structures which are logistic regression [10], multi-layer perceptron [11], bi-directional recurrent neural network [12], and convolutional neural network [13]. From the setting of our experiment in term of the number of the layer, there are only two layers in logistic

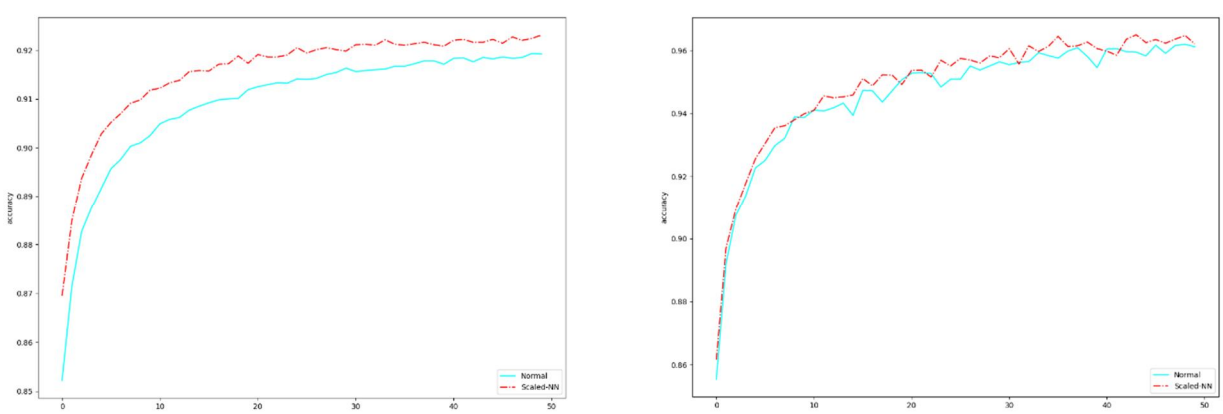
regression which are the input layer and the output layer. For the multi-layer perceptron, we use four layers: one input layer, two hidden layers and one output layer. Noted that we use ReLU activation function in all hidden layers. As the bi-directional recurrent neural network, there are two layers which have the same neural network structure as the logistic regression, but the bi-directional recurrent neural network has its own computation to produce the predicted output at the output layer. In the convolutional neural network, we operate five layers: one input layer, three convolution layers, and one output layer. For the first two convolution layers, we use max pooling method. We apply the dropout [14] feature on the third convolution layer. We ensure the settings are the same all the times between the original neural network structure and our proposed method so that the results are comparable.

4. Experimental results

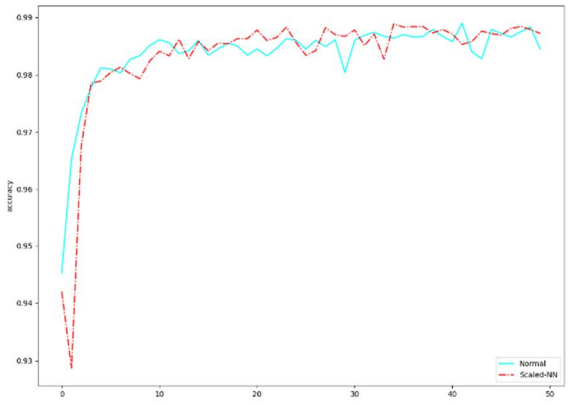
In this paper, we provide the graph of accuracy for four neural network structures in Figure 2. From each graph, red-dash-line refers to our proposed method, Scaled-NN and the cyan-solid-line refers to the original neural network structure. The Figure 2 (a) is the result of the logistic regression. The Figure 2 (b) and (c) are the output for the multi-layer perceptron and bi-directional recurrent neural network respectively. The last figure, Figure 2 (d) is the outcome of the convolutional neural network.

In Figure 2 (a), we observe that our Scaled-NN has significant improvement in term of accuracy. Noted that the y-axis refers to the accuracy and the x-axis stands for the number of the epoch. For the Figure 2 (b), although the result is not as significant as the Figure 2 (a), the area under the curve (AUC) can prove that our proposed method has higher performance than the original one. As the Figure 2 (c), we monitor that the overall performance of our Scaled-NN method is not as good as the previous figures (i.e. Figure 2 (a) and (b)), but we observe that the accuracy at the last epoch (i.e. 50th epoch) has the highest value compared to the original one. We study the Figure 2 (d) has less significant improvement in our Scaled-NN method compared to the Figure 2 (a) but the overall result is very promising in this paper.

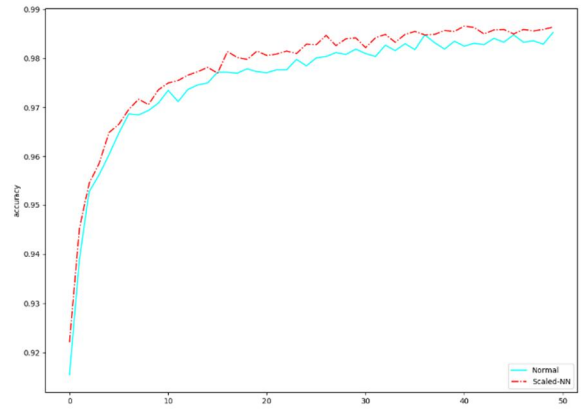
In summary, we analyze that Scaled-NN method has high performance and can be applied to any neural network structure.



(a) Logistic regression **(b) Multi-layer perceptron**



(c) Bi-directional Recurrent Neural Network



(d) Convolutional Neural Network

Figure 2. The graph of accuracy of the neural network. Cyan-solid-line is the original neural network and the red-dash-line is the Scaled-NN.

5. Conclusion

In this study, we propose Scaled-NN to reduce the time taken for the neural network structure and achieve the highest performance at the same time. In our experiment, we show that Scaled-NN method has the highest performance compared to the four original neural network structures which are the logistic regression, multi-layer perceptron, bi-directional recurrent neural network, and convolutional neural network. The result has shown a very promising output in the experiment.

In the future, we would like to use our Scaled-NN method to perform more experiments in other big datasets such as Cifar10, Cifar100, IMAGENET, etc. Besides that, we also believe that we can improve the method to be more efficient and effective during the training phase in the future.

Acknowledgement

This research was supported by 2015 Dongseo University research grants and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (no. NRF-2015R1D1A1A01061328).

References

[1] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning,” *Nature* Vol. 521, pp. 436-444, 2015.

[2] T. Mikolov et al., “Recurrent neural network based language model,” *Interspeech*, Vol. 2, 2010.

[3] G. Hinton, et al., “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal Processing Magazine,* Vol. 29, No. 6, pp. 82-97, 2012.

[4] G. E. Hinton, “Deep belief networks,” Scholarpedia, Vol. 4, No. 5, 2009.

[5] S. Becker and Y. Le Cun, “Improving the convergence of back-propagation learning with second order methods,” *in Proc. of the 1988 connectionist models summer school.* 1988.

[6] Y. Nesterov, “A method of solving a convex programming problem with convergence rate O (1/k2),” *Soviet Mathematics Doklady.* Vol. 27. No. 2. 1983.

[7] J. Duchi, E. Hazan and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research,* pp. 2121-2159, 12 Jul. 2011.

[8] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural networks for machine learning* 4.2, pp. 26-31, 2012.

[9] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv, 1412.6980, 2014.

[10] D. W. Hosmer Jr, S. Lemeshow and R. X. Sturdivant, *Applied logistic regression.* Vol. 398. John Wiley & Sons, 2013.

[11] D. W. Ruck, et al., "The multilayer perceptron as an approximation to a Bayes optimal discriminant function," *IEEE Transactions on Neural Networks*, Vol. 1, No. 4, pp. 296-298, 1990.

[12] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing,* Vol. 45, No. 11, pp. 2673-2681, 1997.

[13] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems,* 2012.

[14] N. Srivastava, et al., "Dropout: a simple way to prevent neural networks from overfitting," *Journal of machine learning research,* Vol. 15, No. 1, pp. 1929-1958, 2014.