Intelligent Information Management, 2018, 10, 99-107
http://www.scirp.org/journal/iim
ISSN Online: 2160-5920
ISSN Print: 2160-5912

An Optimization of Neural Network Hyper-Parameter to Increase Its Performance

Yinxuan Fu

Cranbrook Kingswood Schools, Bloomfield Hills, MI, USA
Email: fustinosej@gmail.com

How to cite this paper: Fu, Y.X. (2018) An Optimization of Neural Network Hyper-Parameter to Increase Its Performance. *Intelligent Information Management*, **10**, 99-107.
https://doi.org/10.4236/iim.2018.104008

Received: October 31, 2017
Accepted: July 8, 2018
Published: July 11, 2018

Copyright © 2018 by author and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

Abstract

With the boost of artificial intelligence, the study of neural network intrigues scientists. Artificial neural network, which was first designed theoretically in 1943 based on understanding of human brains, demonstrated impressing computational and learning capabilities. In this paper, we investigated the neural network's learning capability by using a feed-forward neural network to recognize human's digit hand-writing. Controlled experiments were executed by changing the input values of different parameters, such as learning rates and hidden layer units. After investigating upon the effects of each parameter on the overall learning performance of the neural network, we concluded that, when an intermediate value of one given parameter was implemented, the neural network achieved the highest learning efficiency, and potential problems like over-fitting would be prevented.

Keywords

Learning Efficiency, Neural Network, Intermediate Values

1. Introduction

The human brain has always intrigued scientists. The brain's function is very powerful and efficient [1]. Scientists have devoted themselves for a long time to the deciphering of brain's structure and its power. In 1943, Warren McCulloch and Walter Pitts together produced a theory on how the biological neural network might work [2]. The computational structure became the basis on which future development of artificial neural network (referred to as "neural network" below) was built.

The basic structure of neural network consists of large number of artificial neurons, which execute similar function as biological neurons, but in more abstract forms [3]. Artificial neurons (referred to as "neurons" below) are separated into three or more layers: one input layer, one or multiple hidden layer(s), and one output layer. Information and data enter from the input layer and are transmitted to hidden layer neurons, through which data are analyzed and transformed to the next layer via algorithms. To carry out its function, we must first let the neural network "learn", just as the human brain. We train the neural network with certain information and data, and it will be able to analyze data with similar traits. However, like the human brain, neural network will "behave" differently if input data is different or if different network parameters are applied. Such parameters include learning rate (the rate at which a neural network is adapting to new information and/or data), weights (the measure of influence of one neuron to another), number of hidden layers, etc. [4].

In this paper, we are investigating the effects of these parameters on the neural network. We are using a feed-forward neural network to recognize human's digit handwriting. The dataset is the USPS collection of handwritten digits, which contains images of digits that people wrote [5]. The input is a 16 by 16 image of greyscale pixels, showing an image of a handwritten digit. We don't implement any image pre-processing since it is beyond the scope of this paper; therefore, the input layer contains 256 neurons, one for each pixel. We use only one hidden layer with multiple sigmoid (logistic) neurons. We use 10 soft max neurons for the output layer, and each neuron represents one class among 0 to 9. The structure is shown in **Figure 1**.

2. Optimization

In this section, we applied different learning rates to the neural network, and explored their effects on training performance by comparing the loss on the training data [6]. We tried learning rates of 0.002, 0.01, 0.05, 0.2, 1.0, 5.0, and 20.0, both with momentum (0.1, 0.5, and 0.9 respectively) and without momentum [7]. The result is shown in **Table 1** and depicted in **Figure 2**.

The results show that for each momentum, the training loss is the lowest at either 0.2 or 1.0 learning rate, while the value is larger at very small or very large learning rate. **Figure 3** compares the behavior of the neural network at different

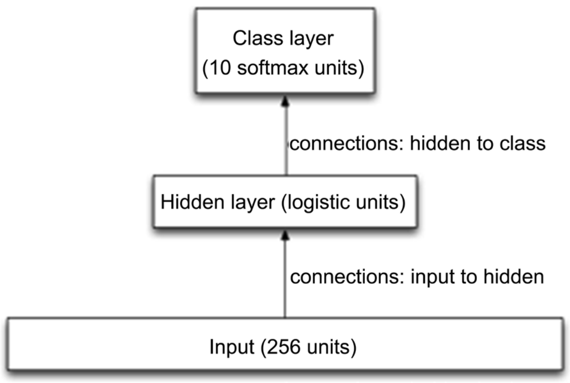
Class layer
(10 softmax units)
connections: hidden to class
Hidden layer (logistic units)
connections: input to hidden
Input (256 units)

Figure 1. Class layer, hidden layer, and input.

Table 1. Affect of momentums in learning rates and training loss.

(a)

Momentum = 0		Momentum = 0.1	
Learning rates	Training loss	Learning rates	Training loss
0.002	2.30428	0.002	2.30422
0.01	2.30212	0.01	2.30183
0.05	2.29297	0.05	2.29170
0.2	2.22897	0.2	2.21025
1.0	1.59884	1.0	1.52411
5.0	2.30132	5.0	2.30185
20.0	2.30259	20.0	2.30259

(b)

Momentum = 0.5		Momentum = 0.9	
Learning rates	Training loss	Learning rates	Training loss
0.002	2.30372	0.002	2.30014
0.01	2.29971	0.01	2.28402
0.05	2.28010	0.05	2.00861
0.2	1.99455	0.2	1.08343
1.0	1.13944	1.0	2..01872
5.0	2.30255	5.0	2.30259
20.0	2.30259	20.0	2.30259

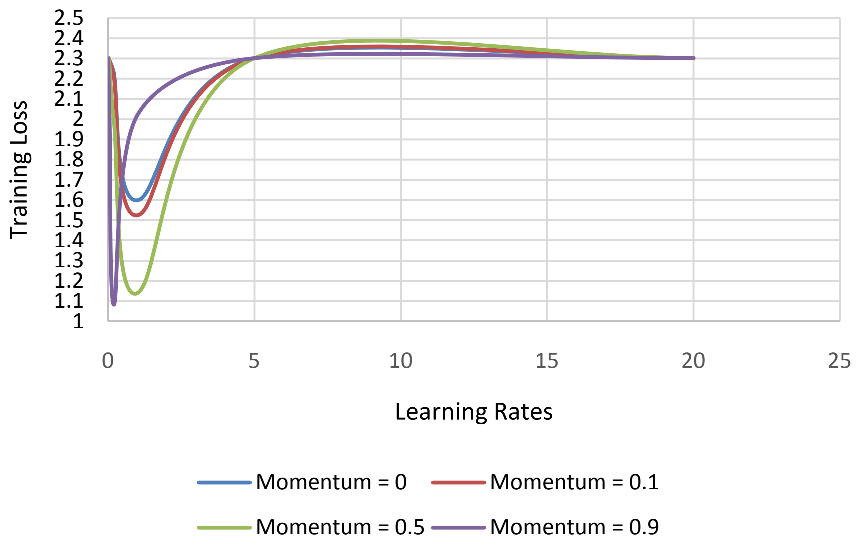

Figure 2. Training loss and learning rates.

momentum. We can see that the training loss value is smaller with momentum, and in our case, the value is the smallest at the biggest momentum (momentum = 0.9). Therefore, momentum is a good way to accelerate the training process. However, since the biggest momentum in our experiment is 0.9, we cannot

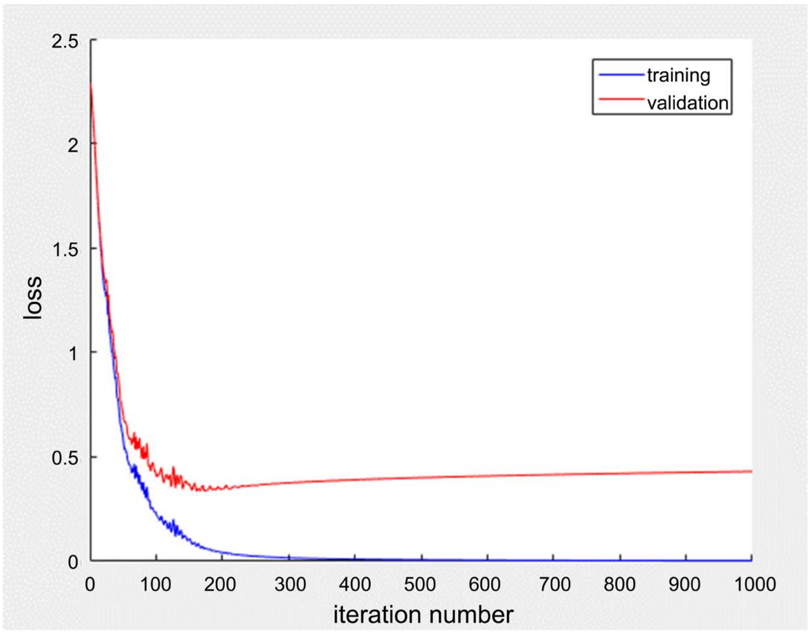

Figure 3. Behavior of the neural network at different momentum.

conclude that the behavior of the neural network with very large momentum would be better. In practice, we expect that too large momentum will also deteriorate the learning performance, which is known as "overshooting".

The neural network converges faster if the loss on the training data is smaller at certain iterations. From the data above, we can conclude that the best learning rate at which the neural network works falls at a specific range, in our case between 0.2 and 5.0, varying due to different momentum. With extremely small learning rate values (*i.e.* very close to 0), it would take very long time to train the neural network. On the other hand, with extremely large learning rate values, the neural network may draw unnecessary information from the data, and thus also decrease the efficiency of the training.

3. Generalization

In this step, we are trying to find a good generalization for the neural network by examining the classification loss on the validation data. We first investigate early stopping as the easiest way to improve network generalization [8]. As the number of iterations increases, the neural network may draw unnecessary information from the data (over-fitting) and thus generate worse results [9], which make the validation loss bigger. Early-stopping is a way to increase neural network's efficiency by choosing the model at the lowest validation loss, which will reduce the chance of over fitting. We first ran the code without early stopping. **Figure 2** shows the behavior of the neural network without early stopping. The validation data loss increased after approximately 200 iterations, which is a sign of over fitting. We then turned on early stopping. MATLAB shows that the validation loss was lowest after 161 iterations, which fitted the graph. Thus, by using early-stopping, we reduce the number of calculations by a great extent while getting better results: the validation loss was 0.43019 without early stopping and 0.33451 with early stopping.

We then ran the data with different weight decay (wd) coefficients [10]. The wd coefficient is a measure of preference on how simple we want our model to be. With larger value of coefficient, the neuron adds a bigger "weight" penalty to the overall loss function, and thus pushes the "weights" to be relatively smaller. The data in **Table 2** and **Figure 4** shows the results. **Figure 5** is a zoomed-in picture of the results where wd coefficient is in the range between 0 and 0.01.

The pattern is similar to that of learning rates. The lowest value of validation loss occurs in a specific range, in our case between wd coefficient of 0.0001 and 0.01. As wd coefficient goes toward extreme values (either close to 0 or infinity), the efficiency of the neural network decreases. This is reasonable. If a certain data is multiplied by too big or too small weight values, it would affect the overall summation by a significant degree, and thus decrease the efficiency of the neural network.

Another possible solution for overfitting is to regularize the number of hidden units [11]. Overfitting means that the neural network is drawing too much information from the data, many of which are necessary or unrelated. The training result of such neural network would be biased. For example, given images of 100 red apples and 10 green apples, a normal neural network may conclude that apples are round with a short stem on the top, while an overfitting neural network would see the color of red as a feature of apple and see the green apples as outliers. To reduce the possibility of overfitting, we reduce the number of hidden layer units to decrease the calculation capacity of the neural network. The results are shown in **Table 3** and **Figure 6**.

The data and graph demonstrate similar results as above that the best generalization occurs at a specific value, in this case, between hidden units 10 and 100. It is thus reasonable to conclude that different means of regularization would all work with the best efficiency at a certain value of that parameter.

What if we combine different ways of regularization methods? Would it generate even better results [12]? Here we combined the parameter of hidden layer units with early stopping. We implemented 38 neurons for the hidden layer,

Table 2. Wd coefficient-validation loss.

Wd coefficient	Validation loss
0	0.430185
0.0001	0.34829
0.001	0.28791
0.01	0.50976
1	2.30259
5	2.30259

Table 3. Hidden layers units-validation loss.

Hidden Layers Units	Validation Loss
10	0.42171
30	0.31708
100	0.36859
130	0.3976
200	0.430185

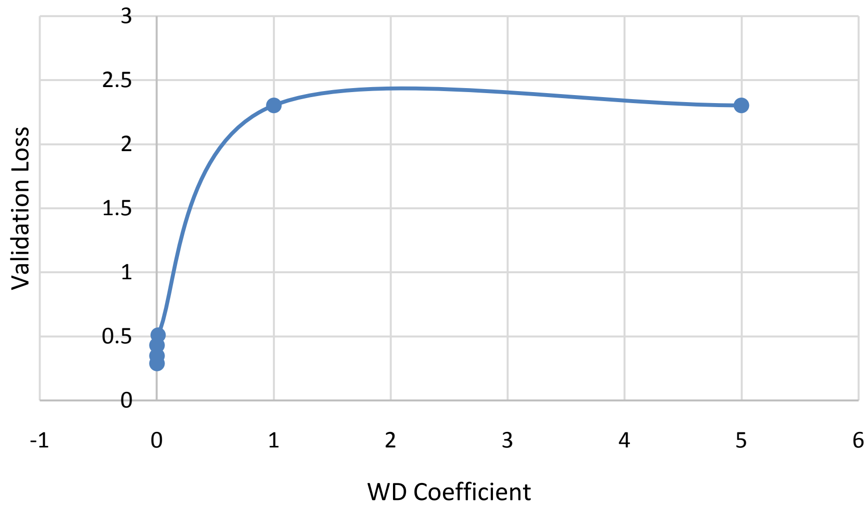

Figure 4. Validation loss-WD coefficient.

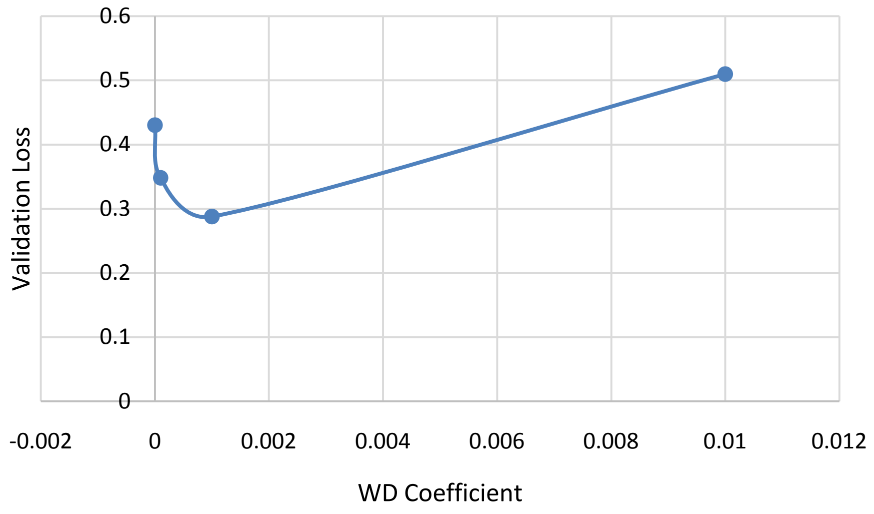

Figure 5. Validation loss-WD coefficient.

which achieves the best results in above experiments. Figure 7 shows a comparison of data with and without early stopping. The data with early stopping exhibits lower validation loss values. The absolute minimum is ever lower than that of the data without early stopping (0.28683 compared to 0.31708).

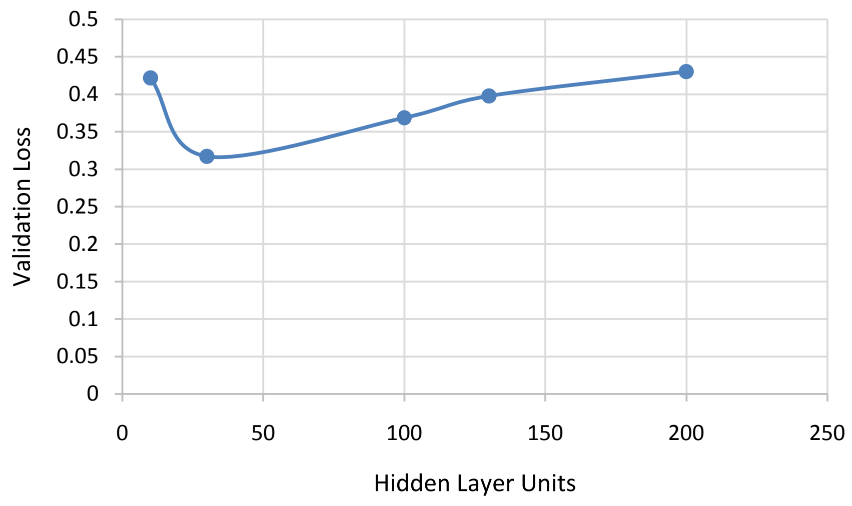

Figure 6. Validation loss-hidden layer units.

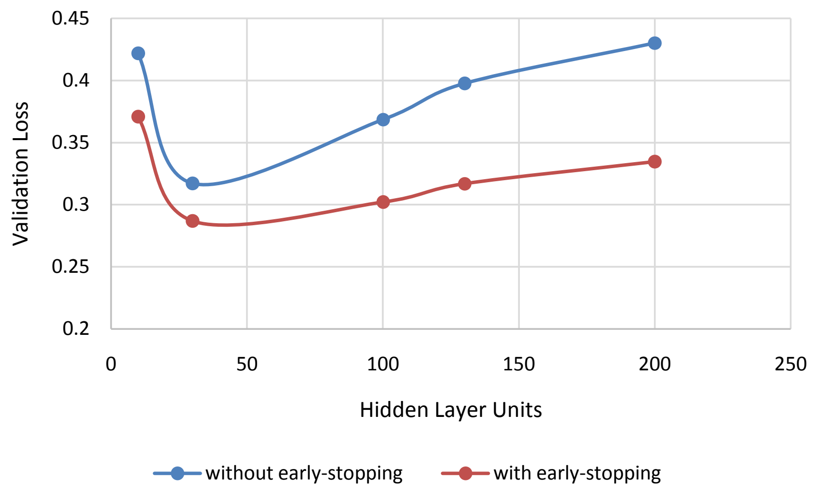

Figure 7. Validation loss-hidden layer units.

4. Initialization

In this section, we first implemented Restricted Boltzmann machine (RBM) to learn the feature of USPS data in an un-supervised fashion [13]. Then, we transfer the RBM weights into the weights of the input layer of our neural network as their initialization [14]. The RBM can generate a distribution of different weights to find the image feature better. Thus, we are using RBM's pre-trained results to produce best initialized weight values for the input neurons in order to increase the training efficiency of the neural network and get even greater accuracy on the validation datasets. Since we used the pre-trained model and thus did not worry about over-fitting too much in this case, we set the hidden layer size of our neural network to be 300, and compared the loss on the validation data with transferred initialized weights to that with randomly initialized weights. The former value was 0.058 and the latter value was 0.094 (shown in **Table 4**), which showed a significant difference. With good initial weights, we

Table 4. Validation loss.

	Validation Loss
a3_rbm_w	0.056
a3	0.091

can reduce the possibility of over-fitting by a great degree. When implemented, the code functions so that the neural network works to discover relevant information in the distribution of the input images. It means that the neural network will not focus only on the difference of the digit class labels, but will also analyze other information from the input data. In comparison to early-stopping the model in only a few iterations, this method makes the neural network work on something else which is also valuable.

In the process above, we turned off early-stopping to investigate solely on the effects of good weight-initialization. Now we turn on early-stopping to see if another regularization method would affect the results. The new validation loss with carefully weights initialization is 0.058 which is bigger than the loss without early-stopping but not significantly. Therefore, implementing early-stop is not necessary in our case when we have a good weight initialization.

5. Conclusion

As we explore the effects of different parameters on the feed-forward neural network, we discover the pattern that the best model is generated when the implemented parameter falls at an intermediate value. It is surprisingly similar to the learning pattern of human brain: learning too slow or having too few neurons to process the learned information would harm the learning efficiency, but learning too fast or thinking too much on a simple topic would also decrease the productivity. As we dig deeper into the function of neural network, we could also decipher more about the secrets of human brains.

References

[1] Thomas, M.S. and McClelland, J.L. (2008) Connectionist Models of Cognition. Cambridge handbook of Computational Cognitive Modelling, 23-58. https://doi.org/10.1017/CBO9780511816772.005

[2] McCulloch, W.S. and Pitts, W. (1943) A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, **5**, 115-133. https://doi.org/10.1007/BF02478259

[3] Preparata, F.P. and Shamos, M.I. (1985) Computational Geometry: An Introduction. Springer, New York, 1-35. https://doi.org/10.1007/978-1-4612-1098-6

[4] Werbos, P.J. (1974) Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Doctoral Dissertation, Harvard University, Cambridge.

[5] Khosravi, H. and Kabir, E. (2007) Introducing a Very Large Dataset of Handwritten Farsi Digits and a Study on Their Varieties. *Pattern Recognition Letters*, **28**, 1133-1141. https://doi.org/10.1016/j.patrec.2006.12.022

[6] Jin, W., Li, Z.J., Wei, L.S. and Zhen, H. (2000) The Improvements of BP Neural Network Learning Algorithm. 2000 *5th International Conference on Signal Processing Proceedings*, Beijing, 21-25 August 2000, 1647-1649.

[7] Attoh-Okine, N.O. (1999) Analysis of Learning Rate and Momentum Term in Backpropagation Neural Network Algorithm Trained to Predict Pavement Performance. *Advances in Engineering Software*, **30**, 291-302. https://doi.org/10.1016/S0965-9978(98)00071-4

[8] Caruana, R., Lawrence, S. and Giles, C.L. (2001) Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping. Neural Information Processing Systems, 402-408.

[9] Tetko, I.V., Livingstone, D.J. and Luik, A.I. (1995) Neural Network Studies. 1. Comparison of Overfitting and Overtraining. *Journal of Chemical Information and Computer Sciences*, **35**, 826-833. https://doi.org/10.1021/ci00027a006

[10] Krogh, A. and Hertz, J.A. (1992) A Simple Weight Decay Can Improve Generalization. Neural Information Processing Systems, 950-957.

[11] Baum, E.B. and Haussler, D. (1989) What Size Net Gives Valid Generalization? Neural Information Processing Systems, 81-90.

[12] Bishop, C.M. (2006) Pattern Recognition and Machine Learning. Springer, New York.

[13] Salakhutdinov, R., Mnih, A. and Hinton, G. (2007) Restricted Boltzmann Machines for Collaborative Filtering. *Proceedings of the 24th International Conference on Machine Learning*, Corvalis, 20-24 June 2007, 791-798. https://doi.org/10.1145/1273496.1273596

[14] Knerr, S., Personnaz, L. and Dreyfus, G. (1990) Single-Layer Learning Revisited: A Stepwise Procedure for Building and Training a Neural Network. *Neurocomputing: Algorithms, Architectures and Applications*, **68**, 71. https://doi.org/10.1007/978-3-642-76153-9_5