Arch Computat Methods Eng (2018) 25:121–129
https://doi.org/10.1007/s11831-017-9237-0

S.I.: MACHINE LEARNING IN COMPUTATIONAL MECHANICS

# Background Information of Deep Learning for Structural Engineering

**Seunghye Lee**[1] · **Jingwan Ha**[2] · **Mehriniso Zokhirova**[1] · **Hyeonjoon Moon**[2] · **Jaehong Lee**[1]

Received: 2 April 2017 / Accepted: 26 June 2017 / Published online: 3 July 2017

**Abstract** Since the first journal article on structural engineering applications of neural networks (NN) was published, there have been a large number of articles about structural analysis and design problems using machine learning techniques. However, due to a fundamental limitation of traditional methods, attempts to apply artificial NN concept to structural analysis problems have been reduced significantly over the last decade. Recent advances in deep learning techniques can provide a more suitable solution to those problems. In this study, versatile background information, such as alleviating overfitting methods with hyper-parameters, is presented. A well-known ten bar truss example is presented to show condition for neural networks, and role of hyper-parameters in the structures.

## 1 Introduction

Deep learning (DL) in artificial neural network (ANN) is a branch of machine learning based on a set of algorithms that attempt to model high level abstractions in data [1]. Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed [2]. Machine learning has been studied with three approaches, namely, neural modeling paradigm, symbolic concept-acquisition paradigm and modern knowledge-intensive paradigm [3]. The neural modeling paradigm was started with a perceptron and has developed to the deep learning. The perceptron is the first model which actually implemented the ANN. In machine learning, the perceptron is an algorithm for supervised learning and the simplest type of ANN [4]. In particular, neural networks with one or more hidden layers are called a multilayer perceptron (MLP) which is a feedforward neural network (FNN) model.

At the beginning of the new millennium, the neural modeling paradigm has been improved to the deep learning by advances in hardware using general purpose graphical processing units and innovative algorithms, such as a deep belief network (DBN) with restricted Boltzman machine (RBM) [5], rectified linear unit (ReLU) for activation function [6], and dropout algorithm for overfitting problems [7]. In this area, newer algorithms and more precise methods have been proposed by using new activation functions, loss functions, alleviating overfitting methods with hyper-parameters, and other effective methods. However, most researches have focused on applications of natural language processing (NLP), image recognition, signal-processing, information retrieval and molecular analysis [8].

Since the pioneering research of neural network application in structural engineering appeared in 1989 [9], a large number of articles about structural analysis and design problems have been published on these fields. However, over the last decade, using neural network in structural engineering application has been significantly reduced [10]. This is mainly due to the poor performance and enormous computational time of neural network model, especially for complicated problems with multiple hidden layers. Accordingly, neural network has shown its limitation and numerical instability in structural engineering field.

✉ Jaehong Lee
jhlee@sejong.ac.kr

1 Deep Learning Architecture Research Center, Sejong University, 209, Neungdong-ro, Gwangjin-gu, Seoul 05006, Korea

2 Department of Computer Science and Engineering, Sejong University, 209, Neungdong-ro, Gwangjin-gu, Seoul 05006, Korea

In this regard, the present paper investigates the state-of-the-art deep learning techniques applicable to structural analysis. To the best of authors' knowledge, the recent deep learning concepts such as rectified linear unit function, dropout, mini-batch, have not yet been applied to structural engineering field in the open literature. In this paper, a well-known ten-bar truss problem described in [11] is presented to show the efficiency and accuracy of the deep learning comparing to the conventional neural networks.

Because the ten bar truss is a simple regression problem, a neural network structure will be started with a feedforward neural network; Sect. 2 provides its structure with various means of designing layers. A supervised learning approach was adopted in the structure; various error cost functions and their effects on results are described in Sect. 3. Section 4 presents a variety of alleviating overfitting methods with hyper-parameters, namely, the learning rate selection, momentum factors, activation functions, and regularizations. As one of its major contributions, this work serves to lower barriers to apply deep learning techniques to structural analysis topics.

## 2 Feedforward Neural Network Basics

### 2.1 The Network Architecture

In the feedforward neural network, each layer contains connections to the next layer. Figure 1 shows the architecture of feedforward neural network with a two-layer perceptron. Figure 1 shows a fully connected network; the unit of $j$th layer $u_j$ $(j = 1, 2, \cdots, J)$ receives a sum of inputs $x_i$ $(i = 1, 2, \cdots, I)$ which is multiplied by the weight function $w_{ji}$ as follows:

$$u_j = \sum_{i=1}^{I} w_{ji} x_i \tag{1}$$

A activation function $f(\cdot)$ is needed to transform the the unit of $j$th (the weighted sum of the inputs) into output signals in the $j$th layer $(z_j)$ as shown below:

$$z_j = f\left(u_j\right) \tag{2}$$

The network must be trained to obtain an acceptable result for given input–output training sets. The result is

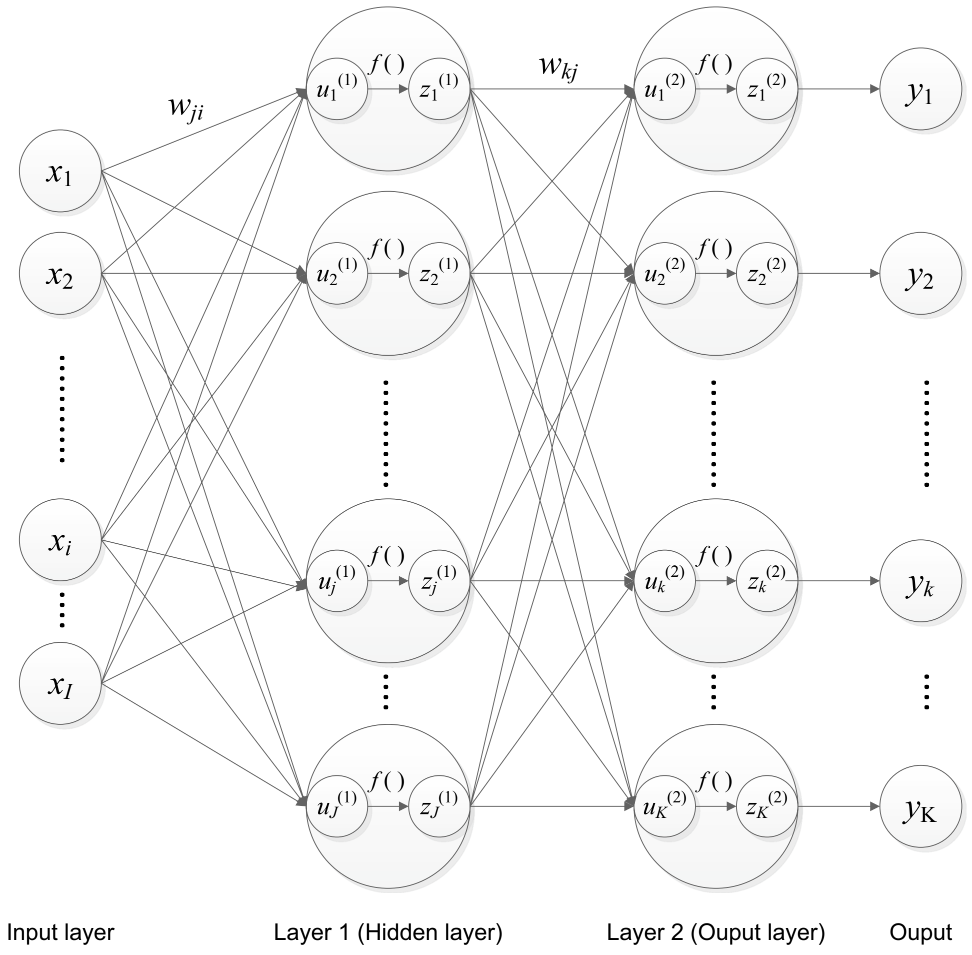


**Fig. 1** Feedforward neural network structure with one hidden layer

then compared to the target output, using a loss function (error function), and an error value is calculated for each of the neurons in the output layer. The input **x** from the training set produces an output $y_k$ different from the target $t_k$; the most-used mean square error (MSE) function $E_{MSE}$ of the network is defined as follows:

$$E_{MSE} = \frac{1}{n}\sum_{k}^{n}\left(y_k - t_k\right)^2 \quad (3)$$

where $n$ contains the number of training elements times number of output neurons.

## 2.2 Supervised Learning Algorithms with Backpropagation

Supervised learning is the machine learning task of inferring a weight function from a training set of examples associated with labels or targets [12]. Here the term training means to achieve the optimal weight parameter from the training set automatically. The optimal weight parameter can be obtained using an optimizer of gradient descent method which is to find a local minimum of that function. In multilayer feedforward neural networks, a backpropagation algorithm is used to calculate the effect of each weight with respect to a loss function [13]. Weight functions can be updated in the backpropagation algorithm with the gradient descent method as follows:

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \epsilon\partial\mathbf{W}^{(t)} + \mu\Delta\mathbf{W}^{(t-1)} \quad (4)$$

where **W** is the matrix form of weight function set; $t$ refers to the epoch through the loop. In Eq. (4), $\epsilon$ is a learning rate which indicates a step size; $\eta$ denotes a momentum parameter which scales the influence of previous weight-step on the current one [14]. The backpropagation algorithm can be decomposed in the following four steps [15]:

– Feedforward computation
– Backpropagation to the output layer
– Backpropagation to the hidden layer
– Weight updates.

## 2.3 Data sets and Mini-batch

As shown in Fig. 2, we give the ten bar truss example to present alleviating overfitting methods with hyper-parameters or error cost methods in the neural networks. The ten bar truss problem is basically a common problem in the field of structural design to verify the efficiency of various optimization methods. However, in this study, we focused on structural analysis results of the system. Figure 2 shows the geometry, loading, and support conditions for this cantilevered truss. The material density and the modulus of elasticity are 0.1 lb/in$^3$ (2768.0 kg/m$^3$) and 10, 000 ksi (68, 950 MPa), respectively.

In machine learning problems, datesets are divided into two parts, training set and test set. The former term is a set of examples used for learning, while the latter is a set of

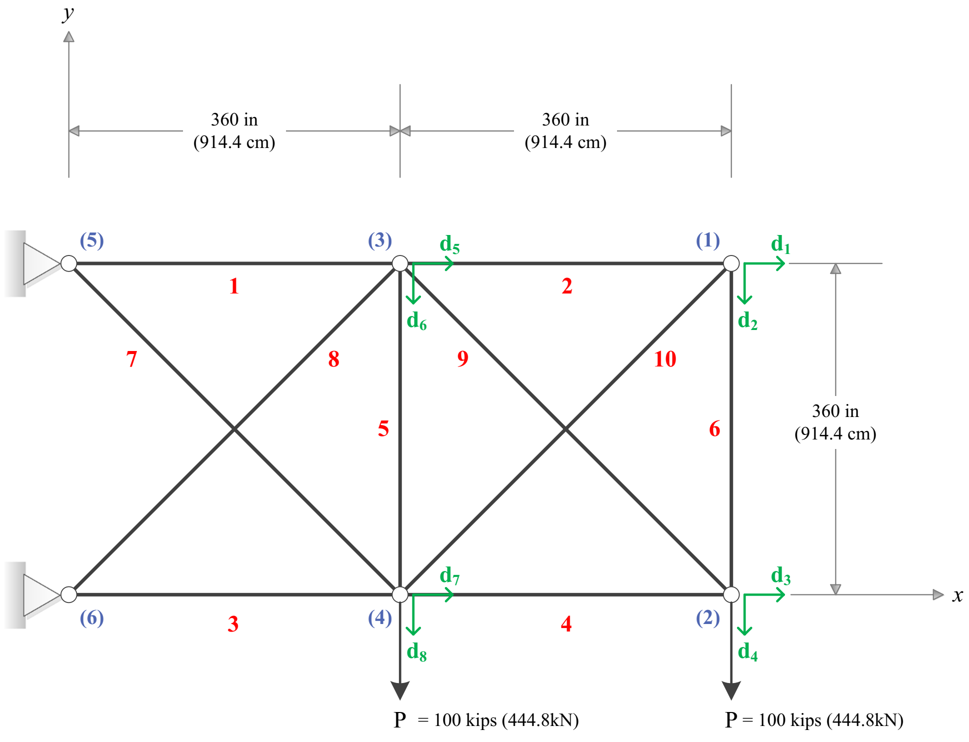


**Fig. 2** A 10-bar planar truss

new data used only to assess the generalization. Because the ability of a predictive model to perform on new dataset is needed in machine learning problems, it is important that we insist on partitioning data into separate roles of training and test rather than just working with all of the data. In this study, training sets generated by a random distribution of cross-sectional area design variables ranging from 0.1 to 35.0 in$^2$ (from 0.6 to 225.8 cm$^2$) were used to train the neural network. Note that all of input data are normalized with respect to the maximum cross-sectional area 35.0 for regularization. We used 500 training sets randomly; 20sets are used for test sets.

Most recently, mini-batch algorithms have been proposed as a way to speed-up stochastic convex optimization problems. The mini-batch method means to divide the training set into small “mini-batches” [16]. The method is an effective technique for solving regularized loss minimization problems and reducing the iteration cost. In this study, 80 mini-batch sets were used to obtain these effects. Figure 3 gives the loss curves for 10−20−2 architecture and shows comparison of two results according to existence of the mini-batch. After 10,000 epochs, the results without mini-batch case and mean square error for training and test data obtained are 4.219 and 2.450, respectively, as shown in Fig. 3a. However, according to Fig. 3b, 80 mini-batch case demonstrated higher accuracy (the mean square error for training and test data : 0.855 and 0.801) than the results without mini-batch. The mini-batch method can give a dramatically faster convergence of loss curves to the networks.

However, depending on the test sets, the algorithm may or may not converge.This is a overfitting problem which is the use of models or procedures that violate parsimony. The overfitting can be divided into two types: using a model that is too flexible, and a model that includes irrelevant components [17]. In order to prevent overfitting problems, it is necessary to use alleviating overfitting methods with hyper-parameters or new activation functions.

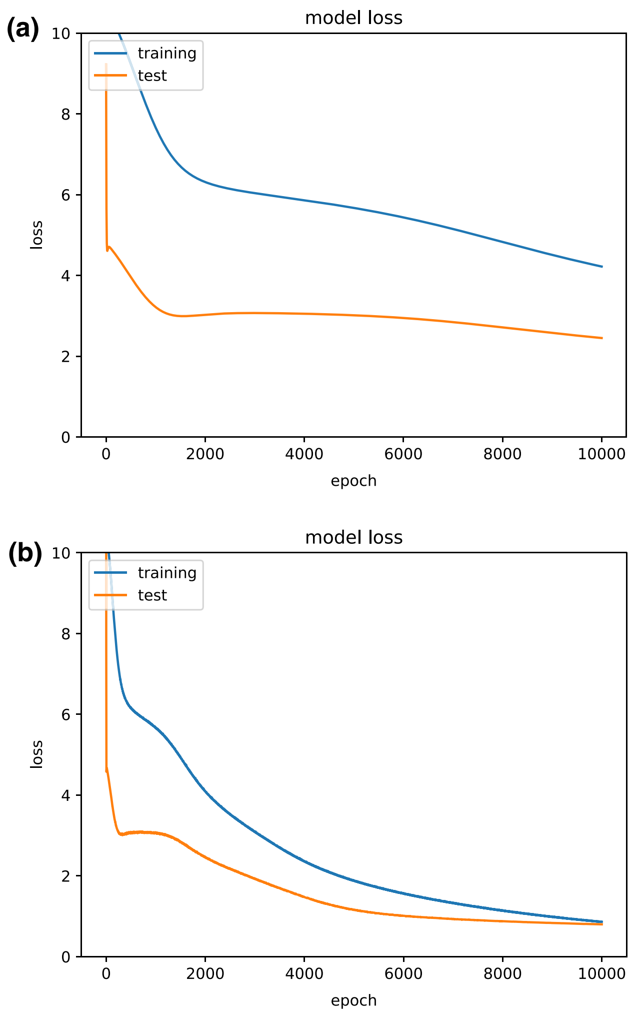


**Fig. 3** Mean square error for (10−20−2 architecture): comparisons between the training data and test data. The sigmoid function, stochastic gradient descent (SGD) ($\epsilon$ = 0.01 and $\eta$ = 0.9) and 500 input sets are used for the basic neural network structure. **a** Without mini-batch. The mean square error for training and test data : 4.219 and 2.450. **b** Mini-batch = 80. The mean square error for training and test data : 0.855 and 0.801. Normal scale

## 2.4 Number of Hidden Layer Neurons

According to Fig. 2, the notation $d$ indicates displacements at each point. In typical structural optimization problems with the ten bar truss, two vertical displacements ($d_2$ and $d_4$ ) are considered as critical constraints and are limited to ±2 in. A basic neural network structure is set using two vertical displacements for the output layer and ten cross sectional areas for the input layer. The mean square error function and a sigmoid function are used for evaluation and activation function, respectively. A stochastic gradient descent (SGD) ($\epsilon$ = 0.01 and $\eta$ = 0.9) is used for optimization. In this study, the simulations were based on libraries Theano [18] and Keras [19].

To optimize the number of hidden layer neurons remains one of the difficult tasks in research areas. Setting too few and too many hidden units cause high training errors and high generalization errors, respectively. For a consideration of this topic, (10−10−2), (10−20−2), (10−30−2), (10−40−2), and (10−50−2) architectures were analyzed. In a previous study, a (10-11-2) architecture with a single hidden layer was presented for the ten bar truss problem [20]. Figure 4a and b show box-whiskers plots of training mean square error in trained network is varied from 10 to 50, and their mean plus or minus one standard deviation, respectively. Each result of the network was tested with ten simulations, each with a different starting random weight. As can be seen from Fig. 4,

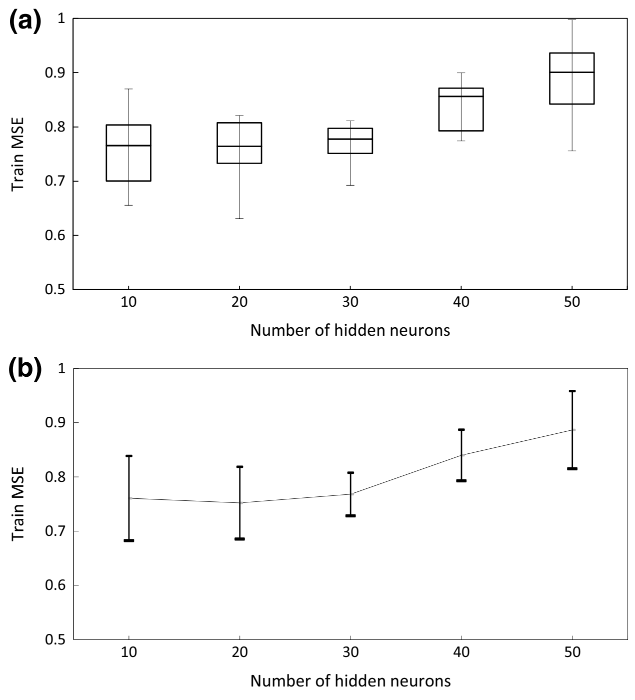


**Fig. 4** **a** Box-whiskers plots of training mean square error by changing number of hidden neurons; **b** mean plus or minus one standard deviation. Each result is averaged over ten simulations. The sigmoid function, stochastic gradient descent (SGD) ($\epsilon = 0.01$ and $\eta = 0.9$) and 500 input sets are used for the basic neural network structure. Normal scale

the best mean training and generalisation error occurs for networks with 20 hidden units. This trend varies according to the generating network size, namely, the number of inputs, hidden neurons and outputs [21].

### 2.5 Activation Functions

Until now, the sigmoid activation function has been used to produce output signals for the presented network. Choosing an appropriate activation function is an important consideration because it can affect how you must format input data. More recently, the rectified linear unit (ReLU) or a softplus activation [22] are preferred activation functions. Table 1 and Fig. 5 indicate equations and graphs of the most commonly used activation functions in neural networks literature. We trained the basic model using the logistic sigmoid, hyperbolic tangent (tanh), softplus and rectified linear unit (ReLU) functions for 10,000 epochs. The results are shown in Fig. 6; the ReLU activation function outperforms all other functions and achieves the lowest training mean square loss after 10,000 epochs as shown in Table 1.

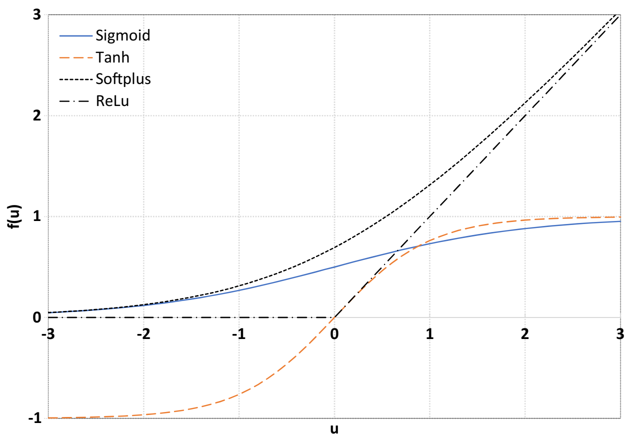


**Fig. 5** Commonly used activation functions in neural networks literature: logistic sigmoid, hyperbolic tangent (tanh), softplus and rectified linear unit (ReLU) functions. Normal scale

## 3 Numerical Example of Single-Layer Perceptron

### 3.1 Selection of Output Neurons

In the previous section, we have dealt with only two outputs, namely, two vertical displacements ($d_2$ and $d_4$ ). However, in numerous cases, the whole displacement (four horizontal and four vertical displacements ($d_1$ to $d_8$)) and stress of each element are demanded for calculating maximum allowable static displacements and stress values. In this section, we will discuss practical matters regarding the effects of combination using various optimizers and activation functions. Note that all of

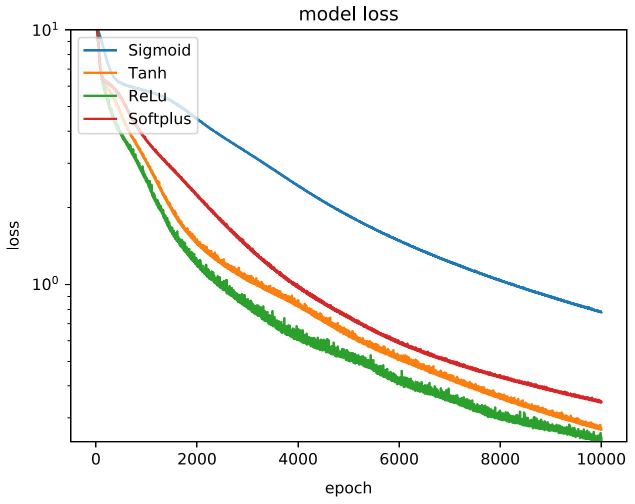


**Fig. 6** Training mean square error for (10–20–2 architecture): comparisons among various activation functions. The stochastic gradient descent (SGD) ($\epsilon = 0.01$ and $\eta = 0.9$) and 500 input sets are used for the basic neural network structure. Log scale

**Table 1** Training mean square error for (10−20−2 architecture) after 10,000 epochs: comparisons among various activation functions. The stochastic gradient descent (SGD) ($\epsilon = 0.01$ and $\eta = 0.9$ ) and 500 input sets are used for the basic neural network structure

| Network | Activation function | Equation | Mean square error (MSE) |
|---|---|---|---|
| 10−20−2 | Logistic (sigmoid) | $f(u) = \frac{1}{1+e^{-u}}$ | 0.775 |
| 10−20−2 | Hyperbolic tangent (Tanh) | $f(u) = tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$ | 0.266 |
| 10−20−2 | Softplus | $f(u) = ln(1 + e^u)$ | 0.342 |
| 10−20−2 | Rectified linear units (ReLU) | $f(u) = \begin{cases} 0 \ for \ u < 0 \\ u \ for \ u \geq 0 \end{cases}$ | 0.245 |

output stress data are normalized for regularization. We used 500 training sets randomly; 20 for improving performance of the network. The structure of network has (10−20−18); 10,000 epochs are then iterated in every case. For combinations between activation functions and optimizers, four activation functions (logistic sigmoid, hyperbolic tangent (tanh), softplus and rectified linear unit (ReLU)) and five optimization algorithms stochastic gradient descent (SGD), AdaGrad, Adadelta, RMSProp and Adam) are used. Each result of the case was tested with ten simulations, the average mean square error of training and test sets can be obtained.

## 3.2 Optimization Algorithms

In machine learning, the SGD is one of the simplest and most popular first-order method to solve stochastic optimization problems. However, the method requires a number of hyper-parameters or is quite inefficient depending on problems. For this reason, several state of the art algorithms for optimizing deep learning models have been evolved. This article introduces five methods including the SGD as optimization algorithms.

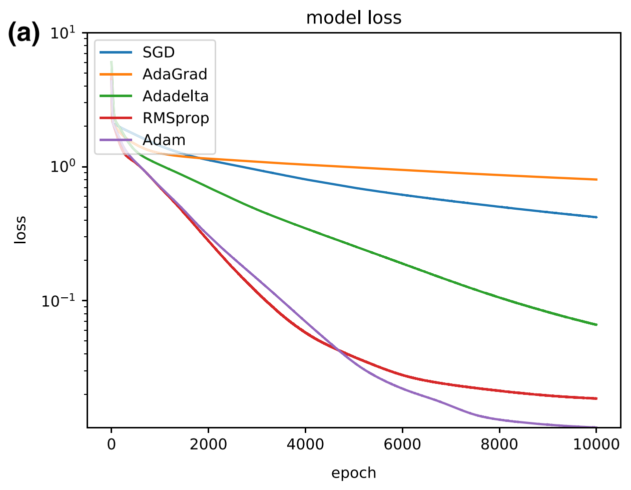
In machine learning processes, the learning rate value ($\epsilon$ ) is an important factor; when the learning rate is too small, the learning algorithm converges very slowly, on the contrary, if the value is too large, the network behaves chaotically and fails to converge. An effective way to determine an optimum leaning rate is a learning rate decay. By using this method, lower learning rate is calculated as the training progresses. AdaGrad (adaptive gradient algorithm) [23] stems from this paradigm in stochastic optimization; that is a learning-theoretic technique for learning rate adaptation which is given by the square root of sum of squares of the historical, component-wise gradient. Adadelta [24] is an extension of the AdaGrad that can improve upon the continual decay of learning rates throughout training, and the need for a manually selected global learning rate. RMSProp [25] is another extension of the AdaGrad in which the learning rate is adapted for each of the parameters. In this method, the learning rate is divided by an exponentially decaying average of squared gradients. Adam (adaptive moment estimation) is a method for efficient stochastic optimization that only requires first-order gradients with less memory requirement [26]. This method is designed to combine the advantages of AdaGrad and RMSProp. The RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances [27]. The Adam might be the best overall choice, however, there is no most efficient method among gradient descent optimization algorithms. Because it depends on problems and conditions, various gradient descent optimization algorithms can be selected to obtain the optimum solution.

## 3.3 Results of (10–20–18) Network

Table 2 shows results of the mean square error for training and test sets with (10−20−18) architecture after 10,000 epochs. The result in Table 2 indicates that the lowest mean square error for training set is obtained when the combination of Adam optimizer and softplus activation function is used (MSE

**Table 2** Mean square error for training and test sets with (10−20−18) architecture after 10,000 epochs: comparisons among various gradient descent optimization algorithms in combination with activation functions. Each value is an average of ten simulations

| Optimizer | SGD | | AdaGrad | | Adadelta | | RMSprop | | Adam | |
|---|---|---|---|---|---|---|---|---|---|---|
| Activation function | Training | Test | Training | Test | Training | Test | Training | Test | Training | Test |
| Sigmoid | 0.093 | 0.202 | 0.032 | 0.190 | 0.127 | 0.155 | 0.015 | 0.245 | 0.011 | 0.289 |
| Tanh | 0.055 | 0.364 | 0.047 | 0.336 | 0.106 | 0.267 | 0.053 | 0.403 | 0.025 | 0.630 |
| Softplus | 0.049 | 0.420 | 0.019 | 0.178 | 0.072 | 0.134 | 0.014 | 0.183 | 0.008 | 0.235 |
| ReLU | 0.058 | 0.690 | 0.118 | 0.835 | 0.099 | 0.374 | 0.032 | 0.338 | 0.100 | 0.502 |



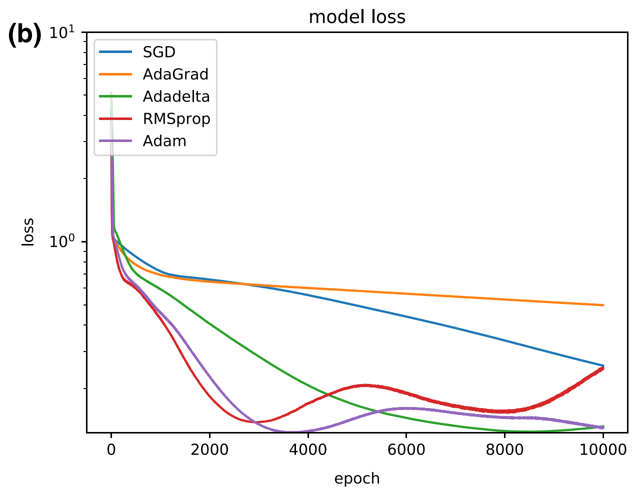


**Fig. 7** Mean square error for **a** training set and **b** test set, with (10−20−18) architecture after 10,000 epochs: comparisons among various gradient descent optimization algorithms in combination with the softplus activation function. Log scale

of training set = 0.008 ). To enhance our understanding of the effects of Optimization algorithms on the training set, five examples are iterated as shown in Fig. 7a. The convergence of the case which uses the Adam algorithm based on the softplus activation function shows the minimum loss. After the 4000th epoch, the combination converges rapidly toward zero.

However, the case of test set differed from trend of the training set. According to Table 2, the average loss of ten simulations in the test set result of Adam optimizer case is not minimum value compared to the softplus activation function. It is clear, from Fig. 7b that two convergence lines associated with Adam and RMSprop optimization algorithm show undulating trend after epoch 2000. In order to solve poor performance problems, the model structures need to insert one or more hidden layer to the architectures.

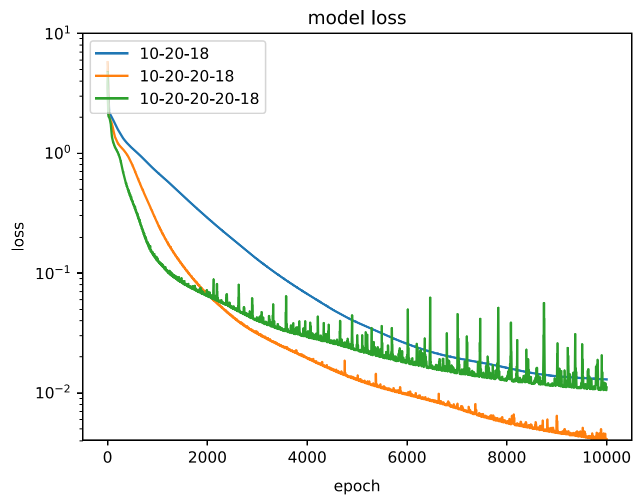


**Fig. 8** Mean square error for a training set with (10−20−18), (10−20−20−18), and (10−20−20−20−18) architectures after 10,000 epochs: comparisons among various number of hidden layers in combination with the softplus activation function and Adam optimization. Log scale

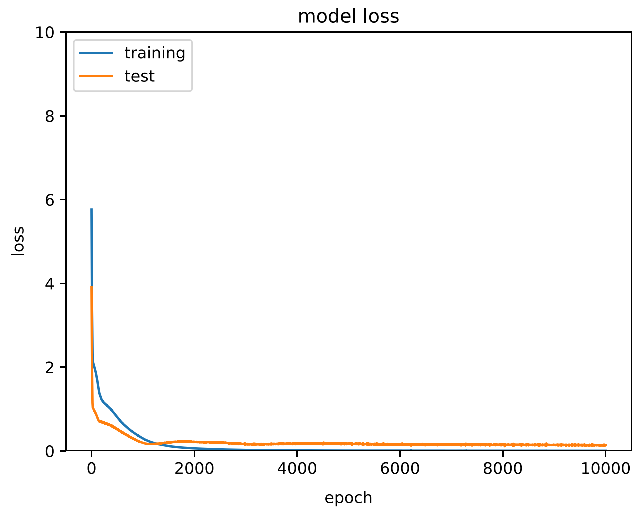


**Fig. 9** Mean square error for (10−20−20−18 architecture): comparisons between the training data and test data. The softplus function, Adam optimization method and 500 input sets are used for the basic neural network structure. Mini-batch=80. Normal scale

## 4 Enhanced Networks

### 4.1 Multilayer Neural Network

So far, we simulated the numerical example of single-layer network, namely, the (10−20−18) architecture. The size and complexity of a neural network depend on both the total number of neurons and the number of hidden layers. More layers can handle complex decision and multiple

**Table 3** Mean square error for training and test sets with (10−20−20−18) architecture after 10,000 epochs: comparisons among various gradient descent optimization algorithms in combination with activation functions. Each value is an average of ten simulations

| Optimizer | SGD | | AdaGrad | | Adadelta | | RMSprop | | Adam | |
|---|---|---|---|---|---|---|---|---|---|---|
| Activation function | Training | Test | Training | Test | Training | Test | Training | Test | Training | Test |
| Sigmoid | 0.050 | 0.346 | 0.022 | 0.321 | 0.057 | 0.335 | 0.012 | 0.171 | 0.005 | 0.288 |
| Tanh | 0.023 | 0.278 | 0.025 | 0.312 | 0.037 | 0.271 | 0.068 | 0.347 | 0.010 | 0.261 |
| Softplus | 0.014 | 0.419 | 0.009 | 0.201 | 0.026 | 0.193 | 0.013 | 0.137 | 0.003 | 0.107 |
| ReLU | 0.023 | 0.656 | 0.085 | 0.783 | 0.031 | 0.439 | 0.027 | 0.361 | 0.019 | 0.394 |

interactions between parameters. However, because implementations of multilayer neural networks will demand huge resource, it might not converge toward a feasible solution. In this section, we will cover two more architectures, namely, (10−20−20−18) and (10−20−20−20−18). Each structure has two hidden layers and three hidden layers, respectively.
In Fig. 8, the results of mean square error for training set are presented to compare effects of number of hidden layers. On this occasion, the softplus activation function and Adam optimization were used; other conditions are same as the previous examples. We note that the case of two hidden layers, namely, (10−20−20−18) architecture, achieves the competitive convergence performance.

### 4.2 Results of (10–20–20–18) Network

Figure 9 displays the progress of the (10−20−20−18) structure in normal scale. The multilayer neural network with the softplus activation function and Adam optimization is seen to achieve a significantly faster convergence rate than the case of single hidden layer with the sigmoid activation function and SGD optimization (Fig. 3b).

We also consider the use of other optimization algorithms and activation functions in the case of two hidden layers. Table 3 shows results of the mean square error for training and test sets with (10−20−20−18) architecture after 10,000 epochs. The combination of softplus activation function and Adam optimization performs the lowest loss value among other available combinations; the average MSE of training and test sets with ten simulations are 0.003 and 0.107, respectively. Moreover, in this case, both of training and test show the minimum values among others.

## 5 Conclusion

This work analyzed the recent deep learning methods to apply the innovative algorithms on structural analysis problems. There are many deep learning algorithms that are currently in development. However, the innovative methods have not been used to structural analysis research topics. The background information described here can directly help guide structural engineers. For the ten bar truss example, the softplus activation function and Adam optimization are effective in the two hidden layer perceptron structure. Nevertheless, it depends on architectures or complexity of the examples, hence, the background information can be extended to find more effective methods or conditions for the structures. For future work, we would look to build the structural optimization method using various deep learning methods.

**Funding** This research was funded by a grant (NRF-2017R1A4A1015660) from NRF (National Research Foundation of Korea) funded by MEST (Ministry of Education and Science Technology) of Korean government.

**Compliance with Ethical Standards**

**Conflict of interest** The authors declare that they have no conflict of interest.

## References

1. Bengio Y, Goodfellow IJ, Courville A (2015) Deep learning. MIT Press, Cambridge
2. Samuel AL (1959) Some studies in machine learning using the game of checkers. IBM J Res Dev 3(3):210–229
3. Carbonell JG, Michalski RS, Mitchell TM (1983) Machine learning: a historical and methodological analysis. AI Mag 4(3):69
4. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65(6):386
5. Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554
6. Nair V, Hinton G E (2010) Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning (ICML-10) (pp 807–814)
7. Hinton GE, Srivastava,N, Krizhevsky A, Sutskever I, Salakhutdinov, RR (2012) Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580
8. Deng L (2014) A tutorial survey of architectures, algorithms, and applications for deep learning. APSIPA Trans Signal Inform Process 3:e2

9. Adeli H (2001) Neural networks in civil engineering: 19892000. Comput-Aided Civ Infrastruct Eng 16(2):126–142
10. Gupta T, Sharma RK (2011) Structural analysis and design of buildings using neural network: a review. Int J Eng Manag Sci 2(4):216–220
11. Haftka RT, Grdal Z (2012) Elements of structural optimization, vol 11. Springer, Dordrecht
12. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge
13. Riedmiller, M, Braun H (1993) A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In: IEEE international conference on neural networks, 1993, (pp 586–591)
14. Rumelhart, D. E., McClelland, J. L., and PDP Research Group. (1988). Parallel distributed processing. In: IEEE (Vol. 1, pp. 443–453)
15. Rojas R (1996) The backpropagation algorithm. In: Neural networks. Springer, Berlin, pp 149–182
16. Hinton G (2010) A practical guide to training restricted Boltzmann machines. Momentum 9(1):926
17. Hawkins DM (2004) The problem of overfitting. J Chem Inf and Comput Sci 44(1):1–12
18. Bastien F, Lamblin,P, Pascanu R, Bergstra J, Goodfellow I, Bergeron A, Bouchard N, Warde-Farley D, Bengio, Y (2012) Theano: new features and speed improvements. arXiv preprint arXiv:1211.5590
19. Chollet F (2015) Keras: Theano-based deep learning library. Code: https://github.com/fchollet. Documentation: http://keras.io
20. Hajela P, Berke L (1991) Neurobiological computational models in structural analysis and design. Computers and Structures 41(4):657–667
21. Lawrence S, Giles CL, Tsoi AC (1996) What size neural network gives optimal generalization? Convergence properties of backpropagation. Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute for Advanced Computer Studies, University of Maryland
22. Dugas C, Bengio Y, Blisle F, Nadeau C, Garcia R (2001) Incorporating second-order functional knowledge for better option pricing. Adv Neural Inf Process Syst 472–478
23. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12:2121–2159
24. Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701
25. Tieleman, T. and Hinton, G. Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report, 2012
26. Kingma D, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980
27. Ruder S (2016) An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747