Two-Stage Backward Elimination Method for Neural Networks Model Reduction

Xiaoquan Tang\textsuperscript{1} and Long Zhang\textsuperscript{2}
\textsuperscript{1} School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan 430074, China
\textsuperscript{2} School of Electrical and Electronic Engineering, The University of Manchester, Manchester, M13 9PL, United Kingdom

E-mail: \textsuperscript{1}xqtang2016@hust.edu.cn; \textsuperscript{2}long.zhang@manchester.ac.uk

Abstract. The single-hidden-layer neural networks (NN) has been widely used for complex system identification. However, the hidden neurons are often determined by trial-and-error method and the amount of neurons is usually large. This commonly leads to over-fitting problem and the training process is time consuming. In this paper, we propose a two-stage backward elimination (TSBE) method to obtain a parsimonious network with fewer hidden neurons but remains a good performance and saves training time. In the first stage, neural networks with a predetermined number of hidden neurons is trained based on stochastic gradient decent (SGD) algorithm with part of training data and Least absolute shrinkage and selection operator (Lasso) is applied for dropping redundant neurons leading to a simplified neural model. In the second stage, the remaining training data is used to update the parameters of the simplified neural model. A simulation example is used to validate and show that the novel approach gives a more compressed model and higher level of accuracy comparing with the recently proposed pruning-based method.

1. Introduction

System identification is widely used for modelling dynamic systems. Most real-world systems are highly nonlinear. Therefore, neural networks is often used to describe the unknown nonlinearities in the system. This is achieved by directly processing raw measurement data of system and automatically learning the description of the system [1]. When building neural networks, determining the amount of hidden neurons and hidden layers is needed. Currently, the widely used method for setting the hidden-neuron numbers is the trial-and-error which generally sets the amount of hidden neurons without any prior, then tests the performance of obtained model. The process mentioned above is usually time consuming. In addition, it often produces an overly complex network with a huge amount of hidden neurons. According to the parsimony principle, a simpler model is preferable to a more complex model when two models produce similar accuracy. The main reason is that the over complex model may require large computations and cause over-fitting problem with a poor generalization performance.

In order to obtain a simplified network, the network pruning method removes the neurons whose weights are smaller than a pre-determined threshold. Recently, an improved pruning method is proposed which is a two-step learning method and has attracted lots of attentions [2]. Pruning or those related methods either calculate sensitivity of error to the removal connections or add penalty to cost function leading to a parsimonious networks [3]. However, most of sensitivity-based methods can not remove those correlated connections. Further, it may be difficult to set a proper threshold to drop redundant
connections, especially when all the connection weight are similar and the number of small weights is only few or very limited.

Another class of related works randomly reduce the complexity of neural networks during the training process. Dropout is the most popular one of these model reduction methods [4]. Although it has attracted significant attentions, dropout may not finally provide a parsimonious model. In addition, a class of methods focus on reducing the storage requirements of the neural networks where the low-rank approximation is commonly used, although its approximation performance may not be guaranteed [5]. A recent work called Dropneuron which can drop neurons and connections simultaneously by regularising all incoming connections’ weights and outgoing connections’ weights to be zeros with combining the pruning method[6]. Although Dropneuron can obtain a more parsimonious networks, it has the same problems with pruning-based methods mentioned above, namely, it is difficult to set a suitable threshold and remove correlated connections. More recently, a novel method is proposed to perform pruning and feature selection simultaneously using sparse group Lasso penalty [7]. It can be viewed as a method based on forward selection method which generally begins with an empty neural model, then gradually adds input neurons using feature selection methods.

In this paper, we propose a novel two-stage backward elimination method to obtain a simplified networks by directly dropping neurons with the aid of Lasso. In the first stage, the predetermined neural networks are trained using SGD with part of training data and then Lasso method is applied for dropping those redundant hidden neurons to obtain a simplified networks. The dropping method can be treated as backward elimination method as it can simplify neural networks by removing redundant neurons from output layer to input layer. In the second stage, updating the parameters of selected neurons is carried out with the remaining train data.

2. Nonlinear system identification via neural networks
With statistical learning algorithms, neural networks can approximate any nonlinear system, even just with a single hidden layer. A large number of neurons can effectively reduce training error. However, a complex neural networks that have too many unnecessary neurons could have very poor testing performance.

A nonlinear system could be described as follows:

\[ y(t) = \sum_{i=1}^{M} \beta_i \phi_i(w_i x(t) + b_i) + e(t) \]  

(1)

here \( y(t) \in \mathbb{R}^N, x(t) \in \mathbb{R}^{r \times N} \) are neural networks output and input, and \( w_i \) is the weights between \( i^{th} \) hidden neuron and neurons of input layer with \( w_i \in \mathbb{R}^r \). \( M \) represents the amount of hidden neurons and \( b_i \) are bias. Suppose \( \phi_i(x) = \frac{1}{1 + \exp(-x)} \), where \( x \) is the input variable.

We rewrite equation (1) as matrix form:

\[ y = \mathbf{P} \Theta + \Xi \]  

(2)

where

\[ y = \begin{bmatrix} y(1), y(2), \cdots, y(N) \end{bmatrix}^T, \Theta = \begin{bmatrix} \beta_1, \beta_2, \cdots, \beta_M \end{bmatrix}^T, \Xi = \begin{bmatrix} e(1), e(2), \cdots, e(N) \end{bmatrix}^T \]  

(3)

and

\[ \mathbf{P} = \begin{bmatrix} \phi_1(w_1 x(1) + b_1) & \phi_2(w_2 x(1) + b_2) & \cdots & \phi_M(w_M x(1) + b_M) \\ \phi_1(w_1 x(2) + b_1) & \phi_2(w_2 x(2) + b_2) & \cdots & \phi_M(w_M x(2) + b_M) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(w_1 x(N) + b_1) & \phi_2(w_2 x(N) + b_2) & \cdots & \phi_M(w_M x(N) + b_M) \end{bmatrix} \]  

(4)
For convenience, \( P \) can be re-written as

\[
P = [P_1, P_2, \cdots, P_M]
\] (5)

The paper deals with neuron dropping, the resultant model only have a proportion of neurons. Therefore, here, define 

\[
P_\varepsilon = [P_{i_1}, P_{i_2}, \cdots, P_{i_k}]
\] for the set \( \varepsilon = \{i_1, i_2, \cdots, i_k\} \subseteq \{1, 2, \cdots, M\} \) and \( P_{-\varepsilon} \) contains those components not in \( \varepsilon \).

3. A two-stage backward elimination method

3.1. Drop redundant neurons with lasso

For a single hidden layer network, the model output could be described as a linear combination of hidden neurons. Lasso can be used for selecting important neurons and dropping those unimportant neurons to make the networks to be simplified as Figure 1. More specifically, Lasso algorithm minimizes the residual sum of squares by

\[
\hat{\Theta} = \arg \min_{\Theta} \frac{1}{2} \sum_{t=1}^{N} (y(t) - \sum_{i=1}^{M} \Theta_i \phi_i(w_ix(t) + b_i))^2 + \lambda \sum_{i=1}^{M} |\Theta_i|
\] (6)

which can be re-written as matrix format:

\[
\hat{\Theta} = \arg \min_{\Theta} \frac{1}{2} \|y - P\Theta\|^2 + \lambda \|\Theta\|_1
\] (7)

To write lasso solution with a more explicit form, we turn to the Karush-Kuhn-Tucker (KKT) optimal conditions for problem (7), which can be expressed as[8]:

\[
\gamma_i \in \{\text{sign}(\hat{\Theta}_i)\} \quad \text{if} \quad \hat{\Theta}_i \neq 0
\]

\[
[-1,1] \quad \text{if} \quad \hat{\Theta}_i = 0
\]

for \( i = 1, 2, \cdots, M \).

\[
(8)
\]

\( \hat{\Theta} \) is the solution of lasso problem (7) if and only if \( \hat{\Theta} \) satisfies equation (8). Now, we know the necessary conditions of obtaining the solution, while how to calculate \( \hat{\Theta} \) in a more explicit form. First we need to define the equicorrelation set

\[
\varepsilon = \{i \in \{1, 2, \cdots, M\} : |P_{i}(y - \hat{P}\hat{\Theta})| = \lambda\}
\] (9)

Second, define the equicorrelation signs as

\[
s = \text{sign}(P_{\varepsilon}^T(y - \hat{P}\hat{\Theta}))
\] (10)

We use \( \hat{\Theta}_{-\varepsilon} = 0 \) for the solution \( \hat{\Theta} \) of problem (7) with the definition of the subgradient \( \gamma \). Therefore, we have

\[
P_{\varepsilon}^T(y - \hat{P}\hat{\Theta}_\varepsilon) = \lambda s
\] (11)

which can be expressed as

\[
P_{\varepsilon}^T P_{\varepsilon} \hat{\Theta} = P_{\varepsilon}^T(y - (P_{\varepsilon}^T)^+\lambda s)
\] (12)

Here \((P_{\varepsilon}^T)^+\) means the Moore-Penrose pseudoinverse of \( P_{\varepsilon}^T \). Due to \( P_{\varepsilon} \hat{\Theta}_\varepsilon = \hat{P}\hat{\Theta} \), then

\[
P\hat{\Theta} = P_{\varepsilon}(P_{\varepsilon})^+(y - (P_{\varepsilon}^T)^+\lambda s)
\] (13)
The solution \( \hat{\Theta} \) can be calculated, and the important nodes are selected and redundant neurons are dropped since \( \hat{\Theta}_\varepsilon \neq 0 \) and \( \hat{\Theta}_\varepsilon - \varepsilon = 0 \).

The Lasso solutions provide some unimportant hidden nodes dropped to obtain a simplified networks. The solution of lasso problem (7) may not be unique for the obtained fully connected networks. In other words, one may get different solutions \( \hat{\Theta} \) using different Lasso solvers. Consider a simple example, suppose there are 5 neurons, namely \( \hat{\Theta}_1 = (-0.8, 0.6, 0.3, 0.5, 0)^T \), \( \hat{\Theta}_2 = (-0.8, 0.7, 0.4, 0, 0)^T \). The index set of first solution is \( \{1, 2, 3, 4\} \) and that for the second solution is \( \{1, 2, 3\} \). It is not easy to know which solution is optimal or unique. Here, we provide the conditions where Lasso can have unique solution.

In [8], it is pointed that if the null space of \( P_\varepsilon \) is 0 or equivalently if the \( \text{rank}(P_\varepsilon) = |\varepsilon| \), then the solution of problem (7) is unique which is given as

\[
\hat{\Theta}_\varepsilon = 0 \quad \text{and} \quad \hat{\Theta}_\varepsilon = (P_\varepsilon^T P_\varepsilon)^{-1}(P_\varepsilon^T \mathbf{y} - \lambda \mathbf{s})
\] (14)

This unique solution is proved in the literature [8]. We pay attention to if \( \text{rank}(P_\varepsilon) = |\varepsilon| \) in our proposed algorithm so that the solution is unique.

**Proof:** \( \omega_i \) are generated randomly in the process of initialization before training the neural networks. In the minimization process, we apply SGD algorithms to iteratively update parameters by [9]

\[
W_t = W_{t-1} - \eta \nabla_W L(f(x(i); W), y(i))
\] (15)

where \( L(f(x(i); W), y(i)) = \frac{1}{2}(y(i) - \sum_{j=1}^{M} \beta_j \phi_j(w_j x(i) + b_j))^2 \) is the loss function and \( \nabla_W L(f(x(i); W), y(i)) \) represents the partial information of the loss function. \( f(x(i); W) \) is a nonlinear function with

\[
f(x(i); W) = \sum_{j=1}^{M} \beta_j \phi_j(w_j x(i) + b_j)
\] (16)
where $f(x(i); W)$ and $y(i)$ represent the predicted output and real output of the system, respectively. $W$ is the set of weights and biases with $W = \{ w, b \}$. It is worth pointing out that $W_{t-1}$ represents the set of previous parameters at the step $t - 1$ and $W_t$ is set of the updated parameters. Then we can assume that $\phi_j(w_jx(t) + b_j) \neq \phi_{j'}(w_{j'}x(t) + b_{j'})$ with any $j \neq j'$ ($1 \leq j \leq M$ and $1 \leq j' \leq M$), namely, $P_j \neq P_{j'}$ with any $j \neq j'$. Therefore, the condition for unique solution is that the column vectors of $P$ can be full-rank, namely, $\text{rank}(P_x) = \varepsilon$.

3.2. Update parameters of significant neurons

In the second stage of the proposed method, we update parameters including weights and biases with SGD method, which is a popular algorithm for training models in machine learning. When using SGD, the learning rate $\eta$ should be tuned carefully since a worse value may make the obtained model perform worse in new data. The main procedure of TSBE is given as follows:

**Algorithm 1**: TSBE

Stage 1: Drop redundant neurons with lasso
1: Initialization
2: for $i = 1, \ldots, S$ do
3: $W_t = W_{t-1} - \eta \nabla_W L(f(x(i); W), y(i))$
4: end for
5: Drop neurons: $\hat{\Theta}_\varepsilon = (P_x^TP_x)^{-1}(P_x^Ty - \lambda s)$ and $\hat{\Theta}_{-\varepsilon} = 0$

Stage 2: Update parameters ($W'$) of significant neurons
6: for $i = S + 1, \ldots, N$ do
7: $W'_t = W'_{t-1} - \eta \nabla_{W'} L(f(x(i); W'), y(i))$
8: end for

4. Simulation example

Consider the following nonlinear example:

$$y(t) = -0.8y(t - 2) + 0.65u(t - 1) - 0.05u(t - 3)y(t - 1) - 0.2u^3(t - 1) + e(t) \quad (17)$$

here $y(t)$, $u(t)$, $e(t)$ represent system output, input and noise. A uniformly distributed white noise is used to excite the nonlinear system with $u(t) \in [-1, 1]$. And Gaussian noise is used to disturb the system with signal-to-noise ratio $15dB$.

Here we select $\{u(t - 1), u(t - 2), u(t - 3), y(t - 1), y(t - 2), y(t - 3), y(t - 4)\}$ as the model input for identification. Generating $1800$ samples for identification with $80\%$ of total data being training data and the rest being test data. Suppose that the amount of initial neurons (size1) is $50$. Then, compare the proposed method with Dropneuron on the efficiency of model simplification, performance and training time.

| Method       | size1 | size2 | sparsity | train   | test    | time(s) |
|--------------|-------|-------|----------|---------|---------|---------|
| TSBE(0.001)  | 50    | 6.24  | 0.1448   | 0.0061  | 0.0077  | 148     |
| TSBE(0.005)  | 50    | 4.86  | 0.1172   | 0.0066  | 0.0082  | 118     |
| Dropneuron(0.1) | 50    | 5.76  | 0.0561   | 0.0402  | 0.0471  | 414     |
| Dropneuron(0.01) | 50   | 17.41 | 0.1905   | 0.0117  | 0.0134  | 413     |
| NN           | 50    | 50    | 1        | 0.0159  | 0.0182  | 181     |

5
In order to validate the efficiency of TSBE method, Monte Carlo simulation is carried out with 100 repetitions. In addition, in order to show the effect of parameter $\lambda$ on model reduction when using TSBE, two different values, namely, $\lambda = 0.001$ and $\lambda = 0.005$ are chosen. To make a fair comparison, the predetermined threshold for model reduction using Dropneuron are also given two values, namely 0.1 and 0.01. The training time, the average number of selected hidden neurons (size2) and sparsity that is defined as the amount of remained connections divided by the amount of full connections are used to compare the performance of TSBE, Dropneuron and NN. From Table 1, one can see that a large threshold can get a simple network when using Dropneuron method, but the accuracy is poor. In addition, although NN method has a similar performance with TSBE, the scale of networks is larger. Therefore, one can say that TSBE obtains a more parsimonious model with high accuracy. At the same time, the training time of the novel method is shorter than two other algorithms.

5. Conclusion

The neural networks with single hidden layer has been used to identify nonlinear dynamical systems and performs well in many practical applications. However, the amount of hidden neurons is often defined randomly, therefore, the size of network is often large and many hidden neurons of the complex network can be dropped. In this paper, the proposed two-stage backward elimination method can simplify neural networks using Lasso and can give a satisfied accuracy with fewer hidden neurons. We validate the proposed method with a nonlinear example and the higher level of prediction error and shorter training time shows that the novel TSBE method is effective and efficient.

References

[1] Lecun Y., Bengio Y. and Hinton G., Deeping learning. Nature, 2015, 521: 436-444.
[2] Han S., Pool J., Tran J. and Dally W., Learning both weights and connections for efficient neural network. In Advances in Neural Information Processing Systems, 2015, 1135-1143.
[3] Reed R., Pruning algorithms-a survey. IEEE Transactions on Neural Networks, 1993, 4(5): 740-747.
[4] Srivastava N., Hinton G., Krizhevsky A., Sutskever I. and Salakhutdinov R., Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 2014, 15: 1929-1958.
[5] Sainath T. N., Kingsbury B., Sindhwani V., Arisoy E. and Ramabhadran B., Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, 6655-6659.
[6] Pan W., Dong H. and Guo Y., DropNeuron: Simplifying the Structure of Deep Neural Networks. arXiv preprint arXiv:1606.07326, 2016.
[7] Scardapane S., Commiendiello D., Hassain A. and Uncini A., Group Sparse Regularization for Deep Neural Networks. Neurocomputing, 2017, 241: 81-89.
[8] Tibshirani R. J., The lasso problem and uniqueness. Electronic Journal of Statistics, 2013, 7(1): 1456-1490.
[9] Goodfellow I., Bengio Y. and Courville A., Deep learning. MIT press, 2016.