Network Complexity Analysis of Multilayer Feedforward Artificial Neural Networks

Helen Yu

Abstract. Artificial neural networks (NN) have been successfully applied to solve different problems in recent years, especially in the fields of pattern classification, system identification, and adaptive control. Unlike the traditional methods, the neural network based approach does not require a priori knowledge on the model of the unknown system and also has some other significant advantages, such as adaptive learning ability as well as nonlinear mapping ability. In general, the complexity of a neural network structure is measured by the number of free parameters in the network; that is, the number of neurons and the number and strength of connections between neurons (weights). Network complexity analysis plays an important role in the design and implementation of artificial neural networks - not only because the size of a neural network needs to be predetermined before it can be employed for any application, but also because this dimensionality may significantly affect the neural network learning and generalization ability. This chapter gives a general introduction on neural network complexity analysis. Different pruning algorithms for multi-layer feedforward neural networks are studied and computer simulation results are presented.

1 Introduction

There are two fundamental issues in neuro-computation: learning algorithm development and the network topology design. In fact, these two issues are closely related with each other. The learning ability of a neural network is not only a function of time (or training iterations), but also a function of the network structure.

A typical neural network contains an input layer, an output layer, and one or more hidden layers. The number of outputs and the number of inputs are usually fixed (note that in some applications, even the number of inputs can be changed — i.e., there may exist some inputs that are not actually related with the system dynamics and/or solution of the problem); while the number of hidden layers and number of hidden neurons in each hidden layer are parameters that can be specified for each application.

In a feedforward neural network, all the neurons are connected only in forward direction (Fig. 1). It is the class of neural networks that is used most often in system identification and adaptive control (1; 8; 19; 24; 26; 27). As we know from literature, a multi-layer feedforward neural network with correct value of weights (and appropriate transfer functions of the neurons) is capable of approximating any measurable continuous function to any degree of accuracy with only one hidden layer (but infinite number of hidden neurons) (18; 20; 25)]. However, in practice, it is impossible to construct such a neural network with infinite number of hidden nodes. On the other hand, none of the above results indicates how the neural network can be trained to generate the expected output and in fact, any existing learning law cannot guarantee this happening in a finite time period.

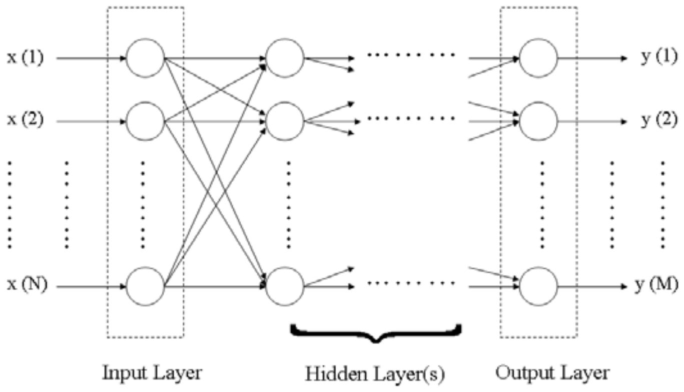

Fig. 1 Multilayer Feed-forward Network

In general, a larger neural network (i.e., more weights and hidden nodes) may yield a faster rate of convergence and is more powerful to solve problems (e.g., to recall patterns). On the other hand, a smaller neural network requires less computation time and is advantageous in real-time environment where speed is crucial. The neural network dimension is also related with the generalization property of a NN (2; 5; 7; 38). For a given set of training pairs, there exist many networks that can learn the underlying mapping function; however, the generalization capability of each network may be different. Generalization indicates the ability of a NN to produce accurate interpolation and/or prediction on data which are not included in the training set. If the structure of a NN is more complicated than necessary, over-fitting may occur. That is, the oversized NN may give exactly the right outputs on training samples, but fails on the other points between and/or beyond samples (Fig. 2). This is similar to the case in curve fitting problem when a high order polynomial is chosen while only a few points are available. Over-fitting is not an issue when a comprehensive training set is available, since all possible input/output pairs are presented and little or no generalization is needed. However, the amount of training data is usually limited; thus a trained network is expected to be able to perform well even on previously unseen data. Therefore, it is very important to find the "optimal", or "appropriate" size of a neural network for a specific application. An ideal neural network should be able to perform well on both training data and unknown testing data while maintaining its form as compact as possible.

An approach to determine the appropriate size of NN using Bayesian model comparison can be found in (40), where the neural networks with different hidden neurons or hidden layers are considered as a set of different models to be evaluated. It is based on the principle that the Bayesian formalism automatically penalizes the models with more complicated structures. A best model is the one which can balance the need of a large likelihood (so that it can fit well with data) and the need of a network with simpler structure.

Some recent research works investigate the optimal neural network problem using evolutionary computation (3; 14; 31; 35). As we know, a genetic algorithm is based on Charles Darwin's theory of natural selection (i.e., "survival of the fittest"). From Darwin's theory, individuals in a population of reproductive organisms inherit traits from their parents; and the desirable traits become more common than the undesirable ones during evolution, due to the fact that the individuals with the desirable traits which "fit" better with the environment are more likely to reproduce.

In a genetic algorithm (39), all the possible solutions are represented by genes which are usually coded as binary strings. As in natural selection process, an initial population of individuals is generated first, with all of their genes randomly selected. Each individual in the population is then sorted based on a certain performance criterion and those individuals with higher fitness levels (according to performance criteria) are more likely to be selected to reproduce the next generation of solutions that may yield a higher fitness index than the parents. The general reproduction operations involve crossover (i.e., exchange genes) and mutation (i.e., randomly change the value of a gene).

A genetic algorithm is a powerful global random search algorithm which finds the optimal solution in parameter space; however, the computation time may be very long.

Other than a lucky guess and/or an extensive search algorithm, there are two fundamental approaches to find the appropriate size of a neural network. The first one is to start with a small network and slowly add more connections to it until an appropriate stopping criterion is satisfied (11; 34). The

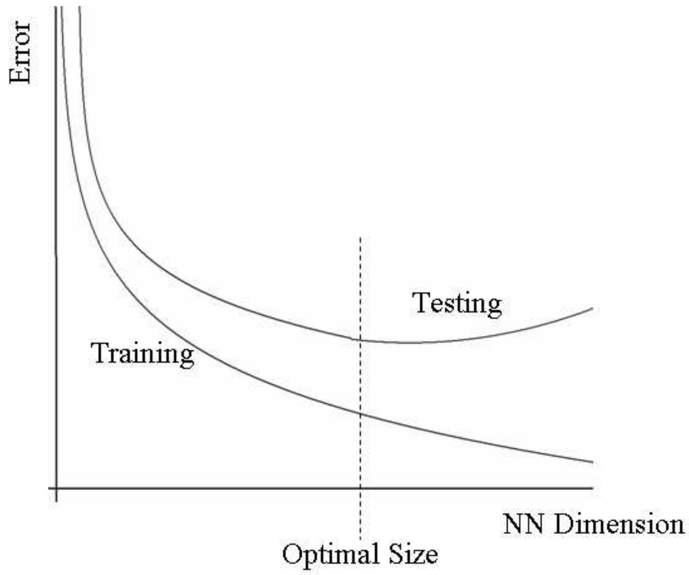

Fig. 2 Relationship between NN dimension, training error, and testing error

difficulties of this approach include when to start the growing process, and where to add the new connections/nodes in the network. In addition, the above procedures may be very tedious and time-consuming. The second approach is to start with a network that is knowingly too large for the data, and then trim it down to the appropriate size. This is called "neural network pruning" and has been studied by many researchers in recent years (12; 15; 16; 17; 29; 30; 32; 33). In this research, we focus on different pruning algorithms for multi-layer feedforward neural networks. Our goal is to reduce the size of a neural network to reduce its computation time while maintaining satisfactory generalization accuracy to improve the overall system performance.

2 Pruning Algorithms

Recent research on pruning algorithms may be classified into two major categories, i.e., the weight decay (or penalty-term) method and the sensitivity based approach. The penalty-term method adds an additional term to the cost function to be minimized so that the unnecessary weights in the network can be eliminated (i.e., driven to zero) during training (28; 36; 37). This penalty term is usually defined as the sum of the quadratic form of weights times a (or several) decay constant(s), as shown in Eq. (1):

$$J_0 = J + \frac{1}{2}\mu \sum_{i,j} w_{ij}^2 \tag{1}$$

where $J_0(\cdot)$ is the overall objective function; w_{ij} is the weight/connection of the neural network from node i to node j; μ is the parameter which controls the contribution (or importance) of the weight minimization in the overall objective function ($0 < \mu \leq 1$), and $J(\cdot)$ is the mean square error (MSE) function. For function approximation problem, $J(\cdot)$ can be defined as:

$$J = \frac{1}{L}\sum_{n=0}^{L-1}\sum_{m=1}^{M}\left[y_m(n) - y_m^{NN}(n)\right]^2 \tag{2}$$

where $y_m(n)$ represent the m-th output of the system at time (or training epoch) index n, $y_m^{NN}(n)$ represents the m-th output of the neural network, where $m = 1, 2, \ldots, M$; and L is the total number of training data if bench training is used.

The weight decay term in Eq. (1) can have different quadric forms. For example, an alternative penalty function is used in (9):

$$J_0 = J + \sum_i \epsilon_1 \left(\sum_i \frac{\beta w_i^2}{1 + \beta w_i^2}\right) + \epsilon_2 \left(\sum_i w_i^2\right) \tag{3}$$

where ϵ_1, ϵ_2 and β are constant coefficients; and w_i represents the connection (weight) vector to hidden unit i.

The drawback with the weight decay method is that it may penalize the weights that have significant influence on the neural network performance. Also, adding the extra penalty term may change the shape of error surface and thus create additional local minima.

Rather than focusing on the magnitude of the weights in the network, the sensitivity based approach attempts to find the contribution of each individual weight in the network and then removes the weights that have the least effect on the objective function. In other words, when the sensitivity of a particular weight is smaller than a pre-set threshold, this weight is insignificant and can be deleted. Furthermore, a neuron can be removed when the sensitivities of all the weights related with this neuron are below the threshold.

The sensitivity s_{ij} of a global error function with respect to a weight can be defined by measuring the difference on the performance of the network with vs. without that weight (12):

$$s_{ij} = J(w_{ij} = 0) - J(w_{ij} = w_{ij}^f) = J(without\ w_{ij}) - J(with\ w_{ij}) \tag{4}$$

where w_{ij}^f is the final value of weight w_{ij} after training. Note that one has to compute Eq. (4) for every weight which is a candidate for elimination throughout all training phases that can be extremely time-consuming for large networks. To simplify, rewrite Eq. (4) as:

$$s_{ij} = -\frac{J(w_{ij} = w_{ij}^f) - J(w_{ij} = 0)}{w_{ij}^f - 0} w_{ij}^f \tag{5}$$

assuming all the other weights (i.e., other than w$_{ij}$) are fixed (at their final states, upon completion of training). It is suggested that the numerator of Eq. (5) can be calculated as (10):

$$J(w_{ij} = w_{ij}^f) - J(w_{ij} = 0) \cong \int_A^F \frac{\partial J}{\partial w_{ij}}\, dw_{ij} \tag{6}$$

where F is the final state of all the weights on error surface, and A is the state/point when w$_{ij}$ =0 while all the other weights are in their final state.

In Eq. (4), the above integral is calculated along the error surface from a zero-state A to final state F. Note that a typical training process actually starts with some small initial values of NN weights (i.e., state I) which are not equal to (but close to) zero. Therefore, Eq. (6) can be approximated as:

$$J(w_{ij} = w_{ij}^f) - J(w_{ij} = 0) \cong \int_I^F \frac{\partial J}{\partial w_{ij}}\, dw_{ij} \tag{7}$$

This expression can be further approximated by replacing the integral by summation, taken over all the discrete steps (or training epochs) that the network passes while learning. Substituting Eq. (7) into Equations (5) and (6), the estimated sensitivity $\hat{s}_{ij}$ (with respect to the removal of connection w_{ij}) becomes:

$$\hat{s}_{ij} = -\sum_{n=0}^{N-1} \frac{\partial J}{\partial w_{ij}} [\Delta w_{ij}(n)] \frac{w_{ij}^f}{w_{ij}^f - w_{ij}^i} \tag{8}$$

where N is the total number of iterations (training epochs) needed to minimize Eq. (2), and w$_{ij}^i$ is the initial value of weight (before training).

For the back-propagation (steepest descent) algorithm (8), all the weights are updated as:

$$w_{ij}(n+1) = w_{ij}(n) + \Delta w_{ij}(n) \tag{9}$$

and

$$\Delta w_{ij}(n) = -\eta \frac{\partial J}{\partial w_{ij}}(n) \tag{10}$$

where η is the learning rate. Thus, Eq. (8) can be expressed as:

$$\hat{s}_{ij} = \sum_{n=0}^{N-1} [\Delta w_{ij}(n)]^2 \frac{w_{ij}^f}{\eta(w_{ij}^f - w_{ij}^i)} \tag{11}$$

For back-propagation with momentum (8), Eq. (10) becomes:

$$\varDelta w_{ij}(n) = -\eta \frac{\partial J}{\partial w_{ij}}(n) + \gamma\, \varDelta w_{ij}(n-1) \tag{12}$$

where γ is the momentum constant. Now the sensitivity calculation is slightly different:

$$\hat{s}_{ij} = \sum_{n=0}^{N-1} \left[\frac{\partial J}{\partial w_{ij}}(n)\right] \left[-\eta \frac{\partial J}{\partial w_{ij}}(n) + \gamma\, \varDelta w_{ij}(n-1)\right] \frac{w_{ij}^{f}}{w_{ij}^{f} - w_{ij}^{i}} \tag{13}$$

Eq. (11) and (13) indicate that the values of sensitivity are also related with the neural network adaptation law. For different training methods, the way for calculating the gradient $\frac{\partial J}{\partial w_{ij}}$ is the same; but the way for evaluating $\varDelta \mathrm{w}_{ij}$ is varied, depending on each individual algorithm. That means, even though we start with the same size of neural network, the same initial value for every weight, the sensitivity matrix may still be distinct if the neural network is trained by different training rules. An appropriate training algorithm can not only improve the training accuracy, but also result in a smaller network as well.

Once the neural network is trained to achieve the input/output mapping with desired accuracy based on certain performance criteria (such as the MSE value), the pruning algorithm can be activated to reduce the size of NN. If the absolute value of the sensitivity of a particular weight is greater than a specific threshold, i.e., the existence of this weight has greater impact on the error function, then it should be kept in the NN weight space; otherwise it can be eliminated:

$$w_{ij} = \begin{cases} 0 & \text{if } |\hat{s}_{ij}| < \tau \\ w_{ij} & \text{if } |\hat{s}_{ij}| \geq \tau \end{cases} \tag{14}$$

Since the partial derivative $\frac{\partial J}{\partial w_{ij}}$ (which is an element in gradient matrix) is always available during training, the only extra computational demand for implementing the pruning procedure is the summation in Eq. (11) (or Eq.(13)); thus is algorithm is easy to implement in practice.

One of the difficulties to implement the above algorithm is to find the appropriate value of threshold in Eq. (14). Ponnapelli et al. (13) suggested that the sensitivities of weights should only be compared with those related with the same node in the same layer. Thus, the concept of local relative sensitivity index (LRSI) is defined as the ratio of the sensitivity of a particular weight and the sum of all the sensitivities of the weights that are connected to the same node from the previous layer:

$$LRSI_{ij} = \frac{|s_{ij}|}{\sum\limits_{m=1}^{M} |s_{mj}|} \tag{15}$$

where M is the total number of connections to node j from the previous layer. For each node, any weight that has a local sensitivity less than a threshold will be pruned:

$$LRSI_{ij} \leq \beta \tag{16}$$

Even though the choice of the threshold (i.e., β) still depends on the rule of thumb, it is now a percentage which is relatively easier to be chosen ($0 \leq \beta \leq 1$), comparing with the selection of τ in Eq. (14).

The pruned NN is then retrained and its performance is evaluated. If the trained NN yields better results than the original network, the pruning procedure can be repeated and a smaller NN can be obtained.

A limitation of the above algorithm is that only weight removal is considered while node pruning is not included. Theoretically, if all the weights that are connected to a single node are pruned, then this node can also be eliminated. In practice, this may take several rounds of pruning and training so it may not be a feasible solution. Also, the network considered in (13) only contains one hidden layer.

Engelbrecht (6) proposed a modified approach to sensitivity analysis from the point of view of statistics. Instead of using the value of the sensitivity directly, Engelbrecht evaluated the average sensitivity of a network parameter (e.g., weight or node) over all the patterns, and then developed a new measure called variance nullity. That is, if the variance of sensitivity of a network parameter over all the patterns (denoted by $\sigma^2_{\theta_k}$ for parameter θ_k) is close to zero and the average sensitivity (also over all the patterns) is small, then we conclude that this parameter has little or no effect on the output of the neural network over all patterns and therefore can be eliminated. The variance of sensitivity is defined as:

$$\sigma^2_{\theta_k} = \frac{1}{P-1} \sum_{p=1}^{P} (s^{(p)}_{\theta_k} - \tilde{s}_{\theta_k})^2 \tag{17}$$

where P is the total number of patterns under consideration, $s^{(p)}_{\theta_k}$ is the sensitivity of θ_k for pattern p, and $\tilde{s}_{\theta_k}$ is the average sensitivity over all the patterns:

$$\tilde{s}_{\theta_k} = \frac{1}{P} \sum_{p=1}^{P} (s^{(p)}_{\theta_k} \tag{18}$$

The parameter variance nullity (PVN) for each parameter is then defined as:

$$\gamma_{\theta_k} = \frac{(P-1)\, \sigma^2_{\theta_k}}{\sigma^2_0} \tag{19}$$

where σ^2_0 is a small constant value related with hypothesis test $H : \sigma^2_{\theta_k} < \sigma^2_0$ (for details, see (6).

Starting from the output layer in a backward order, this algorithm allows the pruning of both nodes and weights in a similar manner layer by layer, with each parameter having a separate formula for the sensitivity calculation. The extension of a sensitivity measurement to nodes (not just weights) allows for the possibility of finding a smaller network, and also decreases the number of times to retrain the network before obtaining its final size. Unlike the algorithms discussed in (10; 13), this algorithm can be applied to NN with multiple hidden layers.

However, as we discussed earlier, relying on one single value of σ_0^2 for the entire network can lead to problems. Fnaiech et. al. (1) suggested that parameters within the same layer should be considered "locally" rather than "globally", and defined a new pruning index called the local parameter variance nullity (LPVN). The PVN for all parameters in the same layer are summed up; then the LPVN for each parameter (which represents the relative importance of PVN of a parameter in the layer) can be obtained and used for pruning:

$$L\gamma_{\theta_k^{[l]}} = \frac{\gamma_{\theta_k^{[l]}}}{\sum\limits_{k=1}^{K} \gamma_{\theta_k^{[l]}}} \tag{20}$$

where $L\gamma_{\theta_k^{[l]}}$ is the LPVN for layer l, and K is the total number of parameters in layer l. Note that in this algorithm, the pruning decision is still based on the hypothesis test H; thus choosing the appropriate threshold for LPVN is crucial to the success of this algorithm.

In practice, the number of weights eliminated also depends on when the pruning process starts. Longer training time before pruning may result in smaller neural network and complete training leads to the minimum neural network size. In addition, if training is continued after pruning, the accuracy of the neural network model can be further improved.

Huynh and Setiono (9) introduced a concept of pruning with cross-validation. The whole dataset is divided into two parts, i.e., the training set T and the cross validation set C. The pruning criterion is still based on the magnitude of each weight (see Eq. (3)); however, a validation step is added to test the pruned network. Pruning is performed on two separate phases, i.e., the weight removal phase and the hidden node removal phase.

At every weight pruning step, the performance of the network with reduced size is compared with the performance of the network before the current pruning phase. Let the performance criterion (objective function) on set T and set C be J_{TR} and J_{CV}, respectively. After pruning, a smaller neural network is obtained and the new performance criterion on set T and set C be J'_{TR} and J'_{CV}, respectively. If

$$(J'_{TR} + J'_{CV}) < (J_{TR} + J_{CV}) \tag{21}$$

i.e., the pruned network outperforms the unpruned one; then the pruned network is accepted and the pruning process can be continued. Otherwise, the network is restored to the size before the current pruning step. Similarly, in the node removal phase, if the removal of hidden node i has the least effect on the pruned network, then it can be pruned. Obviously, the use of an additional cross validation set at each phase of the pruning takes into account that pruning is meant to not only reduce the size of a network, but also improve the network generalization capacity.

As a summary, all the above algorithms have their own advantages and limitations. In the next section, we choose three typical pruning algorithms and compare their performances via computer simulations.

3 Computer Simulation Results

In this section, the performances of some typical pruning algorithms are studied and compared via computer simulations. The three pruning algorithm considered here include the local sensitivity analysis method (KLSA) (13), the local variance sensitivity analysis (LVSA) (1), and the cross validation pruning algorithm (CVP) (9) (for detailed description of each algorithm, see Eq. (15), (16), (20), and (21) in Sect. 2). All three algorithms are tested on the design of the neural network controller for a PS-ZVS-FB (Phase-shifted, zero-voltage switching, full-bridge) DC-DC voltage converter.

As we know, the DC-DC voltage converter plays an important role in many power electronics devices. It is desired that a DC-DC converter keeps its output stable even when the supply voltage and load current fluctuate over a wide range. Under these circumstances, the neural network controller becomes a promising solution based on its adaptive learning ability and nonlinear mapping ability. The output of the neural network, i.e., the control signal for the voltage regulator, is the duty cycle of a square waveform which is used to implement PWM (pulse-width modulation) switching control. The inputs to the neural network include input voltage, load current, and the absolute value of the change of output voltage $|\Delta V_0|$. For a detailed description of the DC voltage regulator and the neural network controller design, see (21),(22), and (23).

The simulation is written in C++ and the general block diagram is shown in Fig. 3. Different initial network configurations are considered to fully test the abilities of each pruning algorithm on a variety of hidden node and hidden layer setups. For this reason, we also extend the KLSA algorithm to the case of NN with more than one hidden layer. Configuration 1 has one hidden layer with fifteen hidden nodes; configuration 2 has two hidden layers with ten nodes in each hidden layer; and configuration 3 has three hidden layers with five nodes in each hidden layer. The data set contains about 1000 input/output data pairs. Similar to the ten-fold cross validation, the data set is divided into ten equal sub-sets; eight of them are used for training, one is used

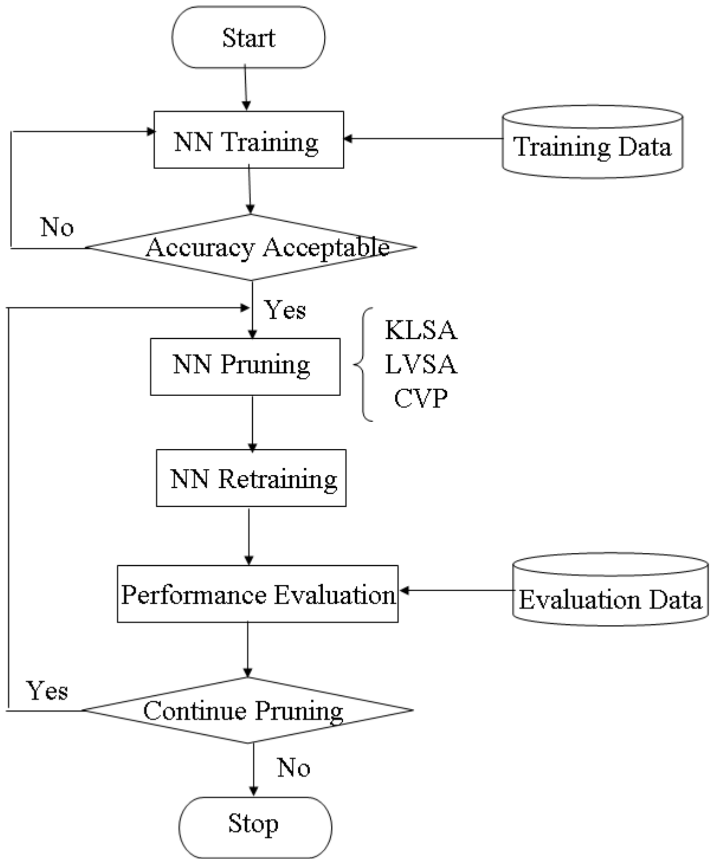

Fig. 3 Computer Simulation

for validation, and the remaining one is for testing. Each of the sub-dataset is used for training, validation, and testing on a rotation basis, resulting in a total of ten different data configurations. During training, all the data are presented in random order. Back-propagation with momentum is an effective training algorithm with fast convergence rate, moderate computational and memory requirement; thus it is chosen to train all the neural networks in this research.

The weights of all the neural networks are initialized to random values before training. The same initial conditions are applied to all the pruning algorithms in each test. To minimize the influences of initial conditions to the test results, ten different sets of initial conditions are chosen for each neural network configuration and each data configuration. Therefore, for each neural network configuration, a total of $10*10 = 100$ simulation runs are performed. This process is repeated for each of the four pruning algorithms.

Table 1 below shows the overall pruning capability of each of the four tested algorithms by displaying the mean and standard deviation (presented as (mean) ± (std)) of the pruning percentage of neural network weights (with respect to the original network). For example, for the first neural network configuration, the KLSA algorithm can prune about 13% of the total neural network weights (an average for 100 runs with different initial conditions and data rotations), with the standard deviation of 12%. Similarly, under the same condition, the CVP algorithm can prune about 34% of the total neural network weights, with a standard deviation of 17%. Obviously, the CVP algorithm outperforms the KLSA algorithm in this case; while the performance of LVSA algorithm is moderate. However, for neural networks with more than one hidden layers, the LVSA algorithm yields the best results. Note that in LVSA algorithm, pruning is based on the relative performance index (LPVN) of different weights that are connected to the same layer; which implies that those weights with smaller LPVN values in each layer can always be removed. However, in CVP algorithm when the network has more than one hidden layer, it may be difficult to relate the removal of the weights that do not connect directly to the output with the neural network performance, because

Table 1 A Comparison of Each Algorithm's Pruning (Percentages)

NN	KLSA	LVSA	CVP
1	13 ± 12	19 ± 13	34 ± 17
2	13 ± 12	23 ± 19	8 ± 21
3	6 ± 5	21 ± 14	5 ± 15

Table 2 The Detailed Results

	KLSA	LVSA	CVP
NN config. 1			
Training	0.0002 ± 0.0001	0.0002 ± 0.0001	0.0002 ± 0.00001
Accuracy	[0.0001, 0.0003]	[0.0001,0.0001]	[0.0001, 0.0003]
Testing	0.0002 ± 0.0001	0.0002 ± 0.0001	0.0002 ± 0.0001
Accuracy	[0.00001, 0.0004]	[0.0001, 0.0011]	[0.0001, 0.0003]
NN config. 2			
Training	0.0002±0.0001	0.0002 ± 0.0001	0.0003 ± 0.0001
Accuracy	[0.0001,0.0004]	[0.0001,0.0003]	[0.0001,0.0011]
Testing	0.0002 ± 0.0001	0.0002 ± 0.0001	0.0003 ± 0.0002
Accuracy	[0.0001,0.0004]	[0.0001,0.0003]	[0.00010,0.0012]
NN config. 3			
Training	0.0003 ± 0.0001	0.0002 ± 0.0001	0.0003 ± 0.0001
Accuracy	[0.0002,0.0005]	[0.0001,0.0004]	[0.0002,0.0007]
Testing	0.0002 ± 0.0001	0.0002 ± 0.0001	0.0003 ± 0.0001
Accuracy	[0.0001,0.0005]	[0.0001,0.0004]	[0.0001,0.0006]

each neuron in the network is a nonlinear MIMO (multi-input, multi-output) system that "interacts" with other neurons in the network.

Table 2 outlines the performance of each algorithm in terms of mean square error on both training and test dataset. The identification error gives a measure of the mean-square-error of the desired output and NN output. It is shown that the overall accuracies of the all three algorithms are similar.

4 Summary

Artificial neural networks have been widely used for many applications in recent years. Before a neural network can be employed, its size must be specified in advance. The purpose of network structure complexity analysis is to find the optimal dimension of a neural network for a specific application.

In this research, three different pruning algorithms are studied, namely, the local sensitivity analysis method (KLSA), the local variance sensitivity analysis (LVSA), and the cross validation pruning algorithm (CVP). The three algorithms are tested and compared on a neural network controller design for a DC-DC voltage converter with different initial network configurations. The simulation results show that the performance of LVSA algorithm is relatively uniform over different network configurations; while the CVP algorithm yields the best pruning result for the network with only one hidden layer. More tests will be conducted to fully evaluate the performances of these three algorithms.

References

1. Fnaiech, N., Abid, S., Fnaiech, F., Cheriet, M.: A modified version of a formal pruning algorithm based on local relative variance analysis. In: First International Symposium on Control, Communications and Signal Processing, pp. 849–852 (2004)
2. Rosin, P., Fierens, F.: Improving Neural Network Generalisation. In: International Geoscience and Remote Sensing Symposium, pp. 1255–1257 (1995)
3. Bevilacqua, V., Mastronardi, G., Menolascina, F., Pannarale, P., Pedone, A.: A novel multi-objective genetic algorithm approach to artificial neural network topology optimisation: the breast cancer classification problem. In: IEEE International Joint Conference on Neural Networks, pp. 1958–1965 (2006)
4. Narendra, K., Parthasarathy, K.: Identification and control of dynamical systems using neural networks. IEEE Transaction on Neural Networks 1(1), 4–27 (1990)
5. Lawrence, S., Giles, C., Tsoi, A.: Lessons in neural network training: overfitting may be harder than expected. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence, pp. 540–545 (1997)
6. Engelbrecht, A.: A new pruning heuristic based on variance analysis of sensitivity information. IEEE Transactions on Neural Networks 12(6), 1389–1399 (2001)

7. Giles, C., Lawrence, S.: Overfitting and Neural Networks: Conjugate Gradient and Backpropagation. In: Proceedings of the IEEE International Conference on Neural Networks, pp. 114–119 (2000)
8. Haykin, S.: Neural networks: a comprehensive foundation. Prentice Hall, New Jersey (1999)
9. Huynh, T., Setiono, R.: Effective neural network pruning using cross-validation. In: IEEE International Joint Conference on Neural Networks, pp. 972–977 (2005)
10. Karnin, E.: A simple procedure for pruning back-propagation trained neural networks. IEEE Transactions on Neural Networks 1(2), 239–242 (1990)
11. Marsland, S., Nehmzow, S., Shapiro, J.: A self-organizing network that grows when required. Neural Networks 15(8-9), 1041–1058 (2002)
12. Mozer, M., Smolensky, P.: Skeletonization: A technique for trimming the fat from a network via relevance assessment. In: Touretzky, D. (ed.) Advances in Neural Information Processing, pp. 107–115 (1989)
13. Ponnapalli, P., Ho, K., Thomson, M.: A formal selection and pruning algorithm for feedforward artificial neural network optimization. IEEE Transactions on Neural Networks 10(4), 964–968 (1999)
14. Yen, G., Lu, H.: Hierarchical genetic algorithm based neural network design. In: IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, pp. 168–175 (2002)
15. Chang, S.J., Leung, C.S., Wong, K.W., Sum, J.: A local training and pruning approach for neural networks. International Journal of Neural Networks 10(6), 425–438 (2000)
16. Chang, S.J., Sum, J., Wong, K.W., Leung, C.S.: Adaptive training and pruning in feedforward networks. Electronics Letters 37(2), 106–107 (2001)
17. Wan, W., Hirasawa, K., Hu, J., Jin, C.: A new method to prune the neural network. In: Proceedings of the IEEE International Conference on Neural Networks, pp. 449–454 (2000)
18. Neruda, R., Stedry, A., Drkosova, J.: Kolmogorov learning for feedforward networks. In: Proceedings of the IEEE International Conference on Neural Networks, pp. 77–81 (2001)
19. Kamran, F., Harley, R.G., Burton, B., Habetler, T.G., Brooke, M.A.: A fast on-line neural-network training algorithm for a rectifier regulator. IEEE Trans on Power Electronics 13(2), 366–371 (1998)
20. Hecht-Nielsen, R.: Kolmogorov's mapping neural network existence theorem. In: Proceedings of the IEEE International Conference on Neural Networks, pp. 11–14 (1987)
21. Li, W.: A neural network controller for a class of phase-shifted full-bridge DC-DC converter. PhD thesis, California Polytechnic State University, San Luis Obispo (2006)
22. Li, W., Yu, X.: Improving DC power supply efficiency with neural network controller. In: Proceedings of the IEEE International Conference on Control and Automation, pp. 1575–1580 (2007)
23. Li, W., Yu, X.: A self-tuning controller for real-time voltage regulation. In: Proceedings of the IEEE International Joint Conference on Neural Networks, pp. 2010–2014 (2007)

24. Quero, J.M., Carrasco, J.M., Franquelo, L.G.: Implementation of a neural controller for the series resonant converter. IEEE Trans on Industrial Electronics 49(3), 628–639 (2002)
25. Leshno, M., Lin, V., Pinkus, A., Shocken, S.: Multilayer feedforward networks with a non-polynomial activation function can approximate any function. Neural Networks 6, 861–867 (1993)
26. Lin, F., Ye, H.: Switched inductor two-quadrant DC-DC converter with neural network control. In: IEEE International Conference on Power Electronics and Drive Systems, pp. 1114–1119 (1999)
27. El-Sharkh, M.Y., Rahman, A., Alam, M.S.: Neural networks-based control of active and reactive power of a stand-alone PEM fuel cell power plant. Journal of Power Resources 135(1-2), 88–94 (2004)
28. Bebis, G., Georgiopoulo, M., Kasparis, T.: Coupling weight elimination with genetic algorithms to reduce network size and preserve generalization. Neurocomputing 17, 167–194 (1997)
29. Sabo, D.: A Modified Iterative Pruning Algorithm for Neural Network Dimension Analysis. PhD thesis, California Polytechnic State University, San Luis Obispo (2007)
30. Sabo, D., Yu, X.: A New Pruning Algorithm for Neural Network Dimension Analysis. In: Proceedings of the IEEE International Joint Conference on Neural Networks, pp. 3312–3317 (2008)
31. Kopel, A., Yu, X.: Optimize Neural Network Controller Design Using Genetic Algorithm. In: Proceedings of the World Congress on Intelligent Control and Automation, pp. 2012–2016 (2008)
32. Yu, X.: Reducing neural network size for dynamical system identification. In: Proceedings of the IASTED International Conference on Intelligent Systems and Control, pp. 328–333 (2000)
33. Yu, X.: Adaptive Neural Network Structure Based on Sensitivity Analysis. In: Proceedings of the World Forum on Smart Materials and Smart Structures Technology (2007)
34. Brouwer, R.: Automatic growing of a Hopfield style net-work during training for classification. Neural Networks 10(3), 529–537 (1997)
35. Vonk, E., Jain, L., Johnson, R.: Automatic generation of neural network architecture using evolutionary computation. World Scientific Publishing Co., Singapore (1997)
36. Huberman, B., Rumelhart, D.: Generalization by weight elimination with applications to forecasting. In: Lippmann, R., Moody, J. (eds.) Advances in neural information processing III, pp. 875–882. Morgan Kaufmann, San Francisco (1991)
37. Gupta, A., Lam, S.: Weight decay backpropagation for noisy data. Neural Networks 11, 1127–1137 (1998)
38. Reed, R., Marks, R.: Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks. MIT Press, Cambridge (1999)
39. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)
40. Bishop, C.: Neural Networks for Pattern Recognition. Oxford University Press, Oxford (1994)