A Decorrelation Approach for Pruning of Multilayer Perceptron Networks

Hazem M. Abbas

The German University in Cairo
Faculty of Media Engineering and Technology
New Cairo, Cairo, Egypt
hazem.abbas@guc.edu.eg,
http://met.guc.edu.eg

Abstract. In this paper, the architecture selection of a three–layer nonlinear feedforward network with linear output neurons and sigmoidal hidden neurons is carried out. In the proposed method, the conventional back propagation (BP) learning algorithm is used to train the network by minimizing the representation error. A new pruning algorithm employing statistical analysis can quantify the importance each hidden unit. This is accomplished by providing lateral connections among the neuron of the hidden layer and minimizing the variance of the hidden neurons. Variance minimization has resulted in decorrelated neurons and thus the learning rule for the lateral connections in the hidden layer becomes a variation of the anti-Hebbian learning. The decorrelation process minimizes any redundant information transferred among the hidden neurons and therefore enables the network to capture the statistical properties of the required input-output mapping using the minimum number of hidden nodes. Hidden nodes with least contribution to the error minimization at the output layer will be pruned. Experimental results show that the proposed pruning algorithm correctly prunes irrelevant hidden units.

Keywords: Neural Networks, Backpropagation Learning, Optimal Network Architecture, Pruning Algorithms, Statistical Learning.

1 Introduction

Finding an optimal architecture of feedforward neural networks is a very important issue for both classification and approximation problems. A small architecture will be unable to capture the internal representation required to perform the required input-output mapping. On the other hand, a large architecture will tend to over-fit the training data which leads to poor generalization capabilities of the designed network. It is therefore necessary to design the network with the smallest architecture and still can perform satisfactorily with unseen data. The generalization of a neural network architecture can be assessed by changing the the network size, i.e., the number of nodes and/or weights. There are many approaches to tackle the network design problem. Network construction algorithms starts with a small number of hidden nodes and then grows additional hidden

N. El Gayar et al. (Eds.): ANNPR 2014, LNAI 8774, pp. 12–22, 2014.

nodes and/or weights until a satisfactory design is found [1, 2, 3, 4]. Pruning algorithms start with a seemingly large network that is trained until an acceptable performance is achieved. Based on certain criteria, some hidden units or weights can be removed if they are considered useless [5, 6, 7, 8, 9, 10]. A recent survey on pruning methods can be found in [11] and older ones in [12, 13]. The third approach employs regularization techniques, which involves the addition of a penalty term to the objective function to be minimized [14, 15, 16, 17]. Algorithms that combine both constructive and pruning methods have been proposed in [18, 19].

The algorithm proposed in this work belongs to the pruning algorithms family. Normally, individual weights, hidden units and/or input units are the parameters that can be considered for removal. Two methods are normally employed to remove any of these candidate parameters: sensitivity analysis and role interpretation of the node. Sensitivity analysis techniques quantify the relevance of a network parameter as how important a slight deviation in a network parameter on the network performance [20, 21, 22]. The sensitivity measures could fail to identify possible correlations among nodes and possess high computational complexity. Node pruning techniques are post-training algorithms where the correlations among nodes in the hidden and output layers are exploited to decide which node to remove [23, 24, 9].

In this paper, a pruning method for a three-layer feedforward network is proposed. The method relies on reducing the variance of hidden layer nodes. Lateral connections among the hidden nodes are provided to accommodate such variance minimization. By doing so, the resulting updating rule will provide a kind of anti-Hebbian learning mechanism that will eventually lead to removal of nodes with least variance or contribution to the network mapping performance.

The paper is organized as follows. Section 2 introduces the proposed network architecture. The proposed decorrelational cost function and the learning rules required to train the network are presented in Section 3. Some necessary conditions on the nature of the node activation functions are discussed in Section 4. The training and pruning algorithm is detailed in Section 5. In Section 6, some simulation results of applying the proposed network and the conventional BP net to some benchmark problems are analyzed.

2 Network Structure

The neural network structure we are concerned with here is the three-layer network: an input layer, I, a hidden layer, H, and an output layer, O (Fig. 1). The neurons in the hidden layer have the sigmoid function while the output layer can have either sigmoidal or linear neurons. Neurons of the input layer can feed into neurons in the following layer through linking weights. Lateral connections are introduced in the hidden layer to decorrelate the output of its neurons. The net input to each node is the sum of the weighted outputs of the nodes feeding into this node.

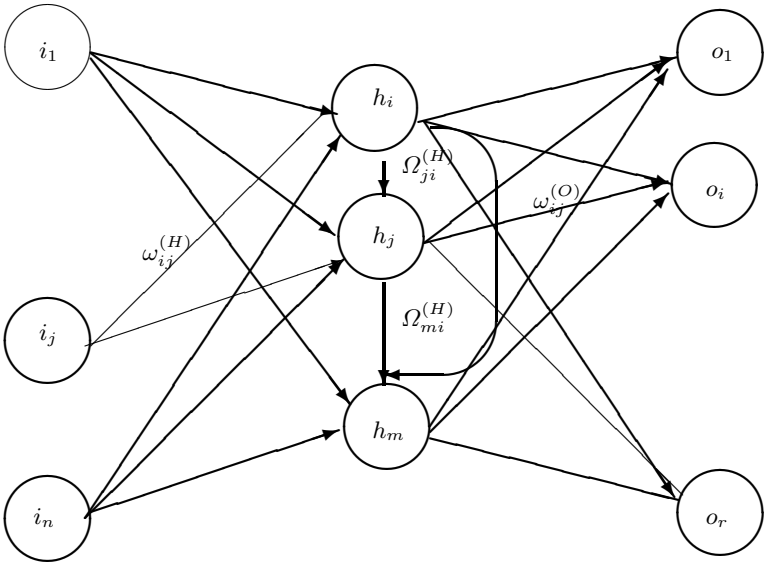

Fig. 1. The proposed network structure

The net input to the ith node in the output layer is calculated as

$$net_i^{(O)} = \sum_j^H \omega_{ij}^{(O)} a_i^{(H)} + \omega_{i0}^{(O)} \tag{1}$$

where $\omega_{ij}^{(O)}$ is the weight connecting the jth hidden node and the ith output node, $\omega_{i0}^{(O)}$ is the threshold (bias) of the node and $a_j^{(H)}$ is the output of the jth node in the hidden layer. Similarly, the net input to the jth node in the hidden layer is

$$net_j^{(H)} = \sum_i^I \omega_{ji}^{(H)} a_i^{(I)} + \omega_{j0}^{(H)} + \sum_{k<j}^H \Omega_{jk}^{(H)} a_k^{(H)} \tag{2}$$

where $\Omega_{jk}^{(H)}$ is a lateral connection from the kth hidden node to the jth hidden node. It should be noted that lateral connections in the hidden layer are only allowed from lower order to higher order nodes as indicated in the second summation. This is to ensure that the network is strictly feedforward. The output of sigmoidal node, j is

$$a_j^{(H)} = f\left(net_j\right) = \frac{1}{2}\left(\frac{1 - e^{-net_j}}{1 + e^{-net_j}}\right) \tag{3}$$

whereas the activation of the linear output nodes is simply

$$a_j^{(O)} = net_j.$$

3 Cost Functions and Training Method

Assuming that there are P distinct training patterns, each consisting of a pair of input and target patterns $\left\{\mathbf{a}^{(I,p)}, \mathbf{t}^{(O,p)}\right\}$. The network connections, $\{\omega^{(H)}, \omega^{(O)}, \Omega^{(H)}\}$, are to be adjusted in order to satisfy two objectives:

1. all the input/output pairs are mapped within a certain acceptable representation error
2. the variance of hidden nodes are minimized so that neurons with smallest influence on the representation error can be removed

Hence, the overall cost function, $\mathcal{J}$, that needs to be minimized can be expressed as follows:

$$\mathcal{J} = \mathcal{J}_o + \mathcal{J}_h \tag{4}$$

The mapping cost function, $\mathcal{J}_o$ is defined as

$$\mathcal{J}_o(\omega^{(H)}, \omega^{(O)}, \Omega^{(H)}) = \frac{1}{P}\sum_{p}^{P}\sum_{k}^{O}\left(t_k^{O,p} - a_k^{O,p}\right)^2 \tag{5}$$

where $t_k^{O,p}$ is the kth target component and $a_k^{O,p}$ is the output of the kth node of the output layer when the p pattern pair is presented. The hidden layer variance cost function, $\mathcal{J}_h$ is defined as

$$\mathcal{J}_h(\omega^{(H)}, \Omega^{(H)}) = \frac{1}{P}\sum_{p}^{P}\sum_{i}^{H}\left(a_i^{H,p} - \overline{a}_i^{H,p}\right)^2 \tag{6}$$

where $\overline{a}_i^{H,p}$ is the average value of hidden node i. The gradient descent learning rule is used to update the all weights of this network, i.e., the weight γ_{kj} is updated along the gradient direction using the equation

$$\Delta\gamma_{ij} = -\eta_\gamma \frac{\partial \mathcal{J}}{\partial \gamma_{ij}}$$

where $\gamma_{ij} \in \{\omega^{(H)}, \omega^{(O)}, \Omega^{(H)}\}$ and η_γ is a suitable learning rate that might differ with the type of γ_{kj}. It is evident that the backpropagation (BP) learning rules [25] can be applied to the three sets of weights when the $\mathcal{J}_o$ is minimized. Therefore, and dropping the variable p, the BP algorithm is applied as follows:

$$\Delta\gamma_{ij} = -\eta_\gamma \frac{\partial \mathcal{J}_o}{\partial \gamma_{ij}} = -\eta_\gamma \frac{\partial \mathcal{J}_o}{\partial net_i} \cdot \frac{\partial net_i}{\partial \gamma_{ij}} = \eta_\gamma\ \delta_i\ a_j \tag{7}$$

where η_γ is the learning rate and $\delta_i = -\frac{\partial \mathcal{J}_o}{\partial net_i}$ which is defined as

$$\delta_i = \begin{cases} \left(t_i^{(O)} - a_i^{(O)}\right) f'\left(net_i^{(O)}\right) & \text{if } i \text{ is an output node} \\ f'\left(net_i^{(H)}\right) \sum_l \delta_l^{(O)}\, \omega_{li}^{(O)} & \text{if } i \text{ is a hidden node.} \end{cases} \tag{8}$$

In addition, the two sets of weights, $\{\omega^{(H)}, \Omega^{(H)}\}$, will be also updated to minimize the hidden layer cost function, $\mathcal{J}_h$. Assuming that $\phi \in \{\omega^{(H)}, \Omega^{(H)}\}$, the update rule for ϕ is defined as

$$\Delta\phi_{ij} = -\eta_\phi \frac{\partial \mathcal{J}_h}{\partial \phi_{ij}} = -\eta_\phi \frac{\partial \mathcal{J}_h}{\partial a_i^{(H)}} \cdot \frac{\partial a_i^{(H)}}{\partial net_i^{(H)}} \cdot \frac{\partial net_i^{(H)}}{\partial \phi_{ij}} = -\eta_\phi \; a_i^{(H)} \; f'\left(net_i^{(H)}\right) \; a_j^{(H/I)} \quad (9)$$

Here, $a_j^{(H/I)}$ is either the output of an input node, $j \in I$ or a hidden node with $j \in H$, and $j < i$.

Although the BP rules (7,8) will reduce the system error to one of its local minima, it does not guarantee that the smallest number of hidden nodes will be employed in the representation task. Actually the error reduction task at the hidden layer is distributed over the all nodes and different nodes can contribute to the same task. Nothing in the BP rules imply that the correlation between the nodes is minimized. By inspecting the update rule, (9), one can observe that the weight update is proportional to the negative product of pre- and post-synaptic activations of the two nodes. This is a form of the anti-Hebbian learning rule [26] that is normally employed to decorrelate the output of two neurons. This rule has been employed in networks that extract the principal components of the presented data [27]. An anti-Hebbian learning rule can accomplish this decorrelation process. Hebb [28] had suggested that an excitatory connection between two neurons be strengthened if their activities are correlated and weakened otherwise. Similarly, an inhibitory connection will be strengthened if the activities were uncorrelated and weakened otherwise. This leads to the anti-Hebbian learning rule

$$\Delta\Omega_{ij}^{(H)} \propto -\beta \; a_i^{(H)} \; a_j^{(H)}. \quad (10)$$

Ideally, at convergence, when the average change in $\Omega_{ij}^{(H)}$ is zero, i.e., $E\{\Delta\Omega_{ij}^{(H)}\} \rightarrow 0$, the correlation, $E\{a_i^{(H)} \; a_j^{(H)}\}$, must vanish. This ensures that the hidden layer activations are orthogonal. The rule (10) differs from the conventional Hebb rule only in the sign before its learning rate, β.

4 Conditions on the Neuron Activations and Network Stability

The application of this decorrelation rule (9) dictates that the neurons are of a certain sigmoidal shape. Also, since the network employs lateral connections in the hidden layer, the stability of the performance should be investigated.

If the hidden layer were composed of sigmoidal neurons whose activation values take any real positive value in the range $[0, 1]$, then the condition (9) will be satisfied only when all hidden neurons activations are at zero level, i.e., the trivial solution of this function minimization. Hence, there will be no internal representation at the hidden layer and thus the system error, $\mathcal{J}$, will be at a very poor local minimum. This makes it necessary that the hidden nodes should produce both positive and negative values. Therefore, a bioploar sigmoid should be utilized.

The major problem with Hebbian learning is that the weights can grow indefinitely. This has led some researchers to modify the Hebb rule either by doing weight normalization [28] or by adding a forgetting factor [29]. Fortunately, the anti-Hebbian rule, (10), is stable without providing any modification and the weights, $\{\Omega\}$, will be bounded. This can be shown as follows. Assume that the weight Ω_{ij} increases, then using (2) and (3), the activation of node i, will increase. By virtue of the updating rule, (10), Ω_{ij} will decrease.

It should be also noted that decorrelation process performed at the hidden layer is similar to the Gram-Schmidt (GS) orthogonalization [30] applied to network pruning [31, 32]. However, there are many differences both in architecture and learning procedure. The GS nets generate a decorrelated version of the hidden layer output by introducing another layer after the hidden layer. The proposed architecture directly generates a decorrelated hidden layer output by simply adding and training lateral connections. Also, the GS orthogonalization is carried out using linear mapping. In this work, the anti-Hebbian decorrelation is embedded into the nonlinear operation of the sigmoid. Moreover, the plain BP rules cannot be applied directly with the GS nets due to the existence of the decorrelation layers.

5 The Training and Pruning Algorithm

The Algorithm (1) describes the procedure of training the network using the BP algorithm and how to select hidden neurons to remove from the hidden layer. First, the weights and biases of an $(n-m-r)$ three–layer network are initialized to small random values. Here, n, m and r stand for the number of neurons in the three layers, respectively. Then, the BP learning algorithm is applied to perform the training task. After every sweep of the whole set of data patterns, the error, $\mathcal{J}_o$ (7) , is calculated and compared to the error threshold, ϵ. If, $\mathcal{J}_o > \epsilon$, the training phase should be resumed till the error becomes less than ϵ.

At this point, the network is able to find the set of weights that can represent the required input-output mapping within a certain threshold. There are two possibilities regarding the hidden layer. The first is that all hidden neurons are fully utilized in mapping process. The other possibility is that the number of active hidden neurons could be reduced without (probably) violating the specified error threshold. A selection criterion needs to be applied to choose which neuron to remove or to become *inactive*. A reasonable criterion is *to excise the hidden node which has the least contribution in the error reduction process*. Alternatively, it is the node which, if removed, will result in the smallest error increase. A suitable way to perform this is to calculate the *average correlation coefficient*, $\overline{\rho_i}$, between the ith hidden neuron and the error at the output layer provided that the ith hidden neuron is not contributing to the output. The correlation coefficient, $\rho(i,j)$, between the output of ith hidden node, h_i, and the error element at jth output neuron, $e_i = t_i - o_i$, assuming that neuron i is inactive, is defined as:

$$\rho(i,j) = \frac{\mathrm{cov}\,(i,j)}{\sqrt{[\mathrm{var}(i)\,\mathrm{var}(j)]}} \tag{11}$$

Algorithm 1. Backpropagation Training and Pruning Algorithm

Initialize the network weights with small random value
while Minimum hidden nodes is not reached **do**
repeat
for Pattern $= 1 : P$ **do**
1. Present input pattern at the input layer
2. Starting from the input layer, use Eqns. (1,2,3) to compute the activities of the neurons at each layer.
3.Calculate the error, $\mathcal{J}_o$, at the output layer
4. Compute the variable, δ_i (8), for all nodes in the ouput and hidden layer, respectively.
5. Compute the change of weights of the three set of weights using

$$\Delta\omega_{ij}^{(O)} = \eta_\gamma \delta_i^{(O)}\, a_j^{(H)}$$

$$\Delta\omega_{ij}^{(H)} = \eta_\gamma \delta_i^{(H)}\, a_j^{(I)} - \eta_\phi\, a_i^{(H)} a'^{(H)}_i a_j^{(I)}$$

$$\Delta\Omega_{ij}^{(H)} = \eta_\gamma \delta_i^{(H)}\, a_j^{(H)} - \eta_\phi a_i^{(H)} a'^{(H)}_i\, a_j^{(I)}$$

6. Updates the weights
end for
Calculate Representation error, $\mathcal{J}_o$, after the update
until Representation error is within threshold, $\mathcal{J}_o < \epsilon$
Calculate the contribution of each hidden node in reducing $\mathcal{J}_o$
Remove the hidden node with the least contribution, $\overline{\rho_i}$ (12)
end while

where $\mathrm{cov}\,(i,j) = E\left\{([t_j - o_j] + \omega_{ji} h_i)\,(h_i)\right\}$, with $\mathrm{var}\,(i) = E\left\{h_i^2\right\}$ and $\mathrm{var}\,(j) = E\left\{([t_j - o_j] + \omega_{ji} h_i)^2\right\}$. The average correlation coefficient, $\overline{\rho_i}$, then becomes

$$\overline{\rho_i} = \frac{1}{n} \sum_{j=1}^{r} \rho(i,j). \tag{12}$$

The criterion guarantees that the neurons with maximal error reduction will be left intact, while improving the convergence by removing the less contributing neurons. It should be noted that the removal of hidden nodes requires a readjustment of the bias vector of the remaining nodes. Thus, an increase in $\mathcal{J}_o$ will be noticed for the first few sweeps after the excision. However, further training will fix this problem.

6 Simulation Results

In this section, we present some simulations of the conventional BP algorithm and the proposed model when they were applied to two benchmark problems: the XOR and the three-bit parity problems. For both models, a momentum term was added to the weight change formula, (7), i.e.,

$$\Delta\omega_{kj}(l) = \eta\, \delta_k(n)\, a_j(l) + \alpha\, \Delta\omega_{kj}(l-1)$$

where α is the momentum factor and l is the current iteration number. To compare the performance of the two networks, they had to be initialized to the same values. The parameters of each network were updated after every training epoch, a sweep of the presentation of the entire training set. The learning is considered complete when the performance index, $\mathcal{J}_o$, went below 0.0001.

6.1 The XOR Problem

Using a conventional BP algorithm,it was found that only two hidden nodes were needed to solve the XOR classification. Hence, the weights of a larger network with a 2–4–1 structure was initialized with small random values. The network is trained using the proposed BP Pruning algorithm. Learning rates were set to: $\eta_\gamma = 0.7$, $\alpha = 0.2$, and $\eta_\phi = 0.3$. The initial values of $\Omega_{ij}^{(H)}$ were set to zero in order to ensure the proposed network will have the same initial start as the BP network. The network was allowed to train till the error threshold was reached after 354 epochs. The calculated values of $\overline{\rho_i}, i = 1, \cdots, 4$ for the 4 hidden nodes were reported to be $+0.2260, +\mathbf{0.0032}, +0.5714$, and $+0.7889$, respectively. The correlation matrix, $C_H = E\{(\boldsymbol{h} - \bar{h})(\boldsymbol{h} - \bar{h})\}$, with $\boldsymbol{h}$ being the activation values of hidden layer nodes, is found to be

$$\begin{pmatrix} 0.0139 & 0 & 0 & 0 \\ +0.0001 & \mathbf{0.0000} & 0 & 0 \\ -0.0046 & -0.0004 & 0.0068 & 0 \\ -0.0015 & +0.0005 & -0.0036 & 0.0054 \end{pmatrix}$$

When the last step in Algorithm (1) is reached and decision about which node to remove, it is quite obvious that node 2 with variance equal to zero and minimum$\overline{\rho_2}$ is to be removed. . The matrix clearly shows that the hidden nodes were highly decorrelated. When the algorithm is resumed after removing the second hidden node, the target error was reached after further 26 epochs of training. The $\overline{\rho_i}$ values are $(\mathbf{0.2307}, 0, +0.6184, +0.7513)$ and

$$\mathbf{C}_H = \begin{pmatrix} 0.0139 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.0038 & 0 & 0.0069 & 0 \\ -0.0010 & 0 & -0.0043 & 0.0043 \end{pmatrix}$$

Here, node 1 will have the least effect on reducing the representation error and is chosen to be excised. After training for extra for extra 400 epochs, the reported values were $\overline{\rho_i} = (0, 0, +0.4317, +0.9020)$ and

$$\mathbf{C}_H = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0131 & 0 \\ 0 & 0 & +0.0035 & 0.0289 \end{pmatrix}$$

Evidently, the variance of the two remaining nodes have increased to account for the lost representation by the removed node while the anti-Hebbian sort of

learning kept the remaining two nodes largely uncorrelated. A very important observation has been reported when a conventional BP network is trained. While the BP network tends to almost equally distribute the internal representation task between the hidden nodes (nearly equal autocorrelation values), the proposed network, on the other hand, forced the hidden nodes to discover only the necessary features. This is represented by the unequal autocorrelation values in $\mathbf{C}_H$. It is worth mentioning that in implementing the algorithm, no matrix calculation is required. The computation of the correlation coefficients, $\rho(i, j)$, is carried out using local information and can be easily performed in a recursive fashion. The matrices, $\mathbf{C}_H$, shown above are for sole purpose of demonstrating the decorrelation capabilities of the proposed method.

6.2 The 3-Bit Parity Problem:

A 3-bit parity problem needs only a two hidden nodes to accomplish a 100% correct mapping. A 3-7-1 network was initialized and trained using the proposed BP pruning algorithm with the same learning parameters used with the XOR problem. After 1600 epochs, the correlation variable $\overline{\rho_i}$ had the following values for the 7 nodes: $+0.1428, +0.9469, +0.2721, +\mathbf{0.0019}, +0.0866, +0.0381, +\mathbf{0.0046}$. Clearly, nodes 7 and 10 have the least contribution and both have zero variance. Node 10 was chosen to be removed. At this point, both nodes could have been removed simultaneously, but selecting one node at a time will help to better understand the learning behaviour. As expected, further training for 100 epochs resulted in removing node 7. Correlation between the removed node and its predecessors are all null which suggests total independence between the nodes in solving the problem. Continuing with the training and pruning process, nodes 9, 8, and 5 were excised at epoch 1786, 2030, and 2155, respectively. The remaining two nodes had a very small correlation value of 0.0004.

The algorithm still needs to be tested with a larger classification data set such as the UCI Machine Learning database and be compared with other pruning algorithms mentioned in Section 1. Also, the generalization capabilities of the resulting network has to be assessed and compared with the a network of the same size that is trained with the conventional BP algorithm. Finally, the proposed method needs to be compared to a simple cross-validation approach in terms of complexity and performance, i.e., how the resulting optimal NN structure, trained on a subset of the data, will perform over the majority of the subsets.

7 Conclusions

In this work, a pruning algorithm for the hidden neurons of a three–layer network was investigated. The network has linear output neurons and bipolar sigmoidal hidden neurons. The algorithm works by providing lateral connection among the hidden nodes in such a way that the nodes are only connected to earlier ones in the same hidden layer. Minimizing the variance of hidden nodes resulted in

a learning rule that is of anti-Hebbian nature. This anti-Hebbian learning rule has been used to train these lateral connections to orthogonalize the outputs of all hidden nodes. The BP rules have been employed to train all forward connections. Test results indicate that the proposed method managed to find the optimal number of hidden nodes for both the XOR and 3-parity problems which fully decorrelating the hidden layer outputs.

References

[1] Ash, T.: Dynamic node creation in backpropagation networks. Connection Sciences 1, 365–375 (1989)
[2] Christian, S.F., Lebiere, C.: The cascade-correlation learning architecture. In: Advances in Neural Information Processing Systems 2, pp. 524–532. Morgan Kaufmann (1990)
[3] Yau Kwok, T., Yeung, D.Y.: Constructive algorithms for structure learning in feedforward neural networks for regression problems. IEEE Transactions on Neural Networks 8, 630–645 (1997)
[4] Platt, J.: A resource-allocating network for function interpolation. Neural Comput. 3(2), 213–225 (1991)
[5] Han, H.G., Qiao, J.F.: A structure optimisation algorithm for feedforward neural network construction. Neurocomput. 99, 347–357 (2013)
[6] Yau Kwok, T., Yeung, D.Y.: Constructive algorithms for structure learning in feedforward neural networks for regression problems. IEEE Transactions on Neural Networks 8, 630–645 (1997)
[7] Xu, J., Ho, D.W.: A new training and pruning algorithm based on node dependence and jacobian rank deficiency. Neurocomputing 70, 544–558 (2006)
[8] Engelbrecht, A.P.: A new pruning heuristic based on variance analysis of sensitivity information. Trans. Neur. Netw. 12(6), 1386–1399 (2001)
[9] Sietsma, J., Dow, R.J.F.: Creating artificial neural networks that generalize. Neural Network 4(1), 67–79 (1991)
[10] Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: Optimal brain surgeon. In: Advances in Neural Information Processing Systems 5, [NIPS Conference], pp. 164–171. Morgan Kaufmann Publishers Inc., San Francisco (1993)
[11] Augasta, M.G., Kathirvalavakumar, T.: Pruning algorithms of neural networks - a comparative study. Central Europ. J. Computer Science 3(3), 105–115 (2013)
[12] Reed, R.: Pruning algorithms-a survey. Trans. Neur. Netw. 4(5), 740–747 (1993)
[13] Castellano, G., Fanelli, A.M., Pelillo, M.: An iterative pruning algorithm for feedforward neural networks. Trans. Neur. Netw. 8(3), 519–531 (1997)
[14] Girosi, F., Jones, M., Poggio, T.: Regularization theory and neural networks architectures. Neural Comput. 7(2), 219–269 (1995)
[15] Schittenkopf, C., Deco, G., Brauer, W.: Two strategies to avoid overfitting in feedforward networks. Neural Networks 10(3), 505–516 (1997)
[16] Miller, D.A., Zurada, J.M.: A dynamical system perspective of structural learning with forgetting. Trans. Neur. Netw. 9(3), 508–515 (1998)
[17] Ishikawa, M.: Structural learning with forgetting. Neural Netw. 9(3), 509–521 (1996)
[18] Islam, M.M., Murase, K.: A new algorithm to design compact two-hidden-layer artificial neural networks. Neural Netw. 14(9), 1265–1278 (2001)

[19] Ma, L., Khorasani, K.: New training strategies for constructive neural networks with application to regression problems. Neural Netw. 17(4), 589–609 (2004)
[20] Cun, Y.L., Denker, J.S., Solla, S.A.: Advances in neural information processing systems 2, pp. 598–605. Morgan Kaufmann Publishers Inc., San Francisco (1990)
[21] Setiono, R.: A penalty-function approach for pruning feedforward neural networks. Neural Comput. 9(1), 185–204 (1997)
[22] Suzuki, K., Horiba, I., Sugie, N.: A simple neural network pruning algorithm with application to filter synthesis. Neural Process. Lett. 13(1), 43–53 (2001)
[23] Kanjilal, P., Dey, P., Banerjee, D.: Reduced-size neural networks through singular value decomposition and subset selection. Electronic Letters 17, 1515–1518 (1993)
[24] Fletcher, L., Katkovnik, V., Steffens, F., Engelbrecht, A.: Optimizing the number of hidden nodes of a feedforward artificial neural network. In: IEEE World Congress on Computational Intelligenc, pp. 1608–1612 (1998)
[25] Rumelhart, D., Hintont, G., Williams, R.: Learning representations by back-propagating errors. Nature 323(6088), 533–536 (1986)
[26] Carlson, A.: Anti-Hebbian learning in a non-linear neural network. Biol. Cybern. 64(2), 171–176 (1990)
[27] Chen, Z., Haykin, S., Eggermont, J.J., Becker, S.: Correlative Learning: A Basis for Brain and Adaptive Systems (Adaptive and Learning Systems for Signal Processing, Communications and Control Series). Wiley-Interscience (2007)
[28] Rubner, J., Tavan, P.: A self-organizing network for principal-component analysis. EPL (Europhysics Letters) 10(7), 693 (1989)
[29] Oja, E.: A simplified neuron model as a principal component analyzer. Journal of Mathematical Biology 15, 267–273 (1982)
[30] Cheney, W., Kincaid, D.R.: Linear Algebra: Theory and Applications, 1st edn. Jones and Bartlett Publishers, Inc., USA (2008)
[31] Maldonado, F., Manry, M.: Optimal pruning of feedforward neural networks based upon the schmidt procedure. In: The Thirty-Sixth Asilomar Conference on Signals, Systems and Computers, vol. 2 (1998)
[32] Orfanidis, S.: Gram-schmidt neural nets. Neural Computation 2, 116–126 (1990)