Exploring Architecture Variations in Constructive Cascade Networks

N.K. Treadgold and T.D. Gedeon
Department of Information Engineering
School of Computer Science & Engineering, The University of New South Wales
{nickt | tom}@cse.unsw.edu.au

Abstract

Constructive neural networks employing a cascade architecture face a number of problems. These include large propagation delays, high fan-in and irregular network connections. These problems are especially relevant with regards to VLSI implementation of these algorithms. This work explores the effect of limiting the depth of the cascades created by the CasPer algorithm, a constructive network algorithm. Instead of a single cascade of hidden neurons, a series of cascade towers are built. The maximum size of each tower is set prior to training, thus limiting maximum network depth, creating regular connections and enabling a reduction in maximum fan-in. The networks created in this manner are shown to maintain or better network generalization over a number of different tower sizes.

1. Introduction

The CasPer [1-3] algorithm has been shown to be a powerful method for training neural networks. CasPer is a constructive algorithm, which inserts hidden neurons one at a time to form a cascade architecture, similar to Cascade Correlation (CasCor) [4]. CasPer has been shown to produce networks with fewer hidden neurons than CasCor, while also improving the resulting network generalization, especially with regression tasks [2]. The reasons for CasPer's improved performance is that it does not use either CasCor's correlation measure, which can cause poor generalization performance [5], or weight freezing, which can lead to oversize networks [6].

A problem faced by constructive cascade algorithms such as CasPer is that the number of hidden neurons that are installed is potentially unlimited. Hence both the fan-in and propagation delay through cascade networks are inherently large, as noted early on by Fahlman [4]. In addition, the cascade structure results in irregular network connections. These features make VLSI implementation of such networks difficult. A further difficulty faced by the cascade architecture is that the number of weights added per hidden neuron is exponential. Figure 1 shows this relationship for a two input, single output cascade network. In order to overcome this exponential growth of weights, while also achieving a regularly connected architecture with limited network depth, the construction of a series of cascade towers is proposed. Each tower is a regular cascade of fixed depth. The weight increase is essentially linear for this type of architecture as shown in Figure 1 for tower sizes 4, 5 and 6.

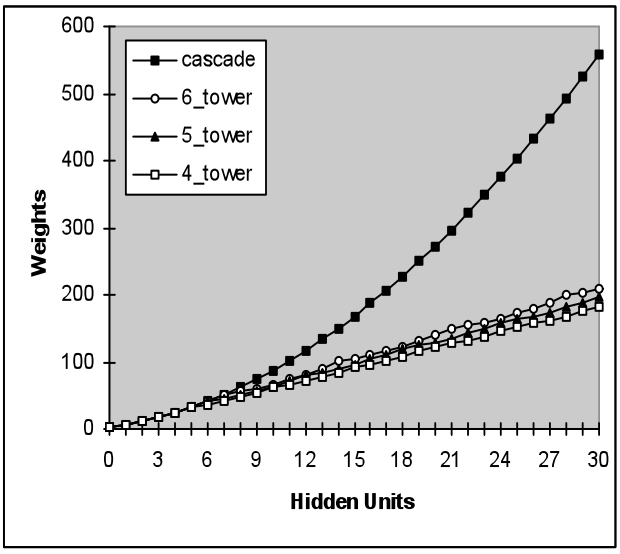

Figure 1. Networks weights vs. hidden units.

In this architecture each tower in the network can be viewed as a single Higher Order Neuron (HON), the complexity of which can be specified by the tower depth. Similar network architectures using a single layer of HONs are constructed by Projection Pursuit Learning [5,7], where the HON complexity is specified by the order of the Hermite polynomial used, and the Ridge Polynomial Network [8], which uses increasing orders of sigma-pi networks as its HON.

Previous work by Phatak and Koren [9] explored the effect of limiting network depth and fan-in with CasCor. This was done by constructing traditional layered topologies, however tradeoffs in performance compared to the traditional cascade architecture were observed.

2. The CasPer Algorithm

CasPer uses a modified version of the RPROP algorithm [10] for network training. RPROP is a gradient descent algorithm, which uses separate adaptive learning rates for each weight. Each weight begins with an initial learning rate, which is then adapted depending on the sign of the error gradient seen by the weight as it traverses the error surface. This results in the update value for each weight adaptively growing or shrinking as a result of the sign of the gradient seen by that weight.

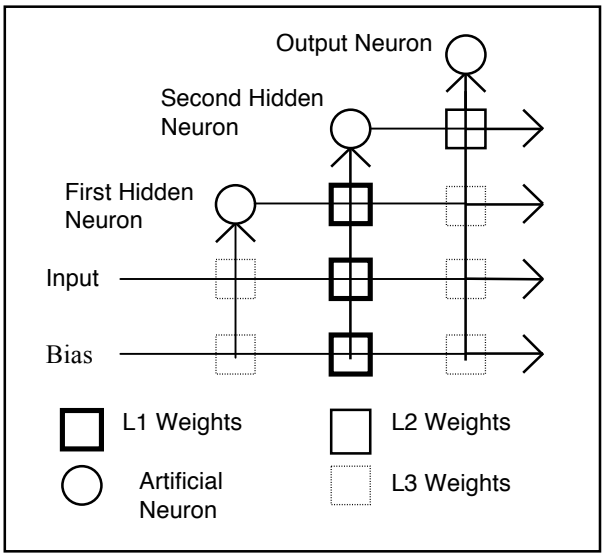

Figure 2. The CasPer architecture - a second hidden neuron has just been added. The vertical lines sum all incoming inputs.

The CasPer algorithm constructs cascade networks in a similar manner to CasCor: CasPer starts with all inputs connected directly to the outputs, and successively inserts hidden neurons to form a cascade architecture. RPROP is used to train the whole network each time a hidden neuron is added.

The use of RPROP is modified, however, such that when a new neuron is inserted, the initial learning rates for the weights in the network are reset to values that depend on the position of the weight in the network. The network is divided into three separate groups, each with its own initial learning rate: *L1*, *L2* and *L3* (Figure 2). The first group is made up of all weights connecting to the new neuron from previous hidden and input neurons. The second group consists of all weights connecting the output of the new neuron to the output neurons. The third group is made up of the remaining weights, which consist of all weights connected to, and coming from, the old hidden and input neurons.

The values of *L1*, *L2* and *L3* are set such that *L1*>>*L2*>*L3*. The reason for these settings is similar to the reason that CasCor uses the correlation measure: the high value of *L1* as compared to *L2* and *L3* allows the new hidden neuron to learn the remaining network error. Similarly, having *L2* larger than *L3* allows the new neuron to reduce the network error, without too much interference from other weights. Importantly, however, no weights are frozen, and hence if the network can gain benefit by modifying an old weight, this occurs, albeit at an initially slower rate than the weights connected to the new neuron.

In addition, the L1 weights are trained by a variation of RPROP termed SARPROP [11]. The SARPROP algorithm is based on RPROP, but uses a noise factor to enhance the ability of the network to escape from local minima. Noise is added to the update value of a weight when both the error gradient changes sign in successive epochs (indicating the presence of a minimum), and the magnitude of the update value is less than a threshold value. The amount of noise added falls as training continues via a Simulated Annealing (SA) term. This combination of techniques has been shown to improve RPROP's ability to escape local minima [11]. The equation used to modify the update value, Δ_{ij}, whenever a change in error gradient sign is encountered is:

$$\Delta_{ij}(t+1) = \Delta_{ij}(t)\,\eta^- + k_1 r S \quad \text{if } \Delta_{ij}(t) < k_2 * S$$
$$= \Delta_{ij}(t)\,\eta^- \quad \text{otherwise}$$

where r is a random number between 0 and 1, η^- is the standard RPROP update constant, k_1 and k_2 are constants, and S is the SA term which is set to $2^{-0.03*HEpoch}$. *HEpoch* refers to the number of epochs elapsed since the addition of the last hidden neuron

CasPer also makes use of weight decay as a means to improve the generalization properties of the constructed network. After some experimentation it was found that the addition of a Simulated Annealing term applied to the weight decay, as used in the SARPROP algorithm [11], often improved convergence and generalization. Each time a new hidden neuron is inserted, the weight decay begins with a large magnitude, which is then reduced by the SA term. The amount of weight decay is proportional to the square of the weight magnitude. This results in larger weights decaying more rapidly. This decay scheme is implemented by a modification to the error gradient used in CasPer. The new error gradient becomes:

$$\delta E/\delta w_{ij}^{CasPer} = \delta E/\delta w_{ij} - D * \text{sign}(w_{ij}) * w_{ij}^2 * 2^{-0.01*HEpoch}$$

where *sign* returns the sign (positive/negative) of its operand, and *D* is a user-defined parameter which effects the magnitude of weight decay used.

In Casper a new neuron is installed after the decrease of the RMS error has fallen below a set amount. The RMS error must fall by at least 1% of its previous value in a given time period. The time period over which this measure is taken is given by the parameter *L*.

3. Architecture Modifications to CasPer

In order to limit the network depth in CasPer it was decided to build a number of cascade towers, each of a fixed size set prior to training. Network training is performed in the standard manner. When the maximum cascade depth is reached, the next hidden neuron begins a new cascade. That is, it receives inputs from only the input units. The next hidden neuron continues the new cascade in the standard manner. When the training set has been learnt to satisfaction, training halts. The result of this training scheme is that a series of cascade towers will be built, all of the specified maximum size (except perhaps for the last tower which may be smaller than the others). An example of this architecture is shown in Figure 3.

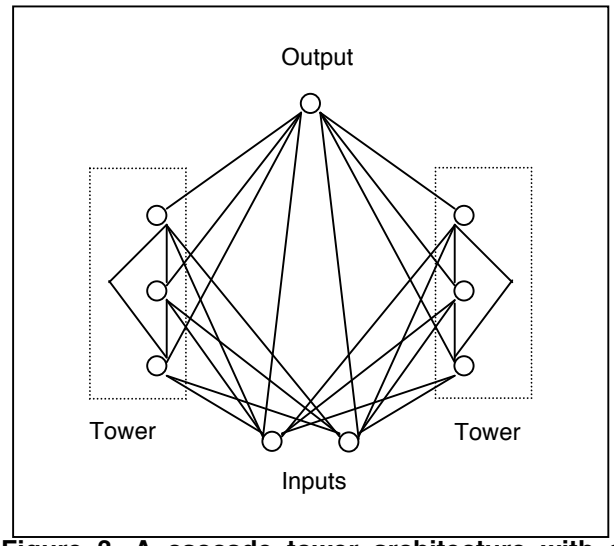

Figure 3. A cascade tower architecture with a tower size of 3.

The only modification to the standard CasPer training algorithm is the addition of a 'backfitting' training stage, which is done after the completion of each tower. It was found that the addition of backfitting improved the convergence of the algorithm significantly. The backfitting proceeds by training each tower in turn, starting with the first tower constructed, up to the most recent. The backfitting is achieved using RPROP with the initial learning parameters set as follows. All weights connected to the tower undergoing backfitting (including incoming, outgoing and internal weights) are assigned the initial learning rate T1. All other weights have initial learning rates T2. The values of these constants are set such that T1>>T2. This was done to maximize the ability of the tower undergoing backfitting to adapt to the remaining error, as in the original CasPer training methodology.

4. Comparative Simulations

To investigate the effects of the tower architecture, three benchmark problems were selected: the two spirals data set, and a regression data set with and without noise. For each data set training was repeated with a number of different tower sizes. The standard CasPer constant settings were used [2,3]. The constants T1 and T2 were set to 0.2 and 0.001 respectively.

4.1. Two Spirals Data Set

The first comparison was performed on the two spirals data set which consists of two interlocked spirals. Each spiral is made up of 97 points, which the network must learn to distinguish. Each training pattern consists of two inputs (the x,y coordinates) and a single output (the spiral classification). This problem was used by Fahlman [4] to demonstrate the effectiveness of the CasCor algorithm on a problem known to be very difficult for traditional Back Propagation to solve. The standard test set for the two spirals data set was used to measure the resulting generalization ability of the networks. This test set consists of two spirals each made up of 96 points, slightly rotated relative to the original spirals.

Fifty independent training runs were performed for tower sizes 5 to 11. The parameter values used for the CasPer algorithm were L = 100 and D = 0.01. The standard symmetric sigmoid non-linearity (-0.5, 0.5) was used as the neuron transfer function. Training was continued until the training set was learnt completely or a maximum of 50 hidden neurons were installed. At this point the mean, standard deviation and median for the following characteristics were measured: number of connection crossings, hidden neurons inserted, percentage correct on the test set, and the number of network weights. Fahlman [4] defines the term connection crossings as "the number of multiply-accumulate steps to propagate activation values forward through the network and error values backward". This measure of computational cost is used instead of epochs since the network sizes are continually changing. These results are shown in Table 1.

In all trials the training set was learnt before the maximum 50 hidden unit limit. Table 1 shows that the test set performance of the networks over many varying tower sizes are comparable to the original CasPer architecture, with often slight improvements obtained.

The training time and number of network weights installed are also comparable, although the results get progressively worse as the tower size is decreased. Further trials were performed using smaller tower sizes to explore this trend. These smaller sizes resulted in 30, 10 and 4 runs failing to converge using tower sizes of 2, 3 and 4 respectively. It is interesting to note the rather quick cut off in convergence performance. This decrease in performance as tower size drops can be attributed to the

reduced complexity of the HON implemented by the towers. The two spirals problem is highly non-linear and networks solving it benefit from the presence of more complex HONs.

Table 1. Two spirals results.

	CONN. CROSS. (10^8)	UNITS	TEST%	WEIGHTS
CasPer				
Average	1.12	11.64	98.38	
Median	1.03	11.00	98.96	102
Std. Dev.	0.44	2.34	1.73	
5_Tower				
Average	3.12	20.98	98.29	
Median	2.38	20.00	98.96	123
Std. Dev.	1.94	6.54	1.13	
6_Tower				
Average	2.35	17.92	98.44	
Median	1.88	18.00	98.96	120
Std. Dev.	1.64	4.74	1.44	
7_Tower				
Average	1.57	15.78	98.72	
Median	1.31	14.00	98.96	101
Std. Dev.	0.76	4.16	1.46	
8_Tower				
Average	1.46	14.96	98.81	
Median	1.3	15.00	98.96	112
Std. Dev.	0.71	3.11	1.45	
9_Tower				
Average	1.33	13.54	98.48	
Median	1.3	14.00	98.96	105
Std. Dev.	0.42	2.89	1.56	
10_Tower				
Average	1.33	12.82	98.76	
Median	1.27	13.00	98.96	103
Std. Dev.	0.58	3.06	1.24	
11_Tower				
Average	1.2	12.00	98.50	
Median	1.05	11.00	98.96	102
Std. Dev.	0.53	2.86	1.49	

4.2. Regression Data Sets

The next comparisons were performed using a regression data set with and without the presence of noise. The data set was generated from the complex interaction function (cif):

$$cif(x_1, x_2) = 1.3356(1.5(1 - x_1) + e^{2x_1 - 1}\sin(3\pi(x_1 - 0.6)^2) + e^{3(x_2 - 0.5)}\sin(4\pi(x_2 - 0.9)^2)).$$

The generation of training and test data follows the method of [5]: two sets of training data were created, one noiseless and one noisy, using 225 randomly selected pairs [0,1] of abscissa values $\{(x_1,x_2)\}$. The noisy data set was created by adding independent and identically distributed Gaussian noise, with zero mean and unit variance, giving an approximate signal to noise ratio of 4. An independent test set of size 2500 was generated on a regularly spaced grid $[0,1]^2$. Fifty independent training runs were performed for tower sizes 3 to 8. The parameter values used were L = 200 and D = 0.0001 and L = 200 and D = 0.001 for the noise free and noisy data sets respectively. The standard symmetric sigmoid non-linearity (-0.5, 0.5) was used, except for the output neuron, which used a linear transfer function.

The fraction of variance unexplained (FVU) was the measure chosen to compare the performances on the test set. The FVU is defined as:

$$FVU = \frac{\sum_{l=1}^{N}(\tilde{f}(x_l) - f(x_l))^2}{\sum_{l=1}^{N}(f(x_l) - \bar{f}(x_l))^2}$$

For each regression function 50 runs were performed using different random starting weight values. Training was continued until 30 hidden neurons had been inserted. The FVU on the test set was measured after the insertion of each hidden neuron. The median FVU values after each hidden neuron had been inserted is shown in Figures 4 and 5 for the noise free and noisy data sets respectively.

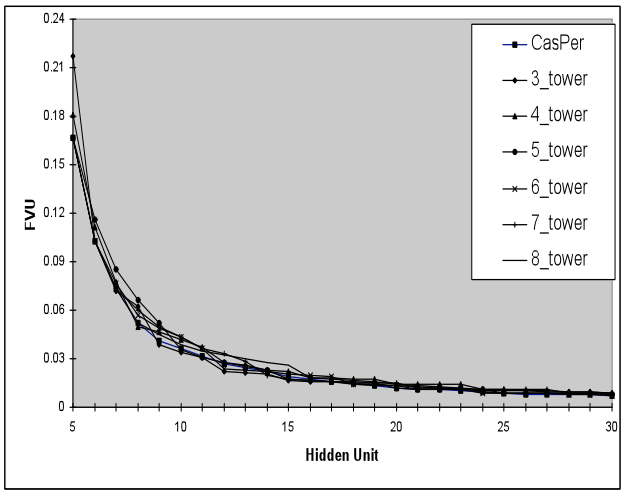

Figure 4. Complex interaction results - noise free.

Figure 4 shows that there is very little difference between different tower sizes and the original CasPer architecture in terms of results on the test set. The results on the noisy data set in Figure 5, however, show that the tower architectures do not suffer from overfitting to the extent that the original CasPer architecture does in the later stages of training. There are two reasons for this: first, the original CasPer architecture has an exponential growth in weights, and so as the number of hidden units installed becomes large, it has a much greater number of weights

than the tower architectures (Figure 1). This increases the chance of overfitting. The second reason, which is related to the first, is that the complexity of the HON implemented by the towers is limited by the tower size. In the original CasPer architecture, however, a single tower is constructed producing an increasingly complex HON that is more likely to overfit the data, especially in the presence of noise.

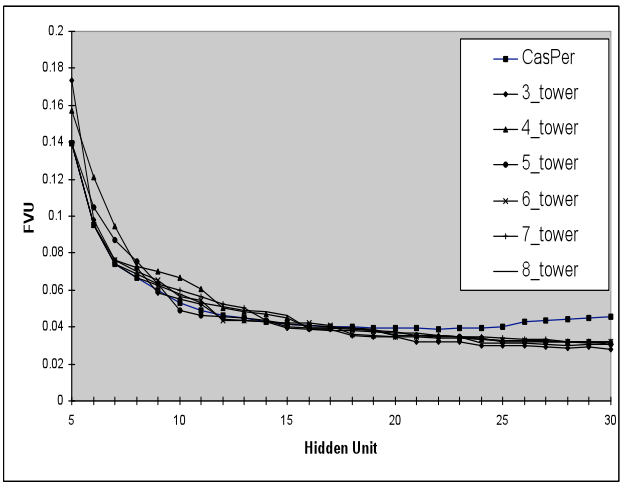

Figure 5. Complex interaction results – noisy.

Table 2. Average FVU on the cif data set.

NETWORK	CIF	CIF (NOISE)
CasPer	0.01298	0.03967
3_Tower	0.00827	0.02925
4_Tower	0.01005	0.03244
5_Tower	0.00801	0.03194
6_Tower	0.00820	0.03149
7_Tower	0.01047	0.03228
8_Tower	0.01110	0.03163
PPL	0.03818	0.09919
PPL_B	0.04013	0.08967

Previous work done by Kwok and Yeung [12] allows a comparison of these results with Projection Pursuit Learning (PPL), a powerful regression tool. Kwok and Yeung used the cif function, taking average FVU results obtained from 10 training runs. Their benchmarking was performed using both PPL and an extension to PPL involving the use of a bias neuron. Table 2 shows the best average FVU obtained by PPL and its extension (PPL_B) from a range of hidden neurons and Hermite function orders. Table 2 also shows the best average FVU for CasPer and various tower architectures. Both the original CasPer architecture and the tower architectures can be seen to produce better results on the test set than PPL.

5. Discussion

The introduction of the cascade tower architecture has a number of advantages. In terms of VLSI implementation, the tower architecture results in limited network depth, regular network connections and reductions in total network weights. In terms of network performance, the tower architecture is able to maintain and in some cases better network generalization, especially in the presence of noise.

One problem that is not addressed by the tower architecture is the high degree of fan-in, which can be a problem for VLSI implementation. The maximum fan-in using the tower architecture is exactly the same as in the original CasPer architecture for a given number of hidden units installed. The fan-in can be greatly reduced in the tower architecture, however, by introducing a new linear summing neuron in each tower. This neuron takes as its input the weighted sum of all tower outputs (which were originally connected to the output neurons) and its output is fed directly to the output neurons. This reduces the maximum fan-in from I+N to max(I+TS, I+N/TS), where I is the number of network inputs (including bias), N is the number of hidden neurons, TS is the tower size, and *max* returns the maximum of its two arguments. Table 3 shows the maximum fan-in obtained (both with and without the additional linear summing neuron) from the median network sizes obtained in the two spirals trials. These results show that the reduction in maximum fan-in obtained by the use of the linear summing neuron is significant, especially with larger networks.

Another problem faced by the tower architecture is that it introduces another parameter into the CasPer algorithm: the maximum tower size. This parameter needs to set be set prior to training (making a total of three parameters needing to be set, including the training length L and the decay parameter D). To remove the maximum tower size as a parameter, one option is to start with a maximum tower size of one, and increment this maximum size after the completion of each tower. This would reduce the chances of producing too complex a HON at an early stage. This strategy is used by Ridge Polynomial Networks [8], which start with a low degree sigma-pi network and increase the degree as training continues. Adapting this strategy to CasPer is the subject of future work.

Table 3. Maximum fan-in for the two spirals problem.

NETWORK	HIDDEN UNITS	FAN-IN (NORMAL)	FAN-IN (SUMMIMG)
CasPer	11	14	N/A
5_Tower	20	23	8

6_Tower	18	21	9
7_Tower	14	17	10
8_Tower	15	18	11
9_Tower	14	17	12
10_Tower	13	16	13
11_Tower	11	14	14

6. Conclusion

The introduction of the tower architecture to the CasPer algorithm has made CasPer more suitable to VLSI implementation, while maintaining and in some cases improving CasPer's good convergence and generalization properties.

7. References

[1] Treadgold, N.K. and Gedeon, T.D. "A Cascade Network Employing Progressive RPROP," *Int. Work Conf. On Artificial and Natural Neural Networks*, 1997, pp. 733-742.

[2] Treadgold, N.K. and Gedeon, T.D. "Extending CasPer: A Regression Survey," *Int. Conf. On Neural Information Processing, 1997,* to appear.

[3] Treadgold, N.K. and Gedeon, T.D. "Extending and Benchmarking the CasPer Algorithm," *AI'97, Perth Australia, 1997,* to appear.

[4] Fahlman, S.E. and Lebiere, C. "The cascade-correlation learning architecture," *Advances in Neural Information Processing*, vol. 2, D.S. Touretzky, (Ed.) San Mateo, CA: Morgan Kauffman, 1990, pp. 524-532.

[5] Hwang, J., You, S., Lay, S. and I. Jou, "The Cascade-Correlation Learning: A Projection Pursuit Learning Perspective," *IEEE Trans. Neural Networks* **7**(2), 1996, pp. 278-289.

[6] Kwok, T. and Yeung, D. "Experimental Analysis of Input Weight Freezing in Constructive Neural Networks," *Proc IEEE Int. Conf. On Neural Networks*, 1993, pp. 511-516.

[7] Hwang, J., Lay, S., Maechler, R. and Martin, D. "Regression Modeling in Back-Propagation and Projection Pursuit Learning," *IEEE Trans. Neural Networks* **5**(3), 1994, pp. 342-353.

[8] Shin, Y. and Ghost, J. "Ridge Polynomial Networks," *IEEE Trans. Neural Networks* **6**(2), 1995, pp. 610-622.

[9] Phatak, D.S. and Koren, I. "Connectivity and Performance Tradeoffs in the Cascade Correlation Learning Architecture," *IEEE Trans. Neural Networks* **5**(6), 1994, pp. 930-935.

[10] Riedmiller, M. and Braun, H. "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," *Proc IEEE Int. Conf. on Neural Networks*, 1993, pp. 586-591.

[11] Treadgold, N.K. and Gedeon, T.D. "A Simulated Annealing Enhancement to Resilient Backpropagation," *Proc. Int. Panel Conf. Soft and Intelligent Computing*, Budapest, 1996, pp. 293-298.

[12] Kwok, T. and Yeung, D. "Use of Bias Term in Projection Pursuit Learning Improves Approximation and Convergence Properties," *IEEE Trans. Neural Networks* **7**(5), 1996, pp. 1168-1183.