A Constructive Cascade Network with Adaptive Regularisation

N.K. Treadgold and T.D. Gedeon

Department of Information Engineering,
Computer Science and Engineering,
University of New South Wales,
Sydney, Australia
{nickt,tom}@cse.unsw.edu.au

Abstract. Determining the optimum amount of regularisation to obtain the best generalisation performance in feedforward neural networks is a difficult problem. This problem is addressed in the *casper* algorithm, a constructive cascade algorithm that uses regularisation. Previously the amount of regularisation used by this algorithm was set by a parameter prior to training. This work explores the use of an adaptive method to automatically set the amount of regularisation as the network is constructed. This technique is compared against the original method of user optimised regularisation settings and is shown to maintain, and sometimes improve the generalisation results, while also constructing smaller networks. Further benchmarking on the Proben1 series of data sets is performed and the results compared to an optimised Cascade Correlation algorithm.

1 Introduction

The *casper* algorithm [1, 2] has been shown to be a powerful method for training feedforward neural networks. It is a constructive algorithm that inserts hidden neurons one at a time to form a cascade architecture, similar to Cascade Correlation (*cascor*) [3]. The amount of regularisation in *casper* is set by a parameter. The optimal value for this parameter is difficult to estimate prior to training, and is generally obtained through trial and error. An inherent problem for the regularisation of constructive networks is that the number of weights in the network is continually changing, and thus even an optimal regularisation level for a given size network will become redundant as the network grows. This work explores the use of a method which adaptively sets the regularisation level as the network is constructed. This paper will first give an introduction to the *casper* algorithm, then describe the adaptive regularisation method and provide the results of some comparative simulations. Finally the algorithm is benchmarked on the Proben1 [4] series of data sets and its performance is compared to an optimised Cascade Correlation algorithm.

2 The *casper* Algorithm

The *casper* algorithm uses a version of the *RPROP* algorithm [5] 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.

Casper constructs a cascade network in a similar manner to *cascor*: it begins with all inputs connected directly to the outputs, and successively inserts neurons which receive inputs from all prior hidden neurons and inputs. *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 regions, each with its own initial learning rate: L1, L2 and L3. The first region is made up of all weights connecting to the new neuron from previous hidden and input neurons. The second region consists of all weights connecting the output of the new neuron to the output neurons. The third region 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* [6]. The *SARPROP* algorithm is based on *RPROP*, but uses a noise factor to enhance the ability of the network to escape from local minima.

In *casper* a new hidden neuron is installed after the decrease of the validation error has fallen below a set amount. All hidden neurons use a symmetric logistic activation function ranging between -0.5 and 0.5. The output neuron activation function depends on the type of analysis performed. Regression problems use a linear activation function. Classification tasks use the standard logistic function for single output classification tasks. For tasks with multiple outputs the softmax activation function [7] is used. Similarly, the error function selected depends on the problem. Regression problems use the standard sum of squares error function. Classification problems use the cross-entropy function [8]. For classification tasks, a *1-of-c* coding scheme for c classes is used, where the output for the class to be learnt is set to 1, and all other class outputs are set to 0. For a two class classification task, a single output is used with the values 1 and 0 representing

the two classes. For multiple classes a *winner-takes-all* strategy is used in which the output with the highest value designates the selected class.

The regularisation used in *casper* is implemented through a penalisation term added to the error function as shown below:

$$\tilde{E} = E + \frac{\lambda}{3} S \sum_{ij} |w_{ij}^3|$$

where λ sets the regularisation magnitude, and S is a Simulated Annealing (SA) term. The SA term reduces the amount of decay as training proceeds, and is reset each time a new neuron is added to the network.

3 Implementing Adaptive Regularisation

One method that would allow the amount of regularisation to change in constructive algorithms is to adapt this parameter as the network is trained. This was done using the following method as applied to the *casper* algorithm. The adaptation process relies on using three training stages for each new hidden neuron added, instead of the usual single training stage. The validation results taken after the completion of each training stage are then used to adapt the regularisation levels for future network training. This process repeats as the network is constructed.

For each new hidden neuron inserted into the network, three training stages are performed. Each training stage is performed using the same method as the *casper* algorithm, and is halted using the same criterion. The commencement of a new training stage results in all *RPROP* and SA parameters being reset to their initial values. Importantly, however, the final weights from the previous training stage are retained and act as the starting point for the next training stage. The motivation for this is that it is likely to increase convergence speed, and thereby construct smaller networks.

The regularisation level for the network once a new neuron is added is set to the initial value, λ_i, termed the initial decay value. This parameter takes the form $\lambda_i = 10^{-\alpha}$. It is this initial decay value that is adapted as the network is constructed. The first training stage uses the initial decay value. Each successive stage uses a regularisation level that has been reduced by a factor of ten from the previous stage. After each training stage the performance of the network on the validation set is measured, and the network weights recorded. On completion of the third training stage, the initial decay value is adapted as follows: if the best performing regularisation level occurred during the first two training stages, the initial decay value is increased by a factor of ten, else it is decreased by a factor of ten. At this point the weights that produced the best validation results are restored to the network. When the next neuron is added, the process repeats using the newly adapted initial decay value.

The initial network with no hidden neurons is trained using a single training stage with a regularisation level of $\alpha = 0$. The adaptation scheme begins with

the addition of the first hidden neuron, which is given an initial decay value of $\alpha = 2$. The initial decay value is chosen to give a relatively high regularisation level as this can easily be reduced through network growth and the adaptation process. The limits placed on the initial decay value are $\alpha = 1$ to 4, which gives a total possible regularisation range of $\alpha = 1$ to 6 (since there are three training stages). The lower initial decay limit ($\alpha = 4$) was selected to stop the regularisation level falling too low, which can occur in early stages of training when the network is still learning the general features of the data set. The top initial decay limit ($\alpha = 1$) was selected since convergence becomes difficult with excessive regularisation levels.

For reasons of efficiency, if the validation result of the second stage is worse than the first, the third training stage is not performed. In addition, if the validation results of the first training stage are worse than the best validation results of the previous network architecture, the weights are reset to their previous values before this training stage was commenced. The regularisation level is then reduced as normal, and the second training stage is started. This was done to stop excessive regularisation levels distorting past learning.

This regularisation selection method allows the network to adapt the level of regularisation as the network grows in size. The motivation for using this adaption scheme is the relationship between good regularisation levels in similar size networks. By finding a good regularisation level in a given network, it is likely that a slightly larger network will benefit from a similar regularisation level. The adaption process allows a good regularisation level to be found by modifying the window of regularisation magnitudes that are examined. This adaption process is biased towards selecting larger regularisation levels since the initial decay value is increased if either of the first two training stages have the best validation result. The reason for this bias is that as the network grows in size, in general more regularisation will be required.

The motivation for reducing the regularisation level through each training stage is that it allows the network to model the main features of the data set, which can then be refined by lowering the regularisation level. This is the same motivation for the use of the SA term in the regularisation function. The algorithm incorporating this adaptive regularisation method will be termed *acasper*. The parameter values for this algorithm were selected after some initial tuning on the Two spirals [3] and Complex interaction [9] data sets. Some tuning was also performed using the cancer1 data set from the Proben1 collection.

4 Comparative Simulations

In order to test the performance of *acasper* it was compared against *casper* on three regression and two classification benchmarks. The regression data sets are based on the Complex additive (Cadd), Complex interactive (Cif), and Harmonic (Harm) functions [9]. Each data set is made up of a training set of size 225 randomly selected points over the input space $[0, 1]^2$, a validation set of size 110 similarly generated, and a test set of size 10,000 generated by uniformly sampling

the grid $[0, 1]^2$. Gaussian noise of 0 mean and 0.25 standard deviation was added to the training and validation sets. The two classification benchmarks were the Glass and Thyroid data sets, which are glass1 and thyroid1 respectively from Proben1.

For each data set 50 training runs were performed for each algorithm using different initial starting weights. The Mann-Whitney U test [10] was used to compare results, with results significant to a 95% confidence level indicated in bold. Training in both *casper* and *acasper* is halted when either the validation error (measured after the installation of each hidden neuron) fails to decrease after the addition of 6 hidden neurons, or a maximum number of hidden neurons have been installed. This maximum was set to 8 and 30 for the classification and regression data sets respectively. The measure of computational cost used is connection crossings (CC) which Fahlman [3] defines as the number of multiply-accumulate steps required to propagate activation values forward through the network, and error values backward. This measure is more appropriate for constructive networks than the number of epochs trained since it takes into account varying network size.

The results on the test sets at the point where the best validation result occurred for the constructed networks after the halting criterion was satisfied are given in Tables 1 and 2. For the classification data sets this measure is the percentage of misclassified patterns, while for the regression data sets it is the Fraction of Variance Unexplained (FVU) [9], a measure proportional to total sum of squares error. Also reported is the number of hidden neurons installed at the point where the best validation result occurred, and the total number of connection crossings performed when the halting criterion was reached. The *casper* results reported are those that gave the best generalisation results from a range of regularisation levels: letting $\lambda = 10^{-\alpha}$, α was varied from 1 to 5.

Table 1. Comparative Results for the Classification Data Sets

Data Set	Algorithm	Property	Mean	StDv	Median	Min	Max
Glass	*casper*	Test Error %	**28.94**	2.34	28.30	26.42	33.96
		Hidden Neurons	**3.06**	2.06	3.00	0.00	8.00
		CC ($\times 10^8$)	**0.52**	0.00	0.52	0.52	0.52
Glass	*acasper*	Test Error %	30.68	2.61	30.19	26.42	35.85
		Hidden Neurons	4.18	2.21	4.00	1.00	8.00
		CC ($\times 10^8$)	1.33	0.09	1.32	1.11	1.50
Thyroid	*casper*	Test Error %	1.68	0.23	1.61	1.33	2.29
		Hidden Neurons	7.18	1.37	8.00	2.00	8.00
		CC ($\times 10^8$)	**25.71**	0.91	25.46	23.77	29.07
Thyroid	*acasper*	Test Error %	1.67	0.26	1.64	1.28	2.28
		Hidden Neurons	**4.64**	2.34	5.00	1.00	8.00
		CC ($\times 10^8$)	69.34	3.80	69.48	60.00	77.38

Table 2. Comparative Results for the Regression Data Sets

Data Set	Algorithm	Property	Mean	StDv	Median	Min	Max
Cadd	*casper*	Test FVU ($\times 10^{-2}$)	1.29	0.60	1.17	0.81	4.03
		Hidden Neurons	21.26	7.70	21.50	4.00	30.00
		CC ($\times 10^{8}$)	**11.93**	0.07	11.81	11.72	12.08
Cadd	*acasper*	Test FVU ($\times 10^{-2}$)	1.18	0.24	1.09	0.84	1.86
		Hidden Neurons	**16.16**	5.74	15.00	7.00	29.00
		CC ($\times 10^{8}$)	34.55	1.74	34.66	30.34	39.88
Cif	*casper*	Test FVU ($\times 10^{-2}$)	2.98	0.98	2.69	1.63	5.98
		Hidden Neurons	24.52	6.84	27.50	6.00	30.00
		CC ($\times 10^{8}$)	**12.09**	0.20	12.07	11.78	12.75
Cif	*acasper*	Test FVU ($\times 10^{-2}$)	**2.38**	0.61	2.21	1.48	4.03
		Hidden Neurons	**20.16**	6.24	19.50	8.00	30.00
		CC ($\times 10^{8}$)	34.27	1.91	34.24	28.61	38.77
Harm	*casper*	Test FVU ($\times 10^{-2}$)	3.12	0.95	2.89	1.59	5.46
		Hidden Neurons	26.00	5.71	29.50	12.00	30.00
		CC ($\times 10^{8}$)	**12.33**	0.37	12.28	11.82	13.78
Harm	*acasper*	Test FVU ($\times 10^{-2}$)	**2.37**	0.69	2.18	1.45	4.70
		Hidden Neurons	**19.34**	5.30	18.00	10.00	30.00
		CC ($\times 10^{8}$)	36.02	2.51	35.84	28.00	41.17

4.1 Discussion

In general the *acasper* algorithm is able to maintain or better the generalisation results obtained by the *casper* algorithm with a fixed, user optimised regularisation level. The only data set where *acasper* performs significantly worse is the Glass data set, although this reduction in performance is relatively small. The good performance of *acasper* can be attributed to its ability to adapt the regularisation level by taking into account such factors as the current network size and the presence of noise. Figure 1 demonstrates *acasper's* ability to adapt regularisation levels depending on the noise present in the data. This figure shows an example of the λ values selected by *acasper* on the Cif data set, with and without added noise, for a typical training run. The regularisation magnitudes selected for the noisy data set become greater as training proceeds, and are successful in preventing the network over-fitting the data.

In terms of the network size constructed, the *acasper* algorithm maintains, and often reduces the number of hidden neurons installed. The reduction is sometimes large, as can be seen for the regression tasks. This can be attributed to two factors. First, the *acasper* algorithm performs more training at each period of network construction. This takes the form of restarting training with different regularisation levels and with reset *RPROP* and SA parameters. This increases the chance of the network escaping from the current (possibly local) minimum and perhaps converging to a better solution. Second, the adaptation of the regularisation level may result in faster convergence in comparison to a fixed level.

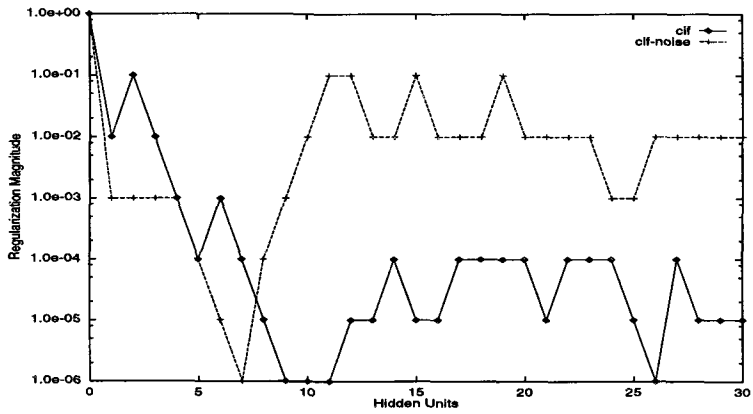

Fig. 1. Regularisation Magnitudes Selected by *acasper*.

The main disadvantage of the adaptive regularisation method used in *acasper* is the increase in computational cost. For the benchmark results obtained, this increase is of the order of two to three in comparison to *casper*. The increase in computational cost is expected to scale approximately linearly in comparison to corresponding size networks trained by *casper*, since it is a result of at most three additional training stages at each point of network construction. Part of the increased cost of training *acasper* is balanced by its ability to construct smaller networks than *casper*. The use of adaptive regularisation also removes the need to select a regularisation level in *casper*. The computational cost of such preliminary training is significant but not easily quantifiable, and not reflected in the results quoted for *casper*.

4.2 Benchmarking *acasper*

In order to allow comparison between the *acasper* algorithm and other neural network algorithms, an additional series of benchmarking was performed on the remaining data sets in the Proben1 collection. The same benchmarking set-up was used as for the previous comparisons, except the maximum network size for the regression problems was set to eight. The four regression data sets in Proben1 are building1, flare1, heartal, and heartac1. The test results for these data sets are given in terms of the squared error percentage as defined by Prechelt [4]:

$$E_{sep} = 100\frac{o_{max} - o_{min}}{N.c}E_{reg}$$

where o_{max} and o_{min} are the maximum and minimum values of the outputs, N is the number of training patterns, and c the number of training patterns.

To allow direct comparison with a well known constructive algorithm, the results obtained by the *cascor* algorithm [3] are also given. These results were

obtained from benchmarking carried out in [11]. This version of *cascor* incorporates a sophisticated implementation of early stopping. The results of these simulations are give in Tables 3 and 4 which give the test and hidden unit results respectively. Results which are significant to a 95% confidence level are printed in bold. At this level, the flare results in Table 3 were given as significantly different by the Mann-Whitney U test, however the test scores were found to have very different distributions, and hence this result was not treated as significant.

Table 3. Proben1 Benchmarking: Test Error Percentage

Data Set	Algorithm	Mean	StDv	Median	Min	Max
cancer1	*acasper*	1.89	0.80	1.72	0.57	4.02
	cascor	1.95	0.38	1.72	1.15	2.87
card1	*acasper*	13.72	0.59	13.37	13.37	16.86
	cascor	13.58	0.43	13.37	12.79	14.54
diabetes1	*acasper*	**23.14**	1.26	22.92	20.31	27.08
	cascor	24.53	1.44	24.48	22.40	28.65
gene1	*acasper*	**11.72**	0.09	11.73	11.60	11.85
	cascor	13.38	0.47	13.49	11.98	14.38
glass1	*acasper*	**30.68**	2.61	30.19	26.42	35.85
	cascor	34.76	5.88	33.96	26.42	47.17
heart1	*acasper*	**19.21**	0.44	19.13	16.96	20.00
	cascor	19.89	1.58	20.44	16.09	22.17
heartc1	*acasper*	**18.85**	1.14	18.67	18.67	26.67
	cascor	19.47	1.28	18.67	18.67	24.00
horse1	*acasper*	32.46	0.71	32.97	29.67	34.07
	cascor	**26.37**	2.58	26.37	20.88	31.87
soybean1	*acasper*	**7.89**	1.03	7.65	5.29	10.00
	cascor	9.46	0.86	9.41	7.65	11.77
thyroid1	*acasper*	**1.67**	0.26	1.64	1.28	2.28
	cascor	3.03	1.15	2.67	2.11	6.56
building1	*acasper*	**0.64**	0.02	0.64	0.61	0.71
	cascor	0.82	0.23	0.72	0.49	1.42
flare1	*acasper*	0.53	0.01	0.52	0.52	0.58
	cascor	0.53	0.01	0.53	0.51	0.55
hearta1	*acasper*	4.74	0.10	4.69	4.67	5.23
	cascor	**4.62**	0.15	4.60	4.43	5.02
heartac1	*acasper*	2.75	0.14	2.72	2.62	3.11
	cascor	2.87	0.44	2.70	2.48	4.25

It can be seen that *acasper* outperforms *cascor* both in terms of test results and constructing smaller networks. There are eight data sets where *acasper* obtains significantly better test results than *cascor*, compared to two where *cascor* outperforms *acasper* (four with no significant difference). For all data sets *acasper* was able to produce smaller networks than *cascor*, with significant re-

sults for twelve out of the fourteen data sets. There are some cases where the difference is surprisingly large, for example the Soybean and Thyroid data sets. One reason for this may be that the halting criteria for *acasper* specifies a maximum network size of eight, although in general this limit is rarely reached by *acasper* during the benchmarking.

Table 4. Proben1 Benchmarking: Hidden Units Used

Data Set	Algorithm	Mean	StDv	Median	Min	Max
cancer1	*acasper*	4.86	2.08	4.50	1.00	8.00
	cascor	5.18	2.05	4.00	3.00	10.00
card1	*acasper*	**0.12**	0.59	0.00	0.00	4.00
	cascor	1.07	0.25	1.00	1.00	2.00
diabetes1	*acasper*	**3.02**	1.55	3.00	1.00	8.00
	cascor	9.78	5.32	9.00	0.00	25.00
gene1	*acasper*	**0.00**	0.00	0.00	0.00	0.00
	cascor	2.73	1.19	2.00	1.00	6.00
glass1	*acasper*	**4.18**	2.21	4.00	1.00	8.00
	cascor	8.07	5.19	7.00	1.00	24.00
heart1	*acasper*	**0.10**	0.36	0.00	0.00	2.00
	cascor	2.64	1.17	2.00	1.00	7.00
heartc1	*acasper*	**0.10**	0.36	0.00	0.00	2.00
	cascor	1.38	0.49	1.00	1.00	2.00
horse1	*acasper*	**0.12**	0.59	0.00	0.00	4.00
	cascor	0.82	0.39	1.00	0.00	1.00
soybean1	*acasper*	**2.16**	1.08	2.00	1.00	5.00
	cascor	16.04	5.17	16.00	6.00	24.00
thyroid1	*acasper*	**4.64**	2.34	5.00	1.00	8.00
	cascor	25.04	8.71	27.00	2.00	44.00
building1	*acasper*	6.36	2.15	7.00	1.00	8.00
	cascor	9.27	9.73	6.00	0.00	29.00
flare1	*acasper*	**1.30**	1.59	1.00	0.00	6.00
	cascor	2.63	0.67	3.00	2.00	4.00
hearta1	*acasper*	**0.40**	0.57	0.00	0.00	2.00
	cascor	2.77	1.72	2.00	0.00	7.00
heartac1	*acasper*	**0.20**	0.86	0.00	0.00	5.00
	cascor	1.47	0.73	1.00	0.00	3.00

Interestingly, many of the data sets are solved by *acasper* using very small networks, often with no hidden units at all. This illustrates a major advantage of using constructive networks: the simple solutions are tried first. It is often the case that many real world data sets, such as the ones in Proben1, can be solved by relatively simple networks.

5 Conclusion

The introduction of an adaptive regularisation scheme to the *casper* algorithm is shown to maintain, and sometimes improve the generalisation results compared to a fixed, user optimised regularisation setting. In addition, smaller networks are generally constructed. In comparisons to an optimised version of *cascor*, *acasper* is shown to improve generalisation results and construct smaller networks. One further advantage of *acasper* is that it performs automatic model selection through automatic network construction and regularisation. This removes the need for the user to select these parameters, and in the process makes the *acasper* algorithm free of parameters which must be optimised prior to the commencement of training.

References

1. N. K. Treadgold and T. D. Gedeon, "A cascade network algorithm employing progressive rprop," in *Proc. of the Int. Work-Conf. on Artificial and Natural Neural Systems*, Lanzarote, Spain, June 1997, pp. 723–732.
2. N. K. Treadgold and T. D. Gedeon, "Extending casper: A regression survey," in *Proc. of the Int. Conf. on Neural Information Processing*, Dunedin, New Zealand, Nov. 1997, pp. 310–313.
3. S. E. Fahlman and C. Lebiere, "The Cascade-Correlation learning architecture," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.
4. L. Prechelt, "Proben1 - a set of neural network benchmark problems and benchmarking rules," Tech. Rep. 21/94, Fakultät für Informatik, Universität Kahlsruhe, 1994.
5. M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. of the IEEE Int. Conf. on Neural Networks*, San Francisco, CA, Apr. 1993, pp. 586–591.
6. N. K. Treadgold and T. D. Gedeon, "Simulated annealing and weight decay in adaptive learning: The sarprop algorithm," *IEEE Transactions on Neural Networks*, vol. 9, pp. 662–668, July 1998.
7. J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neuro-computing: Algorithms, Architectures and Applications*, F. Fogelman Soulié and J. Hérault, Eds. Berlin: Springer-Verlag, 1990, pp. 227–236.
8. C. Bishop, *Neural Networks for Pattern Recognition.* Oxford: Oxford University Press, 1995.
9. J.-N. Hwang, S.-R. Lay, M. Maechler, R. D. Martin, and J. Schimert, "Regression modeling in back-propagation and projection pursuit learning," *IEEE Transactions on Neural Networks*, vol. 5, pp. 342–353, May 1994.
10. R. Steel and J. Torrie, *Principles and Procedures of Statistics A Biomedical Approach.* Singapore: McGraw-Hill, 1980.
11. L. Prechelt, "Investigation of the cascor family of learning algorithms," *Neural Networks*, vol. 10, no. 5, pp. 885–896, 1997.