Increased Generalization through Selective Decay in a Constructive Cascade Network

N.K. Treadgold and T.D. Gedeon
Information Engineering, School of Computer Science & Engineering,
The University of New South Wales, Sydney, Australia.

ABSTRACT

Determining the optimum amount of regularization to obtain the best generalization performance in feedforward neural networks is a difficult problem, and is a form of the bias-variance dilemma. This problem is addressed in the CasPer algorithm, a constructive cascade algorithm that uses weight decay. Previously the amount of weight decay used by this algorithm was set by a parameter prior to training, often by trial and error. This is overcome through the use of a pool of neurons which are candidates for insertion into the network. Each neuron in the pool has an associated decay level, and the one which produces the best generalization on a validation set is added to the network. This not only removes the need for the user to select a decay value, but results in better generalization compared to networks with fixed, user optimized, decay values.

1. INTRODUCTION

The CasPer [1], [2] algorithm has been shown to be a powerful method for training neural networks. CasPer is a constructive algorithm that inserts hidden neurons one at a time to form a cascade architecture, similar to Cascade Correlation (CasCor) [3]. 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 [4], or weight freezing, which can lead to oversize networks [5].

Regularization in CasPer is performed through the use of weight decay, the magnitude of which is set by a parameter. The optimal value for this parameter in terms of network generalization is difficult to estimate prior to training, and is generally obtained through trial and error. An inherent problem for the regularization of constructive networks is that the number of weights in the network is continually changing, and thus an optimal decay rate for a given size network will become sub-optimal as the network grows.

This work explores the use of a pool of neurons which are used as candidates for insertion into the network. Each neuron in the pool has an associated decay level which is applied to the whole network. The network using the pool neuron that produces the best generalization results on a validation set is selected for continued training. This process removes the need for the user to find a value for the decay parameter. The decay level is also able to adapt to produce the best performance as the structure of the network is modified. This automatic decay selection is applied to two versions of the CasPer algorithm: the first version constructs the standard cascade architecture and will be termed A_CasPer. The second version modifies the architecture produced so that a series of cascade towers is constructed [6]. This version will termed AT_CasPer.

This paper will first outline the original CasPer algorithm, and then the modifications used to create A_CasPer and AT_CasPer. The results from a series regression benchmarks are then reported and discussed.

2. THE CASPER ALGORITHM

CasPer uses a modified version of the RPROP algorithm [7] 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.

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 1). 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

0-7803-4778-1/98 $10.00 © 1998 IEEE

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.

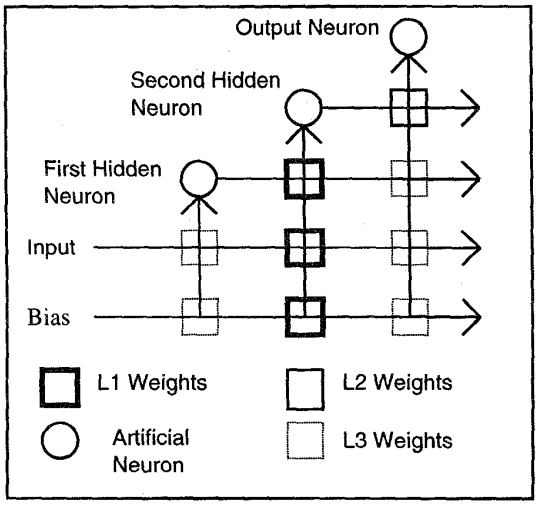

Fig. 1. The CasPer architecture - a second hidden neuron has just been added. The vertical lines sum all incoming inputs.

In addition, the L1 weights are trained by a variation of RPROP termed SARPROP [8]. 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 neuron is installed after the decrease of the RMS error has fallen below a set amount.

The weight decay used in CasPer is implemented through a penalization term added to the standard sum of squares error function. This results in a modification to the error gradient which is shown below, where λ is the decay parameter, 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.

$$\delta E/\delta w_{ij}^{CasPer} = \delta E/\delta w_{ij} + \lambda * |w_{ij}| * w_{ij} * S$$

where $S = 2^{-0.01*Hepoch}$ and *Hepoch* = number of epochs since last neuron was added.

3. IMPLEMENTING SELECTIVE DECAY

The modifications required to the CasPer algorithm to obtain A_CasPer are as follows. First, instead of a single neuron being trained, a pool of neurons is trained, each which has an associated network decay level. Each neuron in the pool is successively connected to the network in the normal manner. Training is continued on this new network until the normal halting condition, at which point the performance on the validation set is measured. The weights and learning parameters of the original network are then reset to their values prior to training. The next neuron in the pool replaces the previously inserted neuron, and the process is repeated. Finally, the network with the best generalization performance is chosen, and its weights reinstalled. A new pool is then generated and the process is repeated until some convergence criterion is satisfied.

A different decay level, λ, is used for the network each time a new neuron in the pool is inserted. In addition, the initial weights connecting each pool neuron to the network are chosen to be small random values in the range [-0.1, 0.1]. A pool of size 6 was found to provide a sufficient range of decay values, which consist of 10^{-1}, 10^{-2} ... 10^{-5}.

The obvious disadvantage to this method of selective decay is the increased computational cost which results from having to retrain the whole network for each neuron in the pool. In order to reduce this computational cost, the CasPer algorithm is modified to construct a series of cascade towers, each of fixed depth [6]. Network construction proceeds as normal, except that once the maximum tower size is reached, a new cascade is begun (that is, the next hidden neuron is connected only to the inputs and bias neurons). This scheme results in a series of cascade towers being constructed, one after the other. The maximum tower size was selected to be eight neurons. The combination of selective decay and the tower architecture will be termed AT_CasPer.

There are a number of advantages in using the architecture of AT_CasPer. The main one is in terms of the number of weights used by the network. The CasPer algorithm adds exponentially more weights per hidden neuron as the network is constructed due to the cascade architecture. AT_CasPer, however, is able to bound the growth of weights added by a linear function [6]. The effect of this architecture in terms of weight reduction is shown in Figure 2. By limiting the number of weights in the network, the computational cost of training a pool of neurons is reduced. Further advantages of the tower architecture are in terms of VLSI implementation. The tower architecture provides fixed maximum network depth and propagation delay, a modular architecture, and through the use of a linear summing

node at the top of each tower, reduced maximum network fan-in [6].

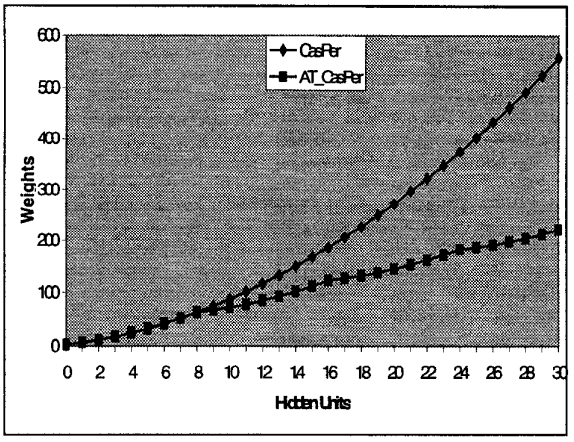

Fig. 2. Networks weights in a given size network

4. COMPARATIVE SIMULATIONS

To investigate the performance of the selective decay, three regression benchmark problems were selected. Comparisons were made between the original CasPer algorithm and A_CasPer and AT_CasPer. The standard CasPer constant settings were used [2] for CasPer, A_CasPer, and AT_CasPer. For CasPer the decay parameter, λ, was set to give the best performance on the given data set.

The regression functions chosen for the comparison are described in detail in [9], and are shown below:

- Radial function:

$$f^{rad}(x_1, x_2) = 24.234\,(r^2(0.75 - r^2)),$$
$$r^2 = (x_1 - 0.5)^2 + (x_2 - 0.5)^2$$

- Complex additive function

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

- Harmonic function

$$f^{harm}(x_1, x_2) = 42.659\,((2 + x_1)/20 + \mathrm{Re}(z^5)),$$
$$z = x_1 + ix_2 - 0.5(1 + i),$$

The set up of training and test data follows the method of [9]. For each function two sets of training data were created, one noiseless and one noisy, created using 225 randomly selected pairs [0,1] of abscissa values $\{(x_1,x_2)\}$. The same abscissa values were used for all three functions. The noisy data was created by adding independent and identically distributed Gaussian noise, with zero mean and variance 0.065, giving an approximate signal to noise ratio of 4 [9]. For the A_CasPer and AT_CasPer algorithms a validation set of 110 randomly selected points was chosen (which included noise for the noisy data sets). For each function an independent test set of size 10000 was generated on a regularly spaced grid $[0,1]^2$, as used in [9].

The fraction of variance unexplained (FVU) was the measure chosen to compare the performances on the test set [9]. FVU is proportional to the total sum of squares error. For each function 50 runs were performed using different random starting weight values. Training was continued until 30 hidden units had been added. The FVU on the test set was measured after the installation of each hidden unit and the median values are plotted in Figures 3 to 8.

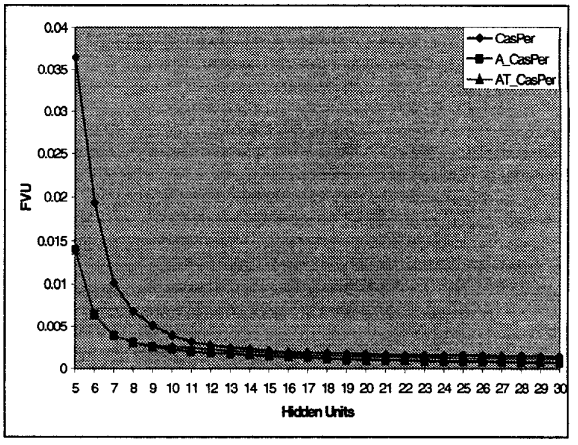

Fig. 3. Radial results – noise free.

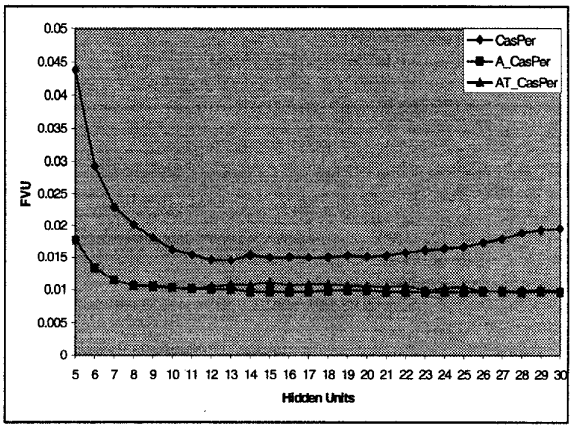

Fig. 4. Radial results - noisy.

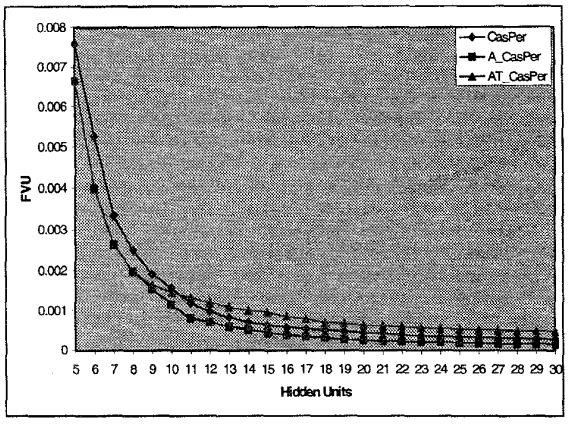

Fig. 5. Complex additive results – noise free.

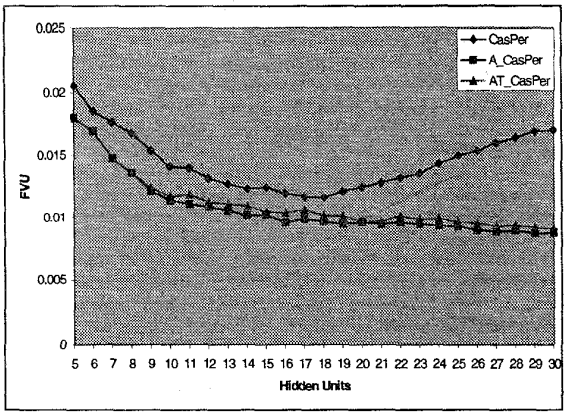

Fig. 6. Complex additive results - noisy

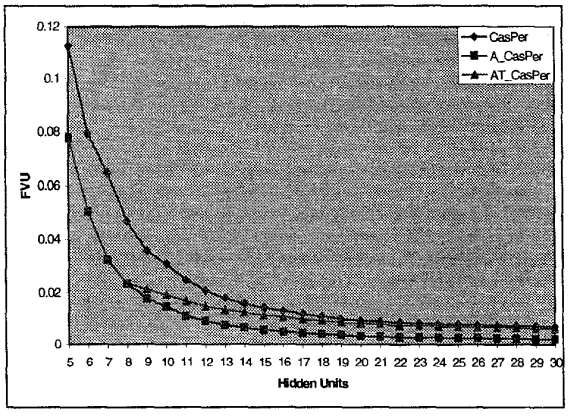

Fig. 7. Harmonic results – noise free

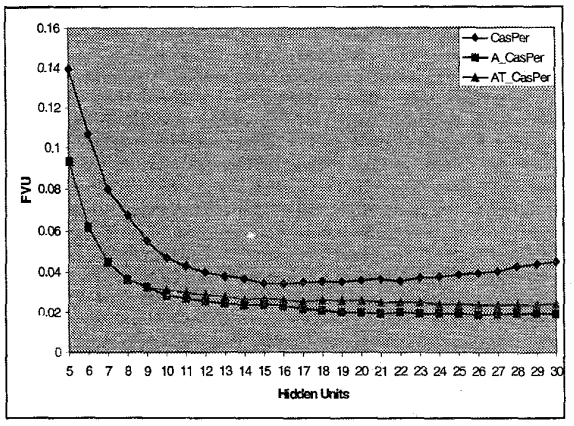

Fig. 8. Harmonic results - noisy

5. DISCUSSION

The results on the regression benchmarks show that A_CasPer and AT_CasPer are generally able to improve generalization results compared to CasPer using user optimized decay levels. This is especially apparent in the data sets containing noise, where A_CasPer and AT_CasPer not only obtain better generalization results, but are also able to avoid overfitting as the network continues to grow.

The improved generalization performance can be attributed to A_CasPer and AT_CasPer's ability to adapt the amount of decay required, taking into account the current complexity of the network and the error level. Since the constructed networks are growing in size, the amount of decay required for good generalization changes. In CasPer the user is unable to take this into account, and must guess at a decay rate that gives good generalization at a low error level. A_CasPer and AT_CasPer, however, can adapt the decay to the current complexity of the network. This is the reason that A_CasPer and AT_CasPer do not overfit the data in the noisy regression problems as the network size increases, as is obvious with CasPer in Figures 4, 6, and 8.

The one disadvantage of A_CasPer and AT_CasPer is the increased training times. Since the whole network must be retrained for each neuron in the pool, the computational complexity of these algorithms is significantly increased. It should be noted, however, that A_CasPer and AT_CasPer remove the need to set the decay parameter, which must be done manually in CasPer. Finding a good value for this parameter often involves repeated training attempts, which is avoided through this automatic selection method. In addition, A_CasPer and AT_CasPer are ideally suited to parallel implementation, either via a specific hardware architecture or the use of multiple processors. This allows for greatly reduced training times for these algorithms.

The method of constructing cascade towers as in AT_CasPer is another way to reduce the computational cost of training. AT_CasPer is able to maintain the good generalization results of A_CasPer, but since it uses fewer weights for a given size network, it require less training time. A plot of the median Connection Crossings (CCs) used by each algorithm through network construction is shown in Figure 9 for the radial function. CCs are defined as "the number of multiply–accumulate steps to propagate activation values forward through the network and error values backward" [3], and so provides a good indication of the computational cost of training. As expected in Figure 9, A_CasPer uses approximately six times as many CCs as CasPer, since A_CasPer is using a pool of six neurons. Figure 9 also shows the computational savings obtained by AT_CasPer over A_CasPer, with AT_CasPer being approximately twice as efficient after the construction of a 30 hidden unit network. Since there is an approximate linear growth of weights in AT_CasPer, this computational saving will increase for larger networks.

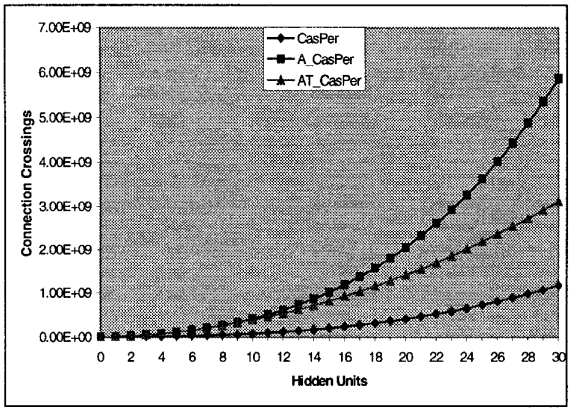

Fig. 9. Computational cost of each algorithm

It should be noted that since CasPer also has an exponential growth in weights, there will come a point when CasPer has six times as many weights as AT_CasPer, giving the two algorithms similar computational cost. This equivalence point happens around the addition of the 77th hidden unit.

It is often the case in the field of neural networks that much time is spent the by user in optimizing parameters such as network architecture, learning parameters and regularization constants. The removal of the decay parameter in the A_CasPer and AT_CasPer algorithms makes them free of any parameters which must be set by the user. This is a great advantage, as it allows these algorithms to be applied directly to problems without the user having to specify or optimize any parameters.

6. CONCLUSION

The introduction of a pool of neurons with differing network decay levels has the advantage of automatically specifying the amount of decay in the CasPer algorithm. This is especially relevant for constructive networks which require differing amounts of decay for a given problem as the network size increases. This results in not only freeing the user from finding a good weight decay value, but also produces networks which give better generalization than those obtained through user optimized decay values.

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, pp. 310-313.

[3] 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.

[4] 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.

[5] 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.

[6] Treadgold, N.K. and Gedeon, T.D. "Exploring Architecture Variations on Constructive Cascade Networks," *Int. Joint Conf. on Nerual Networks*, 1998, pp. 343-348.

[7] 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.

[8] 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.

[9] 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.