Adaptive Regularization in a Constructive Cascade Network

N.K. Treadgold and T.D. Gedeon
Email:nickt@cse.unsw.edu.au

Department of Information Engineering, School of Computer Science and Engineering,
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, generally by trial and error. This work explores the use of three training stages at each point of network construction. Each training stage has a different weight decay level, and the best performing decay level based on validation set performance is used to adapt the decay levels for future network regularization. This not only removes the need to select a decay value, but was also found to result in better generalization results compared to networks with fixed, user optimized, decay levels.

KEYWORDS: Regularization, Weight Decay, Constructive Network

1. Introduction

The CasPer algorithm [1], [2], [3] algorithm has been shown to be a powerful method for training feedforward 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) [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].

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 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 even an optimal decay parameter for a given size network will become redundant as the network grows. This work explores the use of an adaptive decay scheme which is implemented as the network is constructed. This paper will first give an introduction to the CasPer algorithm, then describe the adaptive regularization method and the results of some comparative simulations.

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 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 [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.

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

3. Adaptive Regularization

Adaptive regularization is implemented in CasPer through the use of three training stages for each new hidden neuron inserted into the network. Training is performed using the same method as the original CasPer algorithm for each training stage, 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 weights from the previous training stage are retained and act as the starting point for the next training stage. This retaining of weights was found to increase the convergence speed of the network.

The decay level for the network once a new neuron is added is set to the initial decay value, λ_i. It is this initial decay value which is adapted as the network is constructed. The first training stage uses this initial decay value, and each successive stage reduces this value by a factor of ten. 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 using the following method: if the best performing decay value occurred in the first two training stages, the initial decay is increased by a factor of ten, else it is decreased by a factor of ten. At this point the weights which produced the best validation results are restored to the network, the next neuron is added, and the process repeats.

This scheme begins with the addition of the first hidden unit, which is given an initial decay value of 0.01. The initial network with no hidden units is trained using a single training stage with a decay value of 0.01. For reasons of efficiency, if the validation result of the second stage is worse than the first, the third training stage is not performed.

The limits placed on the initial decay value are a maximum of 0.01 and a minimum of 0.0001, which gives a total decay range of 0.01 to 0.000001 (due to the three training stages). The top limit was found to provide a high enough decay level over a number of noisy regression and real world classification problems. Higher decay limits were found to result in a reduction in network convergence. The lower limit was selected to stop the decay 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.

This decay selection method allows the network to adapt the level of decay as the network grows in size. It should be noted that this method is biased to select larger initial decay values since this 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, more decay in general is needed. It was also found that better performances were obtained by starting the adaptation process with the high initial decay of 0.01, especially for data sets which converge quickly. The reason for this is that if the starting initial decay is too low, the network may converge before the decay level can be brought up to a high enough level.

Another point to note is the choice of reducing the decay level through each training stage. The motivation for this is that it allows the network to move into a general area of good solutions, which can then be refined by lowering the decay. This is the same motivation for the use of the SA term in the weight decay. This technique was found experimentally to be better than the opposite method of increasing decay values through the training stages. The algorithm incorporating this adaptive regularization method will be termed ACasPer.

4. Comparative Simulations

To investigate the performance of the ACasPer it was compared against that of CasPer with user optimized decay levels on a series of benchmark data sets. Three regression functions were chosen to compare CasPer and ACasPer. The functions are described in detail in [9], and are shown below.

- Radial function (rad):

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

- Complex additive function (cadd):

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

- Complex interaction function (cif):

$$\begin{aligned} f^{cif}(x_1,x_2) &= 1.9(1.35 + e^{x_1} sin(13(x_1-0.6)^2) \\ &\quad e^{-x_2} sin(7x_2)). \end{aligned}$$

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. The same abscissa values were used for all three functions. The noisy data was created by adding independent and identically distributed Gaussian noise, with mean zero and variance 0.0625, giving an approximate signal to noise ratio of 4 [9]. For each function an independent test set of size 10000 was generated on a regularly spaced grid $[0,1]^2$. ACasPer used a validation set of 110 randomly selected points (which included noise for the noisy datasets). The fraction of variance unexplained (FVU), as defined in [9], was the measure chosen to compare the performances on the test set. The FVU is proportional to the total sum of squares error. For each function 50 runs were performed using different random starting weights. Training was continued for both algorithms until 30 hidden units had been installed. The FVU on the test set was measured after the installation of each hidden unit and the median values are plotted in Figures 1 to 6. The standard CasPer constant settings were used [3], [4] for both CasPer and ACasPer.

The results displayed in Figures 1 to 6 show that ACasPer is able to obtain better generalization results using smaller networks than those of CasPer with user optimized decay. This improvement is most notable for the noisy data sets, where ACasPer can be seen to avoid overfitting, in comparison to CasPer which suffers significantly from it. The explanation for this is simple: since CasPer is using a fixed decay level, at a certain network size the decay level must become below optimal and overfitting occurs. One solution for this in CasPer is to set a high level of decay, but this can slow learning.

ACasPer, on the other hand, is able to adapt its decay level to maintain a good level of regularization. An example of

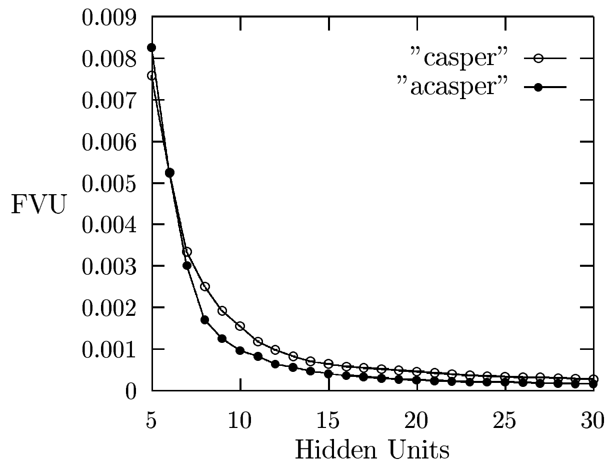

Figure 1: Radial results - noise free

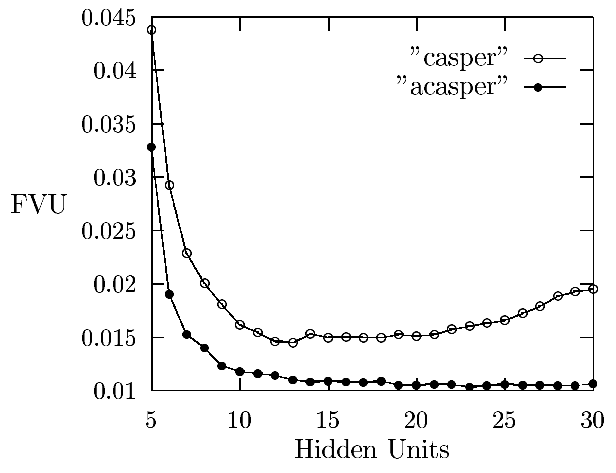

Figure 4: Complex additive results - noisy

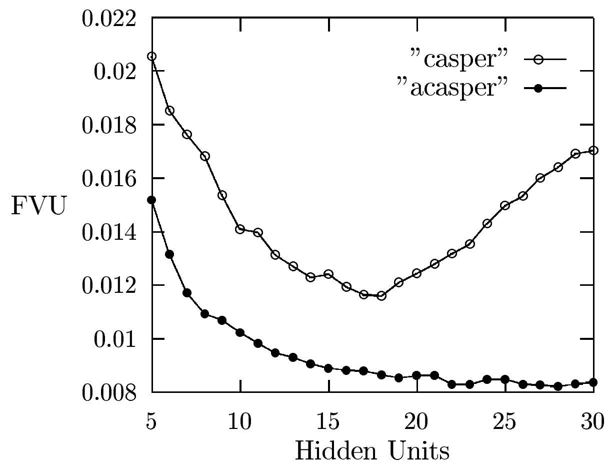

Figure 2: Radial results - noisy

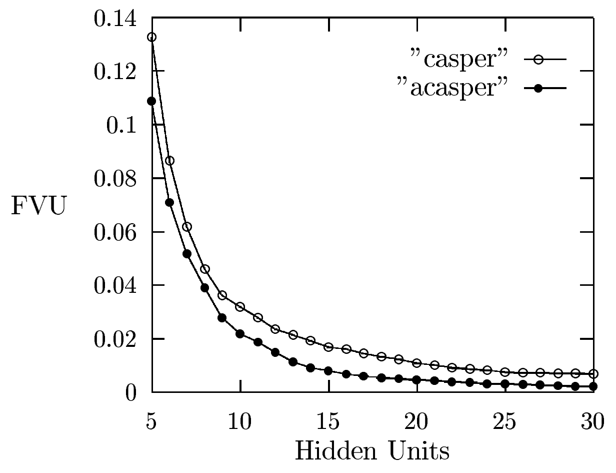

Figure 5: Complex interactive results - noise free

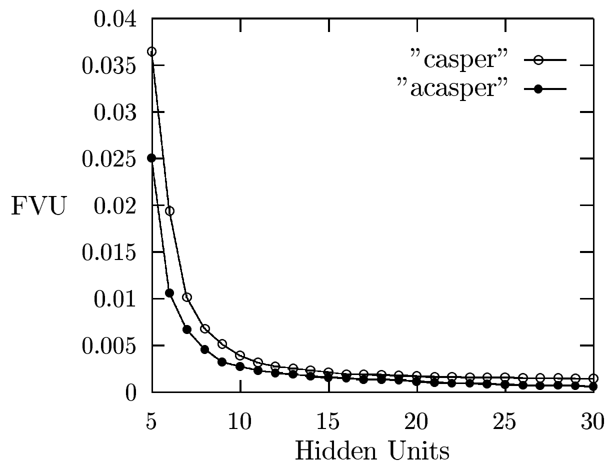

Figure 3: Complex additive results - noise free

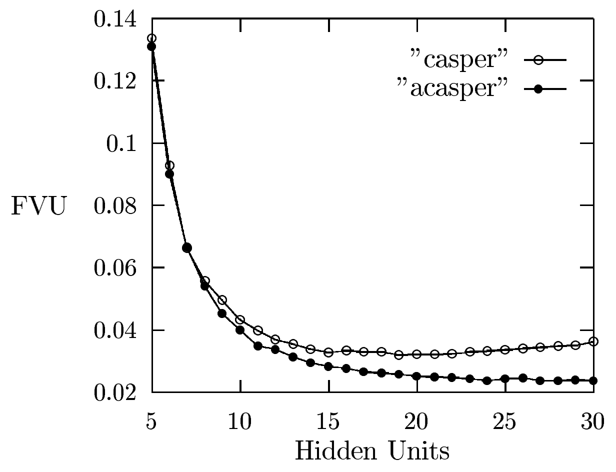

Figure 6: Complex interactive results - noisy

decay values selected by ACasPer in one run are shown in Table 1 for both the noise free and noisy versions of the cif data set. As expected, the noisy version uses greater decay, especially as the network grows larger. It should also be noted in the noisy version that early on in training ACasPer uses low levels of decay. The explanation for this is that at the early stages of training the network is learning the general features of the data and lower levels of decay allow this to be done more successfully. Since at this stage the network is relatively small, overfitting is unlikely to occur, and hence the validation set selects lower levels of decay.

Table 1: An example of adapted decay values

Hidden Unit	cif	cif - noise
0	0.01	0.01
1	0.0001	0.0001
2	0.001	0.00001
3	0.001	0.0001
4	0.0001	0.00001
5	0.0001	0.000001
6	0.0001	0.0001
7	0.001	0.001
8	0.0001	0.01
9	0.00001	0.01
10	0.000001	0.0001
11	0.000001	0.00001
12	0.000001	0.0001
13	0.000001	0.0001
14	0.000001	0.0001
15	0.000001	0.0001
16	0.000001	0.01
17	0.000001	0.01
18	0.000001	0.01
19	0.000001	0.01
20	0.000001	0.01
21	0.000001	0.01
22	0.00001	0.01
23	0.00001	0.01
24	0.000001	0.01
25	0.000001	0.01
26	0.000001	0.01
27	0.00001	0.01
28	0.00001	0.01
29	0.0001	0.001
30	0.00001	0.01

The disadvantage of ACasPer is that for a given network size, it is computationally more expensive than CasPer since it performs three training stages compared to CasPer's single one. Figure 7 shows a plot of the median connection crossings, a measure of computational cost [4], against hidden units installed for both algorithms. While ACasPer can be seen to be more computationally expensive, it has been shown to converge faster than CasPer and so less training will be required in general. In addition, the computational cost of selecting a good decay value by trial and error in CasPer is significant. The reason why ACasPer is less than three times as computationally expensive as CasPer is that often the third training stage is not performed.

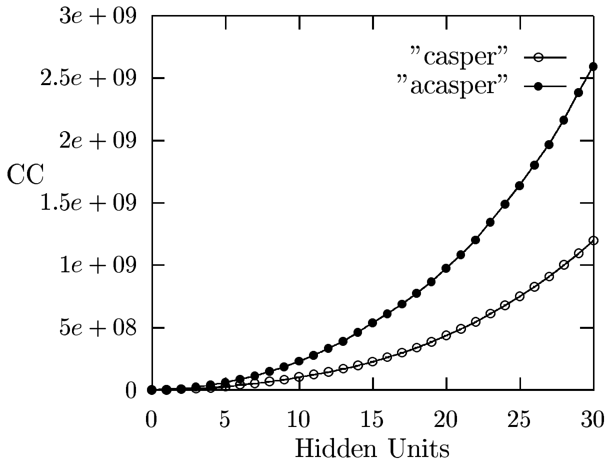

Figure 7: Connection Crossings (CC) for cif with noise

5. Conclusion

The introduction of an adaptive regularization scheme to the CasPer algorithm has been shown to produce better generalization with smaller networks than the original CasPer algorithm using fixed, user optimized, decay levels. It is also able to reduce overfitting of noisy data and removes the computational cost of finding good values for regularization.

References

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

[2] Treadgold, N.K. and Gedeon, T.D., "Extending CasPer: A Regression Survey," Int. Conf. On Neural Information Processing, pp.310–313 (1997).

[3] Treadgold, N.K. and Gedeon, T.D., "Extending and Benchmarking the CasPer Algorithm," AI'97, Perth Australia, pp.398–406 (1997).

[4] Fahlman, S.E. and Lebiere, C., "The Cascade-Correlation Learning Architecture," Advances in Neural Information Processing 2, pp.524–532 (1990).

[5] Hwang, J., You, S., Lay, S. and Jou, I., "The Cascade-Correlation Learning: A Projection Pursuit Learning Perspective," *IEEE Trans. Neural Networks*, Vol.4, 2, pp.278–289 (1996).

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

[7] Riedmiller, M. and Braun, H., "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," Proc. IEEE Int. Conf. On Neural Networks, pp.586–591 (1993).

[8] Treadgold, N.K. and Gedeon, T.D., "A Simulated Annealing Enhancement to Resilient Backpropagation," Proc. Int. Panel Conf. Soft and Intelligent Computing, pp.293–298 (1996).

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