Extending CasPer: A Regression Survey

N.K. Treadgold and T.D. Gedeon

Department of Information Engineering
School of Computer Science & Engineering
The University of New South Wales
Sydney N.S.W. 2052 AUSTRALIA
{ nickt | tom }@cse.unsw.edu.au

Abstract

The CasPer algorithm is a constructive neural network algorithm. CasPer creates cascade network architectures in a similar manner to Cascade Correlation. CasPer, however, uses a modified form of the RPROP algorithm, termed Progressive RPROP, to train the whole network after the addition of each new hidden neuron. Previous work with CasPer has shown that it builds networks which generalise better than CasCor, often using less hidden neurons. This work adds two extensions to CasPer. First, an enhancement to the RPROP algorithm, SARPROP, is used to train newly installed hidden neurons. The second extension involves the use of a pool of hidden neurons, each trained using SARPROP, with the best performing neuron selected for insertion into the network. These extensions are benchmarked on a number of regression problems and are shown to result in CasPer producing networks which generalise better than those produced by the original CasPer algorithm.

1 Introduction

The CasPer algorithm has been shown to be a powerful method for training neural networks [Treadgold and Gedeon, 1997]. CasPer is a constructive algorithm which inserts hidden neurons one at a time to form a cascade architecture, similar to Cascade Correlation (CasCor) [Fahlman and Lebiere, 1990].

CasPer has been shown to produce networks with fewer hidden neurons than CasCor, while also improving the resulting network generalisation, especially with regression tasks [Treadgold and Gedeon, 1997]. The reasons for CasPer's improved performance is that it does not use either Cascor's correlation measure, which can cause poor regression performance [Hwang, You, Lay, and Jou, 1996], or weight freezing, which can lead to oversize networks [Kwok and Yeung, 1993].

A problem faced by both CasPer and CasCor is that the newly created hidden neuron may have difficulty in converging to a good solution on the error surface. One main cause of this poor convergence may be the presence of local minima. CasCor addresses this problem by creating a pool of hidden neurons, each with a different set of starting weights. This enables a wider search of the error surface, and thus reduces the chances of the network becoming stuck in local minima.

The first extension to CasPer involves the addition of a modified version of RPROP, termed SARPROP, to train the weights of the newly installed hidden neuron. SARPROP uses Simulated Annealing in an attempt to overcome the problem of local minima. The second extension involves the use of a pool of neurons, as employed in CasCor, each trained using SARPROP.

2 The CasPer Algorithm

CasPer uses a modified version of the RPROP algorithm [Riedmiller and Braun, 1993] 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 that weight as it traverses the error surface.

The CasPer algorithm constructs cascade networks in a similar manner to CasCor: CasPer starts with a single hidden neuron and successively inserts hidden neurons. 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 which 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 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 benefit can be gained by the network by modifying an old weight, this occurs, albeit at an initially slower rate than the weights connected to the new neuron.

CasPer also makes use of weight decay as a means to improve the generalisation properties of the constructed network. After some experimentation it was found that the addition of a Simulated Annealing (SA) term applied to the weight decay, as used in the SARPROP algorithm [Treadgold and Gedeon, 1996], often improved convergence and generalisation. 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. The error gradient used in CasPer thus becomes:

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

HEpoch refers to the number of epochs elapsed since the addition of the last hidden neuron, *sign* returns the sign (positive/negative) of its operand, and *D* is a user defined parameter which effects the magnitude of weight decay used.

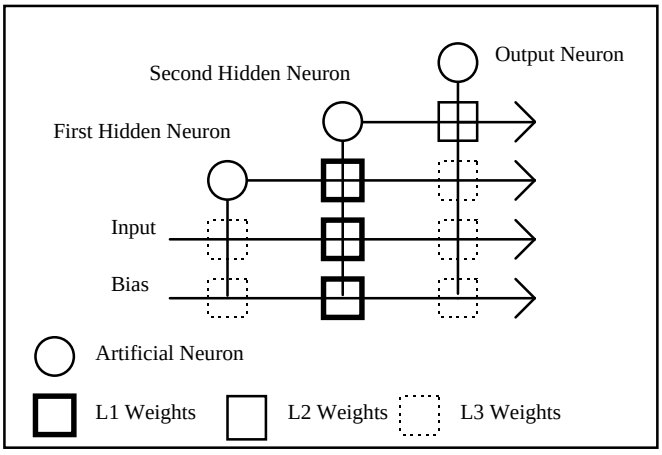

Fig. 1. The CasPer architecture - a second hidden neuron has just been added. The vertical lines sum all incoming values.

In CasPer a new neuron is inserted when the RMS Error falls by less than 1% of its previous value. The time period in epochs over which this measure is taken is given by the heuristic formula: 15 + L*N, which was found experimentally to give good results (*N* is the current number of hidden neurons, and *L* is the training length parameter which is set prior to training). The result of this training method is that CasPer increases the period over which the network is trained as the network grows in size.

3 Extensions To CasPer

Both extensions proposed for CasPer are aimed at increasing the ability of the network to find better solutions on the error surface when a new hidden neuron is added. The first achieves this by modifying the RPROP algorithm to improve its ability to escape local minima. The second uses multiple initial starting points for the new hidden neuron, as in CasCor.

The first extension to the CasPer algorithm involves the use of the SARPROP algorithm to train all the incoming weights of the newly inserted hidden neuron (that is, all weights in group L1). All remaining weights are still trained by RPROP. The idea behind this is that while the old neurons are able to maintain their knowledge to a large extent (although some learning is still possible), the new neuron is able to quickly learn the remaining error while using the SA provided by SARPROP to escape from local minima.

The SARPROP algorithm is based on RPROP, but uses a noise factor to enhance the ability of the network to escape from local minima [Treadgold and Gedeon, 1996]. 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 minima), and the magnitude of the update value is less than a threshold value. The amount of noise added falls as training continues via the use of a SA term. 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 $2^{-0.03*HEpoch}$. The version of CasPer employing this extension will be termed S_CasPer.

The second extension involves the use of a pool of hidden neurons, each with different initial random starting weights. Each hidden neuron in the pool is trained using SARPROP, as in the first extension. Each network is trained with a new hidden neuron from the pool, and the network giving the smallest RMS Error value is selected. It should be noted that in CasPer the whole network must be trained for each hidden neuron in the pool, so this extension will be computationally expensive. This extension will be termed SP_CasPer.

Two minor modifications were also made to the original CasPer algorithm. First, training was started with no hidden neurons. The reason for this change is that many problems can be satisfactorily solved with no hidden neurons. Second, the time period over which hidden neuron training is performed was simplified from 15+L*N to L. This was done to allow the SA term to act over a longer period (a larger L value is used than in the original CasPer). These modifications were used in both S_CasPer and SP_CasPer.

4 Comparative Simulations

To test the effectiveness of S_CasPer and SP_CasPer, their performance was compared against that of CasPer on two regression benchmarks. CasPer, S_CasPer, and SP_CasPer share a number of parameters which were set as follows. The following (standard) RPROP values were used: $\eta^+ = 1.2$, $\eta^- = 0.5$, $\Delta_{max} = 50$, $\Delta_{min} = 1\text{x}10^{-6}$. A constant value of 0.0001 was added to the derivative of the sigmoid in order to overcome the 'flat spot' problem [Fahlman, 1988]. Weights in the initial network were initialised to evenly distributed random

values in the range -0.7 to 0.7. All weights associated with newly inserted hidden neurons were initialised in the range -0.1 to 0.1. Linear output nodes were used.

Training of the initial network used the initial update value $\Delta_0 = 0.2$. The values of *L1*, *L2*, and *L3* were set to 0.2, 0.005, and 0.001 respectively. All of the above parameter values were found to be essentially problem independent, and hence are treated as constants. The remaining parameter values *D* (the weight decay value) and *L* (the training length), were set depending on the problem. For S_CasPer and SP_CasPer, the SARPROP parameters used were: $k_1 = 1$, and $k_2 = 0.4$. SP_CasPer used a pool of eight hidden neurons.

Two regression functions were chosen to compare CasPer, S_CasPer and SP_CasPer. These functions are described in detail in [Hwang *et al*., 1994] and shown below:

Additive Function:

$$add(x_1, x_2) = 1.9(1.35 + e^{x_1}\sin(13(x_1 - 0.6)^2)e^{-x_2}\sin(7x_2)).$$

Complex Interaction Function:

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

The generation of training and test data follows the method of [Hwang *et al*., 1994]. For each function 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 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 unit variance, giving an approximate signal to noise ratio of 4. For each function an independent test set of size 2500 was generated on a regularly spaced grid $[0,1]^2$.

CasPer used $L = 7$ and $D = 5\text{x}10^{-4}$ for all simulations as in Treadgold and Gedeon [1997]. S_CasPer used $L = 200$ for all simulations. For the noisy data sets S_CasPer set $D = 1\text{x}10^{-3}$. For the noise free data sets S_CasPer used $D = 1\text{x}10^{-5}$ and $D = 1\text{x}10^{-4}$ for the additive and complex interaction functions respectively. These parameter values were chosen to maximise the performance of CasPer and S_CasPer on the data sets. SP_CasPer used the same parameter settings as S_CasPer.

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 for both algorithms 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 for CasPer, S_CasPer and SP_CasPer are shown in Figures 2 and 3 for the noise free additive and complex interaction functions respectively. Figures 4 and 5 show the corresponding noisy results.

5 Discussion

As can be seen from the simulation results, S_CasPer produces networks which obtain better generalisation results at earlier stages of training than those of CasPer.

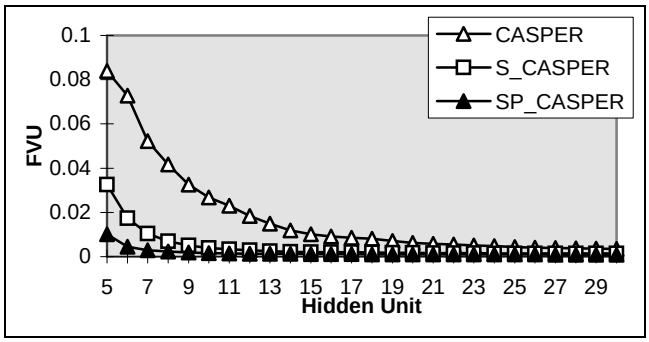

Fig. 2. Additive function results - noise free

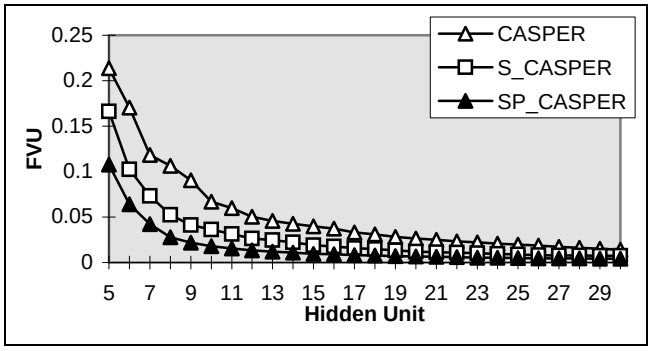

Fig. 3. Complex interaction results - noise free

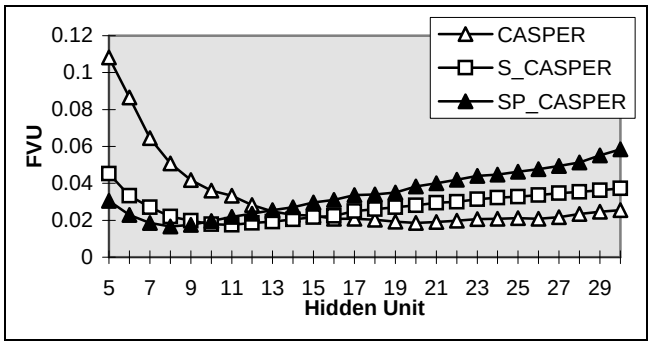

Fig. 4. Additive function results - noisy

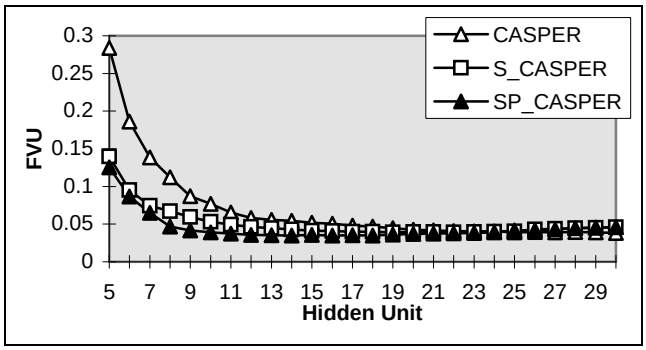

Fig. 5. Complex interaction results - noisy

This improvement can be attributed in the main to SARPROP's more thorough search of the error surface. Similar improvements are made by SP_CasPer over S_CasPer, which again can be attributed to the larger search performed by SP_CasPer's pool of hidden neurons. Unfortunately, SP_CasPer is more computationally expensive than S_CasPer, so there is a tradeoff of the additional training time against the increases in network performance.

An advantage of SP_CasPer, however, is that it is ideally suited for parallel implementation, with each network in the pool trained on a separate processor. This largely overcomes the additional time required to train a pool of neurons.

The results on the noisy data sets show that S_CasPer, and especially SP_CasPer, are more susceptible to overfitting than CasPer as the networks are grown. Importantly, however, S_CasPer and SP_CasPer are still able to produce better overall results with smaller networks on these data sets than CasPer. The increased overfitting of SP_CasPer compared to S_CasPer is expected, since SP_CasPer selects the network from the pool with the best error on the training set. Hence overfitting is more likely with noisy data sets.

Previous work done by Kwok and Yeung [1996] allows a comparison of these results with Projection Pursuit Learning (PPL), a powerful regression tool. Kwok and Yeung used the same regression functions, 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 1 shows the best average FVU obtained by PPL and its extension (PPL_B) from a range of hidden neurons and Hermite function orders. Table 1 also shows the corresponding best averages using S_CasPer and SP_CasPer.

Table 1. Average FVU for S_CasPer, SP_CasPer, PPL, and PPL_B.

Function	S_CasPer	SP_CasPer	PPL	PPL_B
add	0.00195	0.00117	0.00115	0.00143
cif	0.01298	0.00612	0.03818	0.04013
add^{noise}	0.01958	0.01783	0.04439	0.04416
cif^{noise}	0.03967	0.03618	0.09919	0.08967

The results on the noise free data sets show PPL slightly outperforming S_CasPer and SP_CasPer on the easier additive data set, although for the more difficult complex interaction data set, S_CasPer and SP_CasPer show better results. On all the noisy data sets S_CasPer and SP_CasPer significantly outperform PPL and PPL_B.

6 Conclusions

The S_CasPer extension of the CasPer algorithm, which uses SARPROP as the training method for the newly inserted hidden neurons, results in better generalisation occurring at an earlier stage than CasPer. The SP_CasPer extension, which uses of a pool of hidden neurons trained by SARPROP, results in further generalisation improvements, although there is an increased computational cost. Benchmarking results on additional regression problems show similar trends as those reported here.

References

1. S.E. Fahlman and C. Lebiere, The cascade-correlation learning architecture, *Advances in Neural Information Processing* 2, D.S. Touretzky, (Ed.) San Mateo, CA: Morgan Kauffman, 524-532 (1990).
2. S.E. Fahlman, An empirical study of learning speed in back-propagation networks, Technical Report CMU-CS-88-162, Carnegie Mellon University (1988).
3. J. Hwang, S. Lay, R. Maechler, and D. Martin, Regression Modeling in Back-Propagation and Projection Pursuit Learning, *IEEE Trans. Neural Networks* **5**(3), 342-353 (1994).
4. J. Hwang, S. You, S. Lay, and I. Jou, The Cascade-Correlation Learning: A Projection Pursuit Learning Perspective, *IEEE Trans. Neural Networks* **7**(2), 278-289 (1996).
5. T. Kwok and D. Yeung, Experimental Analysis of Input Weight Freezing in Constructive Neural Networks, *Proc IEEE Int. Conf. On Neural Networks*, 511-516 (1993).
6. T. Kwok and D. Yeung, Use of Bias Term in Projection Pursuit Learning Improves Approximation and Convergence Properties, *IEEE Trans. Neural Networks* **7**(5), 1168-1183 (1996).
7. M. Riedmiller and H. Braun, A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm, *Proc IEEE Int. Conf. on Neural Networks*, 586-591 (1993).
8. N.K. Treadgold and T.D. Gedeon, A Cascade Network Employing Progressive RPROP, *Int. Work Conf. on Artificial and Natural Neural Networks*, 733-742 (1997).
9. N.K. Treadgold, N.K. and T.D. Gedeon, A Simulated Annealing Enhancement to Resilient Backpropagation, *Proc. Int. Panel Conf. Soft and Intelligent Computing*, Budapest, 293-298 (1996).