A Dual-Phase Technique for Pruning Constructive Networks

J.P. Thivierge
Department of Psychology
McGill University
Montreal, QC Canada
H3A 1B1
management@neurostate.com

F. Rivest
Département d'Informatique et
de Recherche Opérationnelle
Université de Montréal
Montreal, QC Canada
H3C 3J7
francois.rivest@mail.mcgill.ca

T. R. Shultz
Department of Psychology and
School of Computer Science
McGill University
Montreal, QC Canada
H3A 1B1
thomas.shultz@mcgill.ca

***Abstract* - An algorithm for performing simultaneous growing and pruning of Cascade-correlation (CC) neural networks is introduced and tested. The algorithm adds hidden units as in standard CC, and removes unimportant connections by using Optimal Brain Damage (OBD) in both the input and output phases of CC. To this purpose, OBD was adapted to prune weights according to the two separate objective functions that are used in CC to train the candidate hidden units and to train the network, respectively. Application of the new algorithm to two databases of the PROBEN1 benchmarks reveals that this new dual-phase pruning technique is effective in significantly reducing the size of CC networks, while providing a speed-up in learning times and improvements in generalization over novel test sets.**

I. INTRODUCTION

Setting the ideal size of a neural network's topology is a major problem in many simulations. There are potential advantages as well as disadvantages to both large and small networks. Large networks may fit a given data set, but often generalize poorly. Too many weights in a network create many degrees of freedom, which tends to overfit the data set. In overfitting, a network effectively memorizes the noise components in the data. Large networks are also more computationally demanding to run, and more difficult to analyze. Smaller networks, on the other hand, can fail to learn the data set. Although these size effects are well documented, it is unclear how to find an optimal trade-off point between large and small networks.

In solving the problem of defining an optimal network topology, two main suggestions emerge. First, it is possible to decide on a relatively large network and then prune superfluous or redundant connections. Many approaches exist for pruning feed-forward neural networks, including optimal brain damage (OBD) [1], optimal brain surgeon [2], and skeletonization [3]. A review of major pruning algorithms is available from [4]. There are some problems with the pruning approach, including increases in computational cost due to starting with a larger topology than necessary, and not being certain that the initial topology is big enough. But most importantly, using pruning techniques to determine the ideal size of a network can result in network topologies that never reach acceptable levels of accuracy on some classification problems [5].

The second possibility is to start with a very small network and add nodes to grow it as necessary to learn a problem. A partial review of constructive networks is available from [6]. Cascade-correlation (CC) [7] is a popular example of such an algorithm. This type of network grows as it learns by installing fully connected nodes into the network topology. One drawback of this solution is the creation of a large interconnected system that is very deep in layers. Typically, each new hidden unit in a CC network is installed on a separate layer. The large number of connections may disrupt generalization, and increase computational demands. By adding layers of fully connected neurons, many connections may become redundant. In fact, even when pruning a large number of weights in a CC network, there is no significant reduction in generalization [8]. This perhaps explains the tendency of CC networks to overfit the training data [9]. Finally, CC networks as well as many other growing algorithms have the disadvantage of always ending up with the same type of architecture. CC networks do offer a solution to defining a network architecture in terms of its nodes and layers, but not in terms of its connections. The resulting connection scheme of CC networks is always the same, and is therefore not fully adapted to the target problem. It is thus important to determine a technique to sort connection weights by importance and remove the unnecessary ones.

To our knowledge, integrating connection pruning in the learning scheme of a CC network has not being explored. Reference [8] is the only known source to report a few possibilities for pruning CC networks, but lacks extensive empirical testing and development. In this paper, we first describe a generalized framework for CC algorithms. Then, we describe how OBD pruning can be applied to their connection weights. Finally, we provide some empirical results based on data from the PROBEN1 repository.

II. DESCRIPTION OF CC NETWORKS

CC is a constructive algorithm of supervised learning where new hidden units are added, layer by layer, to the existing network topology based on the requirements of a given problem. A network starts with no hidden units and grows by comparing a number of candidate units in parallel and recruiting the best candidate into the network. In order

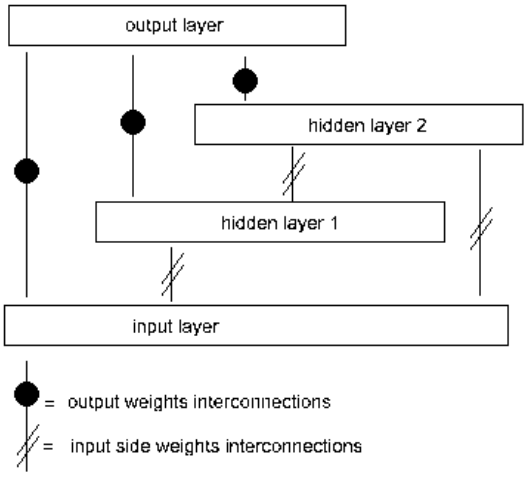

Fig. 1. Architecture of a CC network with 2 hidden layers.

to both grow and learn, CC networks alternate between two phases, namely input and output. In output phase, a network learns by adjusting output weights (see Figure 1) to minimize error at the output level:

$$F = \sum_{p=1}^{PC} \sum_{o=1}^{OUTC} \left(\vec{V}_{out,o,p} - \vec{T}_{o,p} \right)^2 \qquad (1)$$

where o is an output node, p is a pattern, PC is the pattern count, $OUTC$ is the output count, $\vec{V}_{out} \in \Re^{OUTC}$ is a vector of network outputs, and $\vec{T} \in \Re^{OUTC}$ is a target activation column vector.

In input phase, a new candidate unit gets recruited in the network. To determine which of a number of candidates gets installed, the input-side weights feeding those candidates are trained in order to maximize G, a measure of covariance between the network's training error and the output of candidates:

$$\begin{aligned} G &= \frac{1}{c} \sum_{o=1}^{OUTC} \sum_{co=1}^{COUTC} Cov(co,o)^2 \\ &= \frac{1}{c} \sum_{o=1}^{OUTC} \sum_{co=1}^{COUTC} \left[\sum_{p=1}^{PC} \vec{V}_{cand,co,p} \cdot \vec{E}_{o,p} - {}_{PC} \cdot \vec{V}_{cand,co,\bar{p}} \cdot \vec{E}_{o,\bar{p}} \right]^2 \end{aligned} \qquad (2)$$

unit c, $COUTC$ is the candidate output count, $\vec{E} \in \Re^{OUTC}$ is an error column vector equal to $\vec{V}_{out} - \vec{T}$, and $\vec{V}_{cand} \in \Re^{COUTC}$ is a candidate activation column vector. In this phase, the covariance of each candidate's output is compared with the network's training error and a winner-take-all scheme is applied where the candidate with the highest covariation is installed in the network, and all other candidates are discarded until another unit is required. We redefined part of the objective function G in input phase to include a Frobenius norm in its calculation. By squaring the covariance, the Frobenius norm is an alternative to computing the absolute value of the covariances as in classic CCs. Squaring the covariance has the advantage of a continuously differentiable objective function over every output of the network and a given candidate [10].

Once a candidate is installed, its input-side weights are frozen in place to preserve the unit's feature detection abilities. CC networks typically add a new hidden layer every time a candidate unit gets recruited. The new candidate is fed by all previous units, and feeds the output layer with feed-forward connections.

Networks are constrained to always start and end in output phase. Three criteria govern phase switching, namely (1) performance is within some predefined range of error for F or covariance for G; (2) a limit of epochs has been reached; (3) or improvement of the objective function F or G has stagnated for a certain number of epochs [11].

The output value of a CC network can be obtained by:

$$\begin{aligned} \vec{V}_{out} &= \vec{f}_{out} \left(W_{out} \cdot \vec{V}_{\text{int}} \right) \\ &= \vec{f}_{out} \left(\left[\vec{W}_{out,1} \quad \ldots \quad \vec{W}_{out,INC} \right]^{\mathrm{T}} \cdot \vec{V}_{\text{int}} \right) \\ &= \vec{f}_{out} \left(\left[\vec{W}_{out,1} \cdot \vec{V}_{\text{int}} \quad \ldots \quad \vec{W}_{out,INC} \cdot \vec{V}_{\text{int}} \right]^{\mathrm{T}} \right) \end{aligned} \qquad (3)$$

where $W_{out} \in \Re^{INC \times INTC}$ is a weight row vector feeding the output layer, $\vec{V}_{\text{int}} \in \Re^{INTC}$ is an internal activation column vector, $\vec{f}_{out} = \Re^{INC} \mapsto \Re^{OUTC}$ is an output layer activation function (column vector), and INC is the output layer input count. Because each weight vector is used only in the i^{th} input of the output layer function, let's look at the derivative of a network with respect to each weight vector separately and each output separately:

$$\frac{\partial \vec{V}_{out,o}}{\partial \vec{W}_{out,i}} = \frac{\partial \vec{f}_{out,o}\left(W_{out} \cdot \vec{V}_{\text{int}}\right)}{\partial\left(\vec{W}_{out,i} \cdot \vec{V}_{\text{int}}\right)} \cdot \frac{\partial\left(\vec{W}_{out,i} \cdot \vec{V}_{\text{int}}\right)}{\partial \vec{W}_{out,i}} \quad (4)$$
$$= {}_i\vec{f}'_{out,o}\left(W_{out} \cdot \vec{V}_{\text{int}}\right) \cdot \vec{V}_{\text{int}}{}^{T}$$

The second order derivative with respect to its weights can be obtained as:

$$\frac{\partial^2 \vec{V}_{out,o}}{\partial W_{out}{}^2} = \begin{bmatrix} {}_1\vec{f}''_{out,o}\left(W_{out} \cdot \vec{V}_{\text{int}}\right) \cdot \vec{V}_{\text{int}} \cdot \vec{V}_{\text{int}}{}^{\text{T}} \\ \vdots \\ {}_{INC}\vec{f}''_{out,o}\left(W_{out} \cdot \vec{V}_{\text{int}}\right) \cdot \vec{V}_{\text{int}} \cdot \vec{V}_{\text{int}}{}^{\text{T}} \end{bmatrix} \quad (5)$$

for each weight vector and output separately. These values will become useful in the following section for pruning calculations.

III. OPTIMAL BRAIN DAMAGE

There are a variety of methods available for pruning nodes and connections in neural networks, ranging from computationally expensive ones to much simpler ones. The simplest approach is undoubtedly to base pruning on the absolute magnitude of weight values, and to prune weights with values close to zero. The problem with this technique is that it often leads to elimination of the wrong weights [12]. In fact, small weights that are near zero may be important in reducing error. More sophisticated measures try to capture the precise impact that a given node or weight has on the training error or generalization capabilities of a network. Skeletonization [3], for instance, is a method that involves removing nodes that have the least effect on reducing output error. OBD [1], which is explored here, is another example of this technique. OBD removes weights that have the least effect on training error based on a diagonal assumption of the Hessian matrix, which contains all the partial second derivatives of weights with respect to the objective function of a network. In OBD, this matrix is assumed to be diagonal, which means that each row can be estimated by it's corresponding diagonal entry, and that all other weights on either side of the diagonal are assumed to be near zero, and thus of negligible value.

OBD is a technique designed to prune unnecessary or redundant connections from a network. OBD uses information-theoretic ideas to remove unimportant weights by comparing the saliency of each weight and eliminating the ones with the lowest saliency. The measure of saliency is typically calculated as a second order derivative of error of a network with respect to each of its weights. Saliency indicates the sensitivity of the network to removal of that connection. In order to use saliency to prune CC networks, we must redefine this last calculation. For CC networks in output phase, this objective function will be *F*, the minimization of error. However, for CC networks in input phase, this objective function will be *G*, the maximization of covariance between candidate output and network error. Redefining OBD pruning in this way enables input-side weight pruning in input phase and output weight pruning in output phase because those are the weights affected by the objective function in their respective phases.

Two main calculations are necessary in order to perform OBD weight pruning in both input and output phases. First, we obtain the second order derivative of the *F* objective function (minimization of sum squared error) with respect to each separate output weight in a network. Second, we compute the second order derivative of the *G* objective function (maximization of covariance) with respect to each separate input side weights in a network. The first derivative of *F* is obtained from the networks as:

$$\frac{\partial F}{\partial W_{out}} = 2\sum_{p=1}^{PC}\sum_{o=1}^{OUTC}\left(\vec{V}_{out,o,p} - \vec{T}_{o,p}\right) \cdot \frac{\partial \vec{V}_{out,o,p}}{\partial W_{out}} \quad (6)$$

for each pattern *p*.

From the first derivative, the second derivative of *F* with respect to the weights of the network can be obtained as:

$$\frac{\partial^2 F}{\partial W_{out}{}^2} =$$
$$2\sum_{p=1}^{PC}\sum_{o=1}^{OUTC}\left[\frac{\partial \vec{V}_{out,o,p}}{\partial W_{out}} \cdot \frac{\partial \vec{V}_{out,o,p}}{\partial W_{out}}^{\text{T}} + \left(\vec{V}_{out,o,p} - \vec{T}_{o,p}\right) \cdot \frac{\partial^2 \vec{V}_{out,o,p}}{\partial W_{out}{}^2}\right] \quad (7)$$

for every pattern *p* independently.

For input phase pruning of input-side weights, we must obtain the first and second derivatives of the covariance, averaged over all patterns:

$$Cov(co,o) = \sum_{p=1}^{PC}\left(\vec{V}_{cand,o,p} - \vec{V}_{cand,o,\bar{p}}\right) \cdot \left(\vec{E}_{o,p} - \vec{E}_{o,\bar{p}}\right)$$
$$= -PC \cdot \vec{V}_{cand,co,\bar{p}} \cdot \vec{E}_{o,\bar{p}} + \sum_{p=1}^{PC}\vec{V}_{cand,co,p} \cdot \vec{E}_{o,p}$$

$$Cov'(co,o) = \sum_{p=1}^{PC}\vec{V}'_{cand,co,p} \cdot \left(\vec{E}_{o,p} - \vec{E}_{o,\bar{p}}\right)$$

$$Cov''(co,o) = \sum_{p=1}^{PC}\vec{V}''_{cand,co,p} \cdot \left(\vec{E}_{o,p} - \vec{E}_{o,\bar{p}}\right) \quad (8)$$

From these values, the second derivative of *G* with respect to the candidate feeding weights is obtained as:

$$\frac{\partial^2 G}{\partial W_{cand}{}^2} = \frac{\partial}{\partial W_{cand}}\left(\frac{\partial G}{\partial W_{cand}}\right)$$

$$= \frac{2}{C}\sum_{o=1}^{OUTC}\sum_{co=1}^{COUTC} Cov'(co,o)\cdot Cov'(co,o)^{\mathrm{T}} + Cov(co,o)\cdot Cov''(co,o) \tag{9}$$

IV. SIMULATIONS

Simulations were performed to assess the potential advantages of OBD pruning. A technique called *early stopping* was employed to determine when to stop pruning. In this strategy, weights were pruned from the network until error on a generalization set started to be negatively affected. One hundred networks were trained on each of two data sets. There were four different pruning conditions: (1) OBD pruning of input-side weights at the end of every input phase; (2) OBD pruning of output weights at the end of training; (3) a combination of (1) and (2); and (4) no pruning at all. The various pruning conditions were designed to eliminate weights after the particular optimization guiding their fluctuations had reached a final minimum. According to [13] retraining a highly pruned network may lead to inferior performance. Because input-side weights are frozen into place after each input phase, these weights could be pruned while learning without breaking this requirement. Output-side weights, however, are retrained throughout learning, and could only be pruned at the end of learning to maintain the requirement. In this way, no weights were ever retrained after pruning. This extra requirement also solves a performance disadvantage found when pruning networks that are still in a local minimum [14]. As an implementation strategy, pruned weights were set to a value of zero, which is mathematically equivalent to eliminating their connection.

A. DataSets

The two datasets employed for simulations were taken from the PROBEN1 repository [15], available from ftp://ftp.ira.uka.de. The datasets represent realistic tasks and contain real world data. The glass database was comprised of 9 attributes, 7 outputs, and a total of 214 patterns. The diabetes database was comprised of 8 attributes, 1 output, and 768 patterns. Both were classification problems with no missing values. Input data for both databases was represented as continuous values, and a standard Z-score transformation was applied on each value:

$$z_i = \frac{x_i - \mu}{\sigma} \tag{10}$$

where μ is the average and σ is the standard deviation. *Z* scores standardization brings back raw scores to a distribution of μ=0 and σ=1. The resulting *z* scores served as input to the networks. Each data set was divided into three parts, and included a train set, a 10-fold cross-validation test set, and an independent test set representing 10% of the total data set. The independent test set was employed for early stopping, and the cross-validation subset was used to test networks after pruning was applied.

B. Results

A summary of results over the two databases is presented in Table 1. For all comparisons of results, one-way ANOVAs were employed, with a minimal accuracy criterion of $\alpha \leq 0.01$. The goal of this test was to determine if the differences found between the various conditions of pruning are statistically reliable and denote a real impact of pruning. When required, post-hoc comparisons were performed using the Scheffé technique.

TABLE 1
PERFORMANCE OF PRUNED NETWORKS

	Glass database				
	Input and output	Input only	Output only	No pruning	p <
Cross-validation error	**13.56**	14.1	14.61	14.69	0.11 (ns)
Novel error*	**20.41**	27.59	22.34	--	0.01
Epochs	3130.85	**3121.46**	3126.01	3126.01	0.01
Number pruned*	208.2	**208.6**	1	--	0.01
	Diabetes database				
Cross-validation error	**17.37**	31.47	33.1	31.17	0.05 (ns)
Novel error*	**21.32**	48.97	32.92	--	0.01
Epochs	**2956.07**	2974.65	3004.38	3004.38	0.01
Number pruned*	242.7	**246.5**	1.22	--	0.01

Note. Best results in each condition are underlined. Performance measures marked by an asterisk involve pruning. Novel error testing was used in early stopping and was not performed for the condition where no pruning was applied. ns = not significant.

1) *Accuracy of generalization on cross-validation test set*: ANOVA results showed no significant differences for cross-validation generalization ($F(3, 386) = 2.06$, $p > 0.105$). This result could be attributed to high variance in the group of

networks. The same results apply to the diabetes database ($F(3, 386) = 29.84$, $p > 0.045$). We note, however, that in both databases results pointed in the same direction: combined input and output pruning lead to the best generalization.

2) *Accuracy of generalization on novel data*: ANOVA did reveal differences for novel generalization ($F(2, 297) = 132.21$, $p < 0.0005$). Scheffé comparisons yielded significant differences between each of the pruning conditions. Input and output pruning had the least error, followed by input pruning, and then output pruning. The same precise pattern of results held for the diabetes database ($F(2, 297) = 27.22$, $p < 0.0005$).

3) *Number of pruned connections*: For the diabetes database, a total number of 35 output weights were present in the initial networks. The total number of hidden units was of 25, which was the maximum allowable. The total number of input-side weights in our networks reached 525. When considering the weights that could be pruned, the percentage of weights pruned was 43.34% in the combined input and output pruning condition, 44.02% in the input pruning condition, and 0.02% in the output pruning condition. For the second database containing the glass problem, a total number of 36 output weights were present in the initial networks. The total number of hidden nodes grew to 25, which again was the maximum allowable. The total number of input-side weights reached 245. The percentage of weights pruned was 74.09% in the combined input and output pruning condition, 78.23% in the input pruning condition, and 0.36% in the output pruning condition.

For the glass problem, ANOVA revealed group differences between the 3 pruning conditions in the average number of pruned connections ($F(2, 27) = 790.39$, $p < 0.0005$). Scheffé comparisons revealed less pruning for the output condition than in the other two conditions where input phase pruning was present. No significant difference was found between these latter conditions. The same pattern of results was obtained for the diabetes problem ($F(2, 27) = 197.23$, $p < 0.0005$).

4) *Number of epochs required for training*: For the glass database, ANOVA revealed differences in the number of epochs necessary to train the networks across all four conditions ($F(3, 396) = 482.65$, $p < 0.0005$). Scheffé comparisons revealed that input pruning reduced the number of epochs when compared to the other three conditions. No significant differences were found anywhere else. For the diabetes database, significant group differences were found ($F(3, 396) = 916.57$, $p < 0.0005$). Scheffé comparisons revealed that input and output pruning, as well as input pruning, required less epochs than output pruning and no pruning. In sum, pruning, particularly input pruning, can reduce training time.

V. DISCUSSION

The current study is the first to demonstrate how to adapt the OBD pruning technique to a CC constructive neural network. Because CC uses two distinct optimization techniques at different times, OBD was adapted to a dual-phase pruning. Input-side weights were pruned at the end of each input phase, while output weights were pruned at the end of training. Advantages of using this dual-phase OBD to prune CC networks were demonstrated on two real-world tasks taken from the PROBEN1 benchmarks. Results show that the new technique was efficient in reducing network size by pruning almost half of the connections. Pruning also led to better performance on generalizing to unseen data independent of the cross-validation sets. These results replicated those of a number of other studies on using OBD with Backprop networks [1].

One issue of concern was whether pruning would prolong training. Input-side pruning actually reduced training times in both the glass and diabetes database. The goal of hidden units is to specialize in capturing specific areas of the error surface. By pruning input-side weights, it is possible that we have found a way to avoid over-specialization of those hidden units in solving various regions of a target problem, and encourage more general solvers. In any case, input pruning improved generalization over the entire train set. In turn, improving the fit of weights to an entire set instead of local regions lower the overall error, and thus reduce the number of epochs necessary to reach a solution.

OBD pruned significantly more input-side weights than output weights, and so was more responsible for improving generalization and reducing training epochs. One possible reason for this phase difference is that, by assumption, CC networks tend to recruit hidden units that effectively reduce overall error. Thus, in the final solution of the network, all hidden units are expected to contribute to error reduction. Because pruning output weights would eliminate the number of hidden units affecting the global error, it is kept minimal. This finding supports the assumption about hidden unit contribution to overall error. An alternative explanation is also possible. Weights that are trained over a longer period of time tend to distribute their representation more evenly across all activation pathways. In this way, by retraining output weights for many phases before pruning, knowledge of the target task becomes widely distributed. By this process, each output weight thus becomes important in learning the target task and cannot be removed without some consequences to the accuracy of learning and generalization. Input-side weights, on the other hand, get pruned at the end of a single input phase, which has a maximum of 100 epochs. In this way, input-side weights don't get a chance to fully distribute their knowledge across weights, leaving some weights with a much smaller role than others in approximating the objective function. Despite the fact that input-side pruning was more effective in reducing the size of the networks, tests on both cross-validation and novel test

sets revealed that a combination of both input and output weight pruning was necessary to achieve the best novel generalization. This means that both input-side and output-side weights can be responsible for overfitting the data, and should be considered for pruning.

Information in a neural network is often said to be distributed across it's weights. By pruning weights, however, it is essential to assume that information is not completely distributed, in which case all weights would contribute to reducing the global error, and no pruning would be possible. By pruning almost half of the input-side weights of our networks, we were able to show that information was in fact not optimally distributed. In fact, the essential of the information of our networks was contained in 50% of input-side weights plus 99% of the output weights.

Our current results shows that OBD is a promising technique for pruning CC networks. Using early stopping, we pruned a fixed number of weights and tested the network using an independent test set. If error raised on the test set, those weights were restored back. If error lowered, more pruning was performed. One possibility for future research is to set a tolerance factor on the decay of performance on the test set. In other words, if we were willing to tolerate more error on the test set, we might be able to prune significantly more weights than in the current study. Pruning might then become a balanced trade-off between the size of the network and performance on the independent test set.

Empirical results [5] indicate that determining the size of a network using OBD creates networks that never reach acceptable levels of accuracy on some classification problems. On the other hand, techniques that strictly grow networks are often criticized for reaching too much depth and complexity while learning a target task. A technique that combines both growing and pruning provides a solution to this problem and counterbalances too much growing with pruning. The present paper is the first to systematically demonstrate that growing and pruning can be integrated in Cascade-correlation networks, and that doing so is more beneficial than either of these techniques on their own.

AKNOWLEDGMENTS

This research was supported by an FCAR (Québec) scholarship to J.P.T., a Tomlinson scholarship (McGill University) to J.P.T., a grant from FCAR (Canada) to T.R.S., and a grant from NSERC (Québec) to T.R.S. This paper benefited from the comments of anonymous reviewers.

REFERENCES

[1] LeCun, Y., Denker, J., & Solla, S. (1990). Optimal Brain Damage. In D. Touretzky (ed.), Advances in Neural Information Processing Systems, 2, 598-605. San Mateo, CA: Morgan Kaufmann.

[2] Hassibi, B., Stork, D.G. (1993). Second order derivatives for network pruning: Optimal Brain Surgeon. Proceedings of Neural Information Processing Systems, 5, 164-171.

[3] Mozer, MC & Smolensky, P. (1988). Skeletonization: A technique for trimming the fat from a network via relevance assessment. Advances in Neural Information Processing Systems, 1. Morgan Kaufmann. 107-115.

[4] Reed, R. (1993). Pruning algorithms - a survey. IEEE Transactions on Neural Networks 4, 3-16.

[5] Ghosh, J., Tumer, K. (1994). Structural adaptation and generalization in supervised feed-forward networks. Neural Networks, 1, pp. 431-458.

[6] Fiesler, E. (1994). Comparative bibliography of ontogenic neural networks. Proceedings of the International Conference on Artificial Neural Networks, ICANN'94. Sorrento, Italy. pp. 793-796.

[7] Fahlman, S.E., Lebiere, C. (1989). The cascade-correlation learning architecture. Advances in Neural Information Processing Systems, 2, 525-532.

[8] Waugh, S.G. (1995) Extending and benchmarking cascade-correlation. Ph.D. Thesis. University of Tasmania.

[9] Lahnajarvi, J.T., Lehtokangas, M.I., Saarinen, J.P.P. (2002). Evaluation of constructive neural networks with cascaded architecture. Neurocomputing, 48, 573-607.

[10] Rivest, F., & Shultz, T.R. (2002). Application of knowledge-based cascade-correlation to vowel recognition. *IEEE International Joint* Conference on Neural Networks 2002. pp. 53-58.

[11] Shultz, T. R., & Rivest, F. (2001). Knowledge-based cascade correlation: Using knowledge to speed learning. Connection Science, 13, 43-72.

[12] Hertz, Krogh, & Palmer (1991). Introduction to the theory of neural computation. Addison – Wesley Publ. Co, USA.

[13] Hassibi, B., Stork, D.G., & Wolff, G. (1994) Optimal Brain Surgeon: Extensions and performance comparisons. Proceedings of Neural Information Processing Systems, VI, pp. 263-270.

[14] Tresp, V., Neuneier, R., & Zimmermann, H.G. (1997). Early Brain Damage. Advances in Neural Information Processing Systems, 9.

[15] Prechelt, L. (1994). PROBEN1 - A set of benchmarks and benchmarking rules for neural network training algorithms. Technical report 21/94, Fakultät für Informatik, Universität Karlsruhe.