Connection Pruning with Static and Adaptive Pruning Schedules

Lutz Prechelt (prechelt@ira.uka.de)
Fakultät für Informatik
Universität Karlsruhe
D-76128 Karlsruhe, Germany
+49/721/608-4068, Fax: +49/721/694092

August 7, 1996

Accepted article for "Neurocomputing"

Abstract

Neural network pruning methods on the level of individual network parameters (e.g. connection weights) can improve generalization, as is shown in this empirical study. However, an open problem in the pruning methods known today (e.g. OBD, OBS, autoprune, epsiprune) is the selection of the number of parameters to be removed in each pruning step (pruning strength). This work presents a pruning method *lprune* that automatically adapts the pruning strength to the evolution of weights and loss of generalization during training. The method requires no algorithm parameter adjustment by the user. Results of statistical significance tests comparing autoprune, lprune, and static networks with early stopping are given, based on extensive experimentation with 14 different problems. The results indicate that training with pruning is often significantly better and rarely significantly worse than training with early stopping without pruning. Furthermore, lprune is often superior to autoprune (which is superior to OBD) on diagnosis tasks unless severe pruning early in the training process is required.

Keywords: empirical study, pruning, early stopping, generalization

1 Pruning and Generalization

Once an input/output encoding has been chosen, there are several classes of methods for improving the generalization obtained from neural networks. The most important of these are early stopping [5, 11], explicit regularization [5, 12, 17], additive learning [4], and network pruning. All of these techniques try to solve the Bias/Variance dilemma by balancing the representation capability of the network against the information content in the training data [6]. We consider the pruning technique here.

The principal idea of pruning is to reduce the number of free parameters in the network by removing dispensable ones. If applied properly, this approach often reduces overfitting and improves generalization. At the same time it produces a smaller network. Interestingly, many papers on pruning algorithms do show empirically that smaller networks can be obtained without loss of generalization, but do not show that generalization will often be *improved* compared to reasonable static-network training methods; [5, 9] are notable exceptions.

Pruning methods usually either remove complete input or hidden nodes along with all their associated parameters or remove individual connections, each of which carries one free parameter (the *weight*). This latter approach is very fine-grained and makes pruning particularly powerful.

1.1 Some Known Pruning Methods

The key to pruning is a method to calculate the approximate importance of each parameter. Several such methods have been suggested. The simplest one — with obvious flaws [5] — is to assume the importance to be proportional to the magnitude of a weight. More sophisticated approaches are the well-known *optimal brain damage* (OBD) and *optimal brain surgeon* (OBS) methods. OBD [2] uses an approximation to the second derivative of the error with respect to each weight to determine the *saliency* of the removal of that weight. Low saliency means low importance of a weight. OBS [8, 9] avoids the drawbacks of the approximation by computing the second derivatives (almost) exactly. However, even when using accelerations based on Shur's lemma OBS is computationally expensive.

Both methods have the disadvantage of requiring training to the error minimum before pruning may occur. For many problems, this introduces massive overfitting which often cannot be repaired by subsequent pruning. The *autoprune* method [5] avoids this problem. Its weight importance coefficients are defined by a test statistic T for the assumption that a weight becomes zero during the training process:

$$T(w_i) = \log \left(\frac{\left| \sum_p w_i - \eta \, (\partial E / \partial w_i)_p \right|}{\eta \sqrt{\sum_p ((\partial E/\partial w_i)_p - \overline{(\partial E/\partial w_i)})^2}} \right)$$

In contrast to OBD and OBS, this measure does not assume an error minimum has been reached; it can be computed at any time during training. In the above formula, sums are over all examples p of the training set, η is the learning rate, and the overline means arithmetic mean over the examples. A large value of T indicates high importance of the connection with weight w_i. Connections with small T can be pruned. Finnoff et al. [5] have convincingly shown autoprune to be superior to OBD.

Note that many more pruning methods than discussed here have been proposed in the literature. In particular, Bayesian methods can unify the notions of regularization and pruning [18].

1.2 An Open Problem: How Much To Prune?

Given the importance T of each weight at any time during training, two questions remain to be answered:

1. When should we prune?
2. How many connections should be removed in the next pruning step?

The first question is simple to answer: For OBD and OBS, pruning occurs when minimum training set error has been reached. For autoprune, pruning occurs when overfitting begins (here: when the validation set error increased twice during training; see below).

The second question, however, has not yet been answered satisfactorily. The authors of OBD suggest to delete *"some"* parameters. The authors of autoprune at least suggest a concrete pruning schedule: remove 35% of all parameters in the first pruning step and 10% in each following step. Such rules of thumb, however, are not satisfying, because obviously they cannot always be optimal. The following section presents a pruning method, called *lprune*, based on autoprune that tries to solve the problem.

It computes the pruning schedule dynamically during training, adapting to the evolution of the weights and to the amount of overfitting observed. Subsequent sections present and discuss a set of empirical results obtained for lprune.

2 Adaptive Pruning: The lprune Method

2.1 Observations

The lprune method is not based on a theory of weight development, because no such theory is currently available. Instead, it builds on a number of observations made for the distribution of the T coefficients during training:

1. The distribution of the values is roughly normal.
2. During training, both the mean μ_T and the variance σ_T of the distribution tend to increase.
3. When pruning occurs, the variance suddenly drops and the mean suddenly rises.
4. Afterwards, the variance increases again and the mean decreases again. After a while, normal development continues as in (2) above.

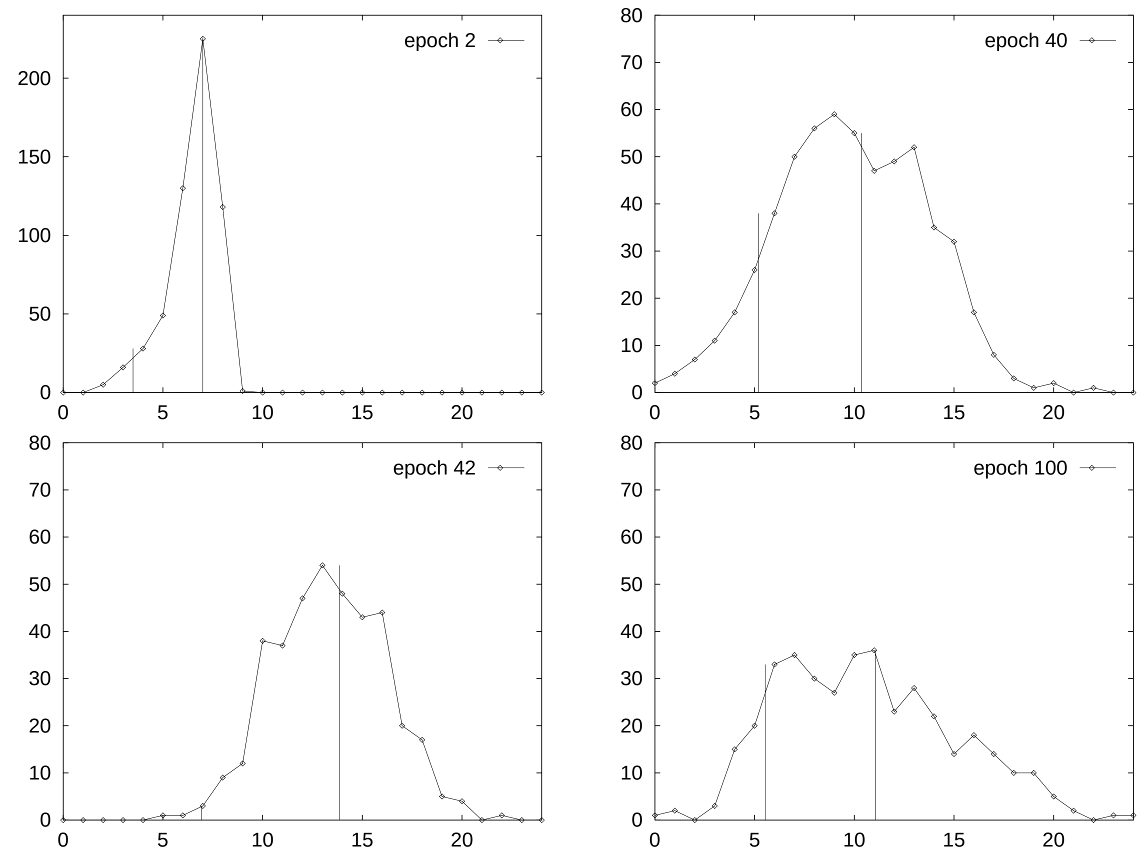


Figure 1: Four pruning coefficient histograms from the same training run (glass1 problem with standard architecture) in epochs 2, 40, 42, and 100. Horizontal axis: coefficient size, grouped in classes of width 1. Vertical axis: absolute frequency of weights with this coefficient size. The right vertical line is the arithmetic mean of the coefficient sizes, the left vertical line is 0.5 times that. The area under the curve left of the left line thus indicates what would be pruned at that point for $\lambda = 0.5$ when a pruning step would occur. In the run shown, pruning occurred after epoch 40.

See Figure 1 for an example of this behavior. The observations suggest that a certain fraction of the mean of the coefficient distribution can be used as a threshold for pruning. All connections whose T is below the threshold are pruned. Early during training the fraction of these connections is rather small, because the variance is small. Once the weights have evolved and differentiated, the variance is larger and it is safe to prune a larger fraction of the connections. After a pruning step, immediate further pruning should remove only a few connections, if any, since the remaining weights have to differentiate again before the important ones can safely be distinguished from the less important ones. This reduction of pruning strength is ensured by the reduced variance after pruning and is further pronounced immediately after pruning due to the larger mean of the distribution (see epoch 42 vs. 40 in Figure 1).

2.2 Approach

From these observations, the following rule seems reasonable for determining how many connections to prune:

At each pruning step, prune all those connections i whose weights w_i satisfy
$T(w_i) < \lambda \mu_T$ for some $\lambda \in [0 \ldots 1]$

However, no fixed value of λ results in good adaptation of pruning strength; we have to choose λ dynamically as well. The key factor for determining an appropriate value of λ is the amount of overfitting that is observed for the network. The higher the overfitting, the more should be pruned, and the higher λ must be.

2.3 Definitions

To formalize this notion we define "overfitting" quantitatively, as well as some other concepts that can be used to express criteria for stopping or triggering pruning.

Let E be the objective function (error function) of the training algorithm, for instance the squared error. Then $E_{tr}(t)$ is the average error per example over the training set, measured after epoch t. $E_{va}(t)$ is the error on the validation set and is used to determine overfitting. $E_{te}(t)$ is the error on the test set; it is not known to the training algorithm but characterizes the quality of the network resulting from training.

The value $E_{opt}(t)$ is defined to be the lowest validation set error obtained in epochs up to t:

$$E_{opt}(t) := \min_{t' \leq t} E_{va}(t')$$

Now we define the *generalization loss* at epoch t to be the relative increase of the validation error over the minimum-so-far (in percent):

$$GL := GL(t) := 100 \cdot \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right)$$

The generalization loss directly characterizes the amount of overfitting.

A high generalization loss is one candidate reason to stop training or to perform a pruning step. This leads us to a class of triggering criteria: Stop or prune as soon as the generalization loss exceeds a certain threshold. We define the class GL_α as

$$GL_\alpha : \text{ satisfied after first epoch } t \text{ with } GL(t) > \alpha$$

However, we might want to suppress stopping or pruning if the training is still progressing very rapidly. When the training error still drops quickly, generalization losses may have a higher chance to be "repaired". To formalize this notion we define a *training strip of length* k to be a sequence of k epochs numbered $n+1 \ldots n+k$ where n is divisible by k. The training *progress* (in per thousand) measured after such a training strip is then

$$P_k(t) = 1000 \cdot \left(\frac{\sum_{t'=t-k+1}^{t} E_{tr}(t')}{k \cdot \min_{t'=t-k+1}^{t} E_{tr}(t')} - 1 \right)$$

that is, "how much was the average training error during the strip larger than the minimum training error during the strip?"

The progress is guaranteed to approach zero in the long run unless the training is globally unstable (e.g. oscillating). Alternatively, we could use the variance of E_{tr} to measure progress. In the following we will always assume strips of length 5 (i.e., $k = 5$) and measure the cross validation error only at the end of each strip.

Another class of triggering criteria relies only on the sign of the changes in the generalization error. These criteria say "stop or prune when the generalization error increased in s successive strips":

$$UP_s : \text{ satisfied after epoch } t \text{ iff } UP_{s-1} \text{ was satisfied after epoch } t-k \text{ and } E_{va}(t) > E_{va}(t-k)$$

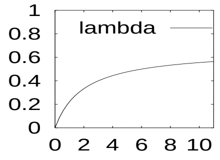
$$UP_1 : \text{ satisfied after first end-of-strip epoch } t \text{ with } E_{va}(t) > E_{va}(t-k)$$

(The name UP is a mnemonic for "error went *up*".) This class of criteria is independent of E_{opt}. It is preferable over GL_α for triggering pruning steps, because pruning always results in an intermediate increase of GL. The next pruning step should however occur not when GL is high, but when it increases.

2.4 Algorithm

Initial experiments showed that an appropriate way to adapt λ is to increase it with growing GL, saturating at some maximum value. This leads to the following adaptation rule for λ:

$$\lambda := \lambda(GL) := \lambda_{max}\left(1 - \frac{1}{1 + \frac{GL}{\alpha}}\right) \qquad \begin{array}{rcl} \lambda_{max} & := & 2/3 \\ \alpha & := & 2 \end{array}$$



The given values of λ_{max} and α were found by a small number of experiments with 4 of the 42 example problems used below. These parameters are only moderately critical and the values given here are certainly not exactly optimal.

The complete *lprune* algorithm ('lambda-prune') can now be formulated as

```
REPEAT
  Train network for one epoch;
  IF epoch number MOD k = 0 THEN
    Compute $E_{va}$, $E_{opt}$, and $GL$ using the validation set;
  END;
UNTIL $GL > 5$; (* i.e., apply normal early stopping *)
Reset network to the state that exhibited $E_{opt}$;
(* Now begin training with pruning: *)
REPEAT
  Train network for one epoch and compute $T(w_i)$ values;
  IF epoch number MOD k = 0 THEN
    Compute $E_{va}$, $E_{opt}$, and $GL$ using the validation set;
    IF $UP_2(t)$ satisfied AND no pruning k epochs ago THEN
      Prune all connections $i$ whose weights $w_i$ satisfy   $T(w_i) < \lambda(GL)\,\mu_T$;
    END;
  END;
UNTIL $t > 5000$ OR $P_5(t) < 0.1$ OR
      (At Least 25 Epochs trained since last pruning AND $GL > 100$ AND $P_5(t) < 0.4$)
```

The constants 5000, 0.1, 25, 100, and 0.4 are not critical and make a conservative stopping criterion for the whole process. The result of the training is the network that exhibited the lowest validation error E_{opt}. Note that in order to implement this scheme, only one network duplicate is needed.

3 Results And Discussion

This section first describes the experimental setup used and the overall behavior observed for the pruning algorithms. It then presents and interprets the results of a quantitative comparison of the algorithms.

3.1 Experiment Setup

Extensive benchmark comparisons were made between autoprune, lprune, and static backpropagation with early stopping. 14 different problems were used, all from the PROBEN1 benchmark set [13], a collection of diagnosis problems. Most of these problems stem from the UCI machine learning databases archive. The problems have between 8 and 120 inputs, between 1 and 19 outputs, and between 214 and 7200 examples. 9 of the problems are classification tasks using 1-of-n output encoding (*cancer, card, diabetes, gene, glass, heart, heartc, horse, soybean*, and *thyroid*), 4 are approximation tasks (*building, flare, hearta*, and *heartac*); all problems are real datasets from realistic application domains.

All runs were done using the RPROP weight update rule [15], squared error function, and the RPROP parameters $\eta^{+} = 1.2$, $\eta^{-} = 0.5$, $\Delta_0 \in [0.05 \ldots 0.2]$ randomly per weight, $\Delta_{max} = 50$, $\Delta_{min} = 0$, initial weights from [-0.1…0.1] randomly.[1] RPROP is a fast backpropagation variant that is about as fast as quickprop [3] but more robust in the choice of parameters. Its basic idea is to adapt the step size taken for each weight separately; the adaptation is independent of the size of the gradient for that weight and depends only on its sign: as long as the sign is the same, the step size is multiplied by

[1]Note that RPROP requires a modification in the way the $T(w_i)$ are computed, because the weight change is not proportional to $\partial E / \partial w_i$.

η^+, when the sign changes the step size is multiplied by η^-. RPROP requires epoch learning, i.e., the weights are updated only once per epoch.

In three different random ways, the examples of each problem were partitioned into training set (50%), validation set (25%), and test set (25% of examples), resulting in 42 datasets (*cancer1, cancer2, cancer3, card1, card2, card3* etc.). Each of these datasets was trained with two different initial topologies. The first is the dataset's *standard architecture* network topology, which is found by a coarse trial-and-error search and can be considered to be a "reasonable" topology for the dataset. The topologies have between 2 and 32 hidden nodes, either one or two hidden layers, and contain all possible feedforward connections, not only those from one layer to the next. The derivation of the standard architectures is described in [13][2]. The second is the *noshortcut standard architecture*, which is derived from the standard architecture by excluding all connections that do not go from one layer to the *immediately* following layer. For each of the 42 datasets and each of the two network topologies for each dataset, 30 runs were made with autoprune, 30 with lprune, and 30 with backpropagation with early stopping using the GL_5 stopping criterion; more than 7500 runs overall.

After each of these runs, the error E_{te} of the resulting network was measured.[3] For each dataset, the autoprune sample of 30 such test set errors was compared to the corresponding lprune sample and the backprop sample using the t-test. For correct application of the t-test, it was necessary to take the logarithm of the test error (since the test errors have log-normal distribution), to remove a few outliers above and below (1.1% of all runs for autoprune and 2.0% for lprune), and to apply the Cochran/Cox correction for the unequal variances case. The results of the tests are shown in Tables 1 to 4.

3.2 Qualitative Behavior

Each significant pruning step leads to a large sudden increase of GL, followed by a rapid decrease. Whether the decrease leads to a lower or higher GL than before pruning depends on whether the pruning occurred at the right time and in the right strength. To employ the UP_2 triggering criterion means to accept the view that pruning should occur whenever a substantial deterioration of generalization behavior (as measured noisily by the validation set error) begins. It would probably be better in some cases to wait longer before pruning, because during certain phases in training overfitting occurs but vanishes automatically later. It is not at all clear, however, how such a situation should be detected at its beginning. Therefore, UP_2 seems to be a reasonable way to determine *when* to prune. The pruning strength of autoprune, however, is often not appropriate.

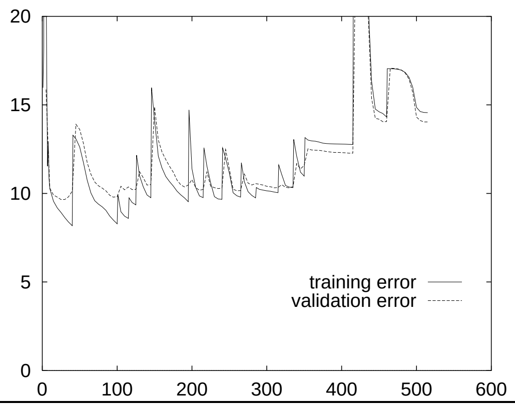


Figure 2: Development of training and validation set error over time for the same run of autoprune from which the figures above were derived. x-axis: number of epoch, y-axis: Error. There are 15 pruning steps after which 15% of the initial connections remain. In this example pruning is not successful: no lower validation set error occurs than before the first pruning step. This is not a rare case.

Since early stopping is performed as the first phase of the pruning algorithm (for both autoprune and lprune), these training methods take significantly, but not prohibitively longer than training with

[2]In that reference, the standard architectures are called *pivot architectures.*

[3]Caveat: In the experiments, the data of the validation set was never used for actual gradient training. In a real application, one would not want to waste valuable data points in this manner.

static networks and early stopping. In the setup chosen, typically three to five times as many epochs are trained. However, epochs after pruning consume less time, since the network is smaller and the total number of epochs could be reduced by using a faster stopping criterion than the extremely conservative one chosen in the given setup.

In the example autoprune run shown in Figure 2, the network tolerates the 35% pruning of the first pruning step, yet is ruined by the second pruning step many epochs later, which removes only 10% of the weights. Towards the end of the training run, the network is always overpruned, since such a conservative stopping criterion is used.

For lprune, the situation is a bit different. As long as overfitting is only moderate, the pruning strength is usually small. The same is true when the weights have not yet sufficiently evolved since the last pruning step or since the beginning of training. On the other hand, when overfitting is large, pruning can be quite severe in lprune.

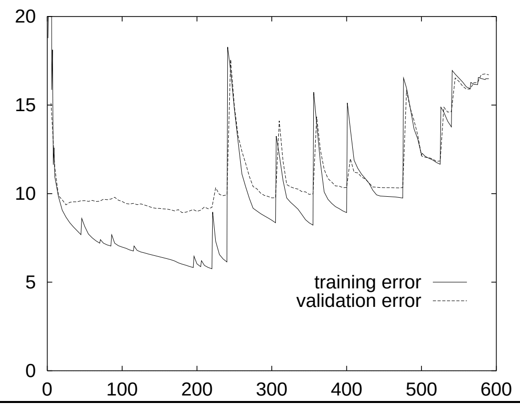


Figure 3: The corresponding curves for lprune. There are 21 pruning steps, initially removing only few connections, later removing more in each step. Finally, 0.5% of the initial connections remain. Pruning is successful: after 7 pruning steps (in epoch 180) a lower validation set error is reached than before the first pruning step.

In the example lprune run shown in Figure 3, this behavior leads to several small pruning steps (the first four remove 2%, 3%, 3%, and 5% of the connections, respectively) that manage to keep overfitting low over a longer training period and finally reduce the validation error. In this example, lprune is superior to autoprune.

The behavior observed in this example is not prototypical, though. Very different error curves and pruning sequences occur as well. However, one observation prevails: pruning with a static schedule sometimes destroys the generalization ability of the network unnecessarily. In the case of the schedule used in autoprune this is usually because of too heavy or too fast pruning. Significantly lower pruning strengths could avoid this, but would exhibit another problem: namely that overfitting cannot be reduced as fast as it builds up. Therefore, pruning with very small pruning strength and static schedule would probably be similar to OBD, which has been shown inferior to autoprune by [5]. Adaptive pruning schedules are clearly necessary.

3.3 Quantitative Results

As we see in Tables 1 and 2, lprune is better in some cases and autoprune is better in others. For 2 of the 14 problems, there is never a significant difference. How *often* autoprune is better than lprune and vice versa, depends on the particular selection of datasets and should thus not be overemphasized. However, there is a pattern in the results: autoprune tends to be better for problems that have overly large networks, for instance the gene problems that have 120 input units — and the difference is larger with shortcut connections than without. On the other hand, lprune is often better when pruning is delicate, for instance for the building, glass, and thyroid problems that have only 14, 9, and 21 inputs, respectively.

standard architectures

Problem	1	2	3
building	L 0.0	—	—
cancer	—	—	—
card	—	—	—
diabetes	—	A 2.5	L 0.9
flare	—	A 7.3	—
gene	A 0.0	A 0.0	A 0.0
glass	—	L 2.3	L 1.8
heart	A 5.5	A 0.4	—
hearta	—	A 0.1	—
heartac	—	—	—
heartc	—	—	A 2.5
horse	—	—	A 3.4
soybean	—	—	—
thyroid	—	L 6.2	L 0.3

noshortcut standard architectures

Problem	1	2	3
building	L 0.3	—	L 3.7
cancer	—	A 0.9	—
card	—	—	—
diabetes	—	A 4.3	—
flare	A 0.8	A 1.0	A 0.0
gene	A 5.4	L 6.6	A 1.7
glass	A 7.6	—	—
heart	—	A 3.3	—
hearta	—	—	—
heartac	—	—	—
heartc	—	—	—
horse	—	—	A 1.5
soybean	L 7.0	—	—
thyroid	—	—	L 0.0

Tables 1 and 2: Comparison of autoprune ("A") to lprune ("L") using the standard architectures and noshortcut standard architectures, respectively. Compares test set errors E_{te} for variants 1, 2, 3 of each problem. The entries show differences (in samples of 30 runs each) that are statistically significant on a 10% level and the corresponding p-values (in percent). Low p-values indicate high significance. The letter indicates which algorithm is better; a dash means that no significant difference was found.
Table 1 (standard architectures): 26 times no significant difference, 10 times A better, 6 times L better.
Table 2 (noshortcut standard architectures): 27 times no significant difference, 10 times A better, 5 times L better.

An explanation of this effect is that lprune is unable to perform heavy pruning very early during training when overfitting is only small. However, such heavy pruning is what would be needed to perform well on e.g. the gene problems and it is what autoprune does. On the other hand, the static pruning schedule of autoprune is too rigid. It prunes too much in situations where waiting for further weight differentiation is required despite the fact that overfitting has begun. Such situations are recognized by lprune and its pruning removes only very few weights, sometimes even none at all. Thus, lprune solves a part of the pruning schedule problem, namely adapting pruning strength to the stage of development of the weights. The rest of the problem is still unsolved, namely determining the absolute number of weights that should be pruned.

In a second series of benchmarks, pruning was compared to training a static network with early stopping without pruning, using the same setup as before. The results are shown in Tables 3 and 4. We see that pruning indeed usually does improve generalization significantly; a fact that is often not properly recognized. Therefore, pruning algorithms are preferable over static networks, at least in applications where small improvements of generalization do matter. This is particularly true if one uses networks with very many parameters (as is often recommended for the early stopping method): without shortcut connections, backprop is significantly better than pruning in eight of the cases (Table 4), whereas with the shortcut connections this value drops to just two (Table 2).

4 Conclusion

Extensive benchmarking compared adaptive and non-adaptive pruning and backprop without pruning. For the former, a method for adaptive calculation of pruning strength for connection pruning algorithms was described. It represents a partial solution to an open problem in network pruning, standard architectures

Problem	1	2	3
building	(A 0.0)	—	—
cancer	—	B 3.1	B 9.9
card	—	A 0.0	A 2.2
diabetes	—	A 4.0	—
flare	A 0.0	A 0.0	A 0.0
gene	A 0.0	(A 0.0)	(A 0.0)
glass	A 8.6	A 2.3	A 0.1
heart	—	—	—
hearta	—	A 2.0	—
heartac	—	—	—
heartc	—	—	A 0.0
horse	—	A 0.4	A 0.1
soybean	—	—	—
thyroid	A 0.4	—	—

noshortcut standard architectures

Problem	1	2	3
building	(A 0.0)	—	B 7.9
cancer	—	—	B 0.1
card	—	A 0.0	A 7.1
diabetes	—	A 6.1	—
flare	—	A 0.0	A 0.3
gene	A 1.1	—	(A 0.4)
glass	—	—	—
heart	B 0.2	—	—
hearta	B 2.4	A 3.4	A 0.5
heartac	—	B 9.2	(A 5.4)
heartc	—	B 2.4	A 1.3
horse	B 4.7	—	B 8.6
soybean	—	(A 1.7)	—
thyroid	A 0.0	A 0.1	A 2.2

Tables 3 and 4: Comparison of autoprune ("A") to backprop with early stopping ("B"). Analogous to Tables 1 and 2 above.
Table 3 (standard architectures): 22 times no significant difference, 18 times A better (3 times slightly dubious due to non-normal backprop samples), 2 times B better.
Table 4 (noshortcut standard architectures): 18 times no significant difference, 16 times A better (4 times slightly dubious), 8 times B better.

determining pruning strength. The following conclusions apply to the class of learning tasks covered by the experiments:

1. Training with pruning very often results in better networks than training with early stopping without pruning, but rarely results in worse networks. Thus, pruning methods should be used more often than they are used today.
2. The automatic pruning strength adaptation of the lprune method can result in better networks than pruning with static pruning schedules. This is true in particular for small networks.
3. However, the lprune solution to the pruning strength problem is only partial, because lprune is unable to execute severe pruning in early training stages as it is sometimes needed, in particular for networks with overly many inputs.
4. As the very different results for the various problems and even for the dataset permutations show, benchmarking has to be extensive and careful in order to yield significant and correct results — this is in sharp contrast to the state of the practice [14].

Acknowledgements

Thanks to Michael Philippsen, Rainer Storn, Ben Gomes, Alexander Linden, and the anonymous reviewers for commenting on drafts of this article.

Availability of Raw Data

The datasets used in the experiments are from the Proben1 benchmark collection, which is available for anonymous ftp from ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz and also from ftp://ftp.cs.cmu.edu/afs/cs/project/connect/bench/contrib/prechelt/proben1.tar.gz

The raw result data obtained in the experiments (one record per program run) is available electronically, too. Use either http://wwwipd.ira.uka.de/~prechelt/nndata.html or ftp://ftp.ira.uka.de/pub/neuron/nndata.tar.gz.

References

[1] Jack D. Cowan, Gerald Tesauro, and J. Alspector, editors. *Advances in Neural Information Processing Systems 6*, San Mateo, CA, 1994. Morgan Kaufman Publishers Inc.

[2] Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *[16]*, pages 598–605, 1990.

[3] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, September 1988.

[4] Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation learning architecture. In *[16]*, pages 524–532, 1990.

[5] William Finnoff, Ferdinand Hergert, and Hans Georg Zimmermann. Improving model selection by nonconvergent methods. *Neural Networks*, 6:771–783, 1993.

[6] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[7] Stephen J. Hanson, Jack D. Cowan, and C. Lee Giles, editors. *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufman Publishers Inc.

[8] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *[7]*, pages 164–171, 1993.

[9] Babak Hassibi, David G. Stork, Gregory Wolff, and Takahiro Watanabe. Optimal Brain Surgeon: Extensions and performance comparisons. In *[1]*, 1994.

[10] Richard P. Lippmann, John E. Moody, and David S. Touretzky, editors. *Advances in Neural Information Processing Systems 3*, San Mateo, CA, 1991. Morgan Kaufman Publishers Inc.

[11] N. Morgan and H. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *[16]*, pages 630–637, 1990.

[12] Steven J. Nowlan and Geoffrey E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.

[13] Lutz Prechelt. PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, Germany, September 1994. Anonymous FTP: /pub/papers/techreports/1994/1994-21.ps.gz on ftp.ira.uka.de.

[14] Lutz Prechelt. A quantitative study of experimental evaluations of neural network learning algorithms: Current research practice. *Neural Networks*, 9(3):457–462, April 1996.

[15] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. of the IEEE Intl. Conf. on Neural Networks*, pages 586–591, San Francisco, CA, April 1993.

[16] David S. Touretzky, editor. *Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990. Morgan Kaufman Publishers Inc.

[17] Andreas S. Weigend, David E. Rumelhart, and Bernardo A. Huberman. Generalization by weight-elimination with application to forecasting. In *[10]*, pages 875–882, 1991.

[18] Peter M. Williams. Bayesian regularization and pruning using a Laplace prior. Technical Report CSRP-312, School of Cognitive and Computing Sciences, University of Sussex, Brighton, England, February 1994. ftp://ftp.cogs.susx.ac.uk/pub/reports/csrp/csrp312.ps.Z.