Early Stopping — but when?

Lutz Prechelt (prechelt@ira.uka.de)

Fakultät für Informatik; Universität Karlsruhe
D-76128 Karlsruhe; Germany

Abstract. Validation can be used to detect when overfitting starts during supervised training of a neural network; training is then stopped before convergence to avoid the overfitting ("early stopping"). The exact criterion used for validation-based early stopping, however, is usually chosen in an ad-hoc fashion or training is stopped interactively. This trick describes how to select a stopping criterion in a systematic fashion; it is a trick for either speeding learning procedures or improving generalization, whichever is more important in the particular situation. An empirical investigation on multi-layer perceptrons shows that there exists a tradeoff between training time and generalization: From the given mix of 1296 training runs using different 12 problems and 24 different network architectures I conclude slower stopping criteria allow for small improvements in generalization (here: about 4% on average), but cost *much* more training time (here: about factor 4 longer on average).

1 Early stopping is not quite as simple

1.1 Why early stopping?

When training a neural network, one is usually interested in obtaining a network with optimal generalization performance. However, all standard neural network architectures such as the fully connected multi-layer perceptron are prone to overfitting [10]: While the network *seems* to get better and better, i.e., the error on the training set decreases, at some point during training it actually begins to get worse again, i.e., the error on unseen examples increases. The idealized expectation is that during training the generalization error of the network evolves as shown in Figure 1. Typically the generalization error is estimated by a validation error, i.e., the average error on a *validation set*, a fixed set of examples not from the training set.

There are basically two ways to fight overfitting: reducing the number of dimensions of the parameter space or reducing the effective size of each dimension. Techniques for reducing the number of parameters are greedy constructive learning [7], pruning [5, 12, 14], or weight sharing [18]. Techniques for reducing the size of each parameter dimension are regularization, such as weight decay [13] and others [25], or early stopping [17]. See also [8, 20] for an overview and [9] for an experimental comparison.

Early stopping is widely used because it is simple to understand and implement and has been reported to be superior to regularization methods in many cases, e.g. in [9].

1.2 The basic early stopping technique

In most introductory papers on supervised neural network training one can find a diagram similar to the one shown in Figure 1. It is claimed to show the evolution over time of the per-example error on the training set and on a validation set not used for training (the *training error curve* and the *validation error curve*). Given this behavior, it is clear how to do early stopping using validation:

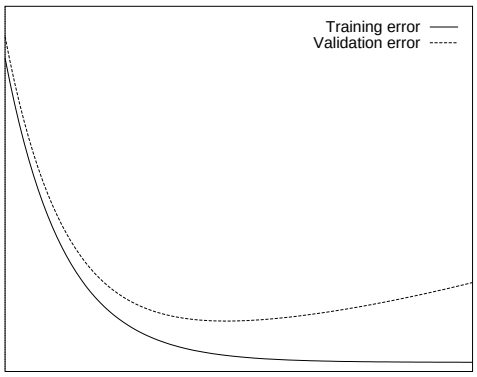

Fig. 1. Idealized training and validation error curves. Vertical: errors; horizontal: time

1. Split the training data into a training set and a validation set, e.g. in a 2-to-1 proportion.
2. Train only on the training set and evaluate the per-example error on the validation set once in a while, e.g. after every fifth epoch.
3. Stop training as soon as the error on the validation set is higher than it was the last time it was checked.
4. Use the weights the network had in that previous step as the result of the training run.

This approach uses the validation set to anticipate the behavior in real use (or on a test set), assuming that the error on both will be similar: The validation error is used as an estimate of the generalization error.

1.3 The uglyness of reality

However, for real neural network training the validation set error does not evolve as smoothly as shown in Figure 1, but looks more like in Figure 2. See Section 4 for a rough explanation of this behavior. As we see, the validation error can still go further down after it has begun to increase — plus in a realistic setting we do never know the exact generalization error but estimate it by the validation set error instead. There is no obvious rule for deciding when the minimum of

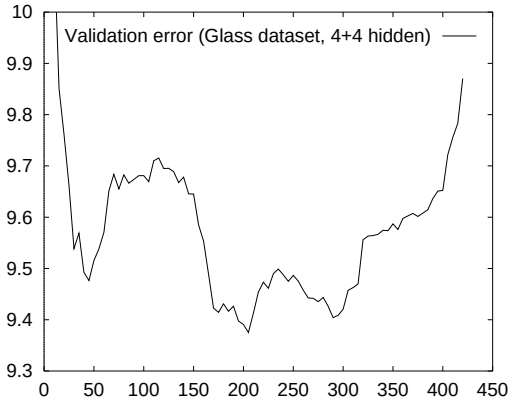

Fig. 2. A real validation error curve. Vertical: validation set error; horizontal: time (in training epochs).

the generalization error is obtained. Real validation error curves almost always have more than one local minimum. The above curve exhibits as many as 16 local minima before severe overfitting begins at about epoch 400. Of these local minima, 4 are the global minimum up to where they occur. The optimal stopping point in this example would be epoch 205. Note that stopping in epoch 400 compared to stopping shortly after the first "deep" local minimum at epoch 45 trades an about sevenfold increase of learning time for an improvement of validation set error by 1.1% (by finding the minimum at epoch 205). If representative data is used, the validation error is an unbiased estimate of the actual network performance; so we expect a 1.1% decrease of the generalization error in this case. Nevertheless, overfitting might sometimes go undetected because the validation set is finite and thus not perfectly representative of the problem.

Unfortunately, the above or any other validation error curve is not *typical* in the sense that all curves share the same qualitative behavior. Other curves might never reach a better minimum than the first, or than, say, the third; the mountains and valleys in the curve can be of very different width, height, and shape. The only thing all curves seem to have in common is that the differences between the first and the following local minima are not huge.

As we see, choosing a stopping criterion predominantly involves a tradeoff between training time and generalization error. However, some stopping criteria may typically find better tradeoffs that others. This leads to the question of which criterion to use with cross validation to decide when to stop training. This is why we need the present trick: To tell us how to *really* do early stopping.

2 How to do early stopping best

What we need is a predicate that tells us when to stop training. We call such a predicate a *stopping criterion*. Among all possible stopping criteria we are

searching for those which yield the lowest generalization error and also for those with the best "price-performance ratio", i.e., that require the least training for a given generalization error or that (on average) result in the lowest generalization error for a certain training time.

2.1 Some classes of stopping criteria

There are a number of plausible stopping criteria and this work considers three classes of them. To formally describe the criteria, we need some definitions first. Let E be the objective function (error function) of the training algorithm, for example the squared error. Then $E_{tr}(t)$, the training set error (for short: training error), is the average error per example over the training set, measured after epoch t. $E_{va}(t)$, the validation error, is the corresponding error on the validation set and is used by the stopping criterion. $E_{te}(t)$, the test error, is the corresponding error on the test set; it is not known to the training algorithm but estimates the generalization error and thus benchmarks the quality of the network resulting from training. In real life, the generalization error is usually unknown and only the validation error can be used to estimate it.

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

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

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):

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

High generalization loss is one obvious candidate reason to stop training, because it directly indicates overfitting. This leads us to the **first class of stopping criteria: stop as soon as the generalization loss exceeds a certain threshold.** We define the class GL_α as

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

However, we might want to suppress stopping if the training is still progressing very rapidly. The reasoning behind this approach is that when the training error still decreases quickly, generalization losses have higher chance to be "repaired"; we assume that often overfitting does not begin until the error decreases only slowly. 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?" Note that this progress measure is

high for unstable phases of training, where the training set error goes up instead of down. This is intended, because many training algorithms sometimes produce such "jitter" by taking inappropriately large steps in weight space. The progress measure is, however, guaranteed to approach zero in the long run unless the training is globally unstable (e.g. oscillating).

Now we can define the **second class of stopping criteria: use the quotient of generalization loss and progress.**

$$PQ_\alpha: \text{ stop after first end-of-strip epoch } t \text{ with } \frac{GL(t)}{P_k(t)} > \alpha$$

In the following we will always assume strips of length 5 and measure the validation error only at the end of each strip.

A completely different kind of stopping criterion relies only on the sign of the changes in the generalization error. We define the **third class of stopping criteria: stop when the generalization error increased in s successive strips.**

UP_s : stop after epoch t iff UP_{s-1} stops after epoch $t-k$ and
$E_{va}(t) > E_{va}(t-k)$

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

The idea behind this definition is that when the validation error has increased not only once but during s consecutive strips, we assume that such increases indicate the beginning of final overfitting, independent of how large the increases actually are. The UP criteria have the advantage of measuring change locally so that they can be used in the context of pruning algorithms, where errors must be allowed to remain much higher than previous minima over long training periods.

None of these criteria alone can guarantee termination. We thus complement them by the rule that training is stopped when the progress drops below 0.1 or after at most 3000 epochs.

All stopping criteria are used in the same way: They decide to stop at some time t during training and the result of the training is then the set of weights that exhibited the lowest validation error $E_{opt}(t)$. Note that in order to implement this scheme, only one duplicate weight set is needed.

2.2 The trick: Criterion selection rules

These three classes of stopping criteria GL, UP, and PQ were evaluated on a variety of learning problems as described in Section 3 below. The results indicate that "slower" criteria, which stop later than others, on the average lead to improved generalization compared to "faster" ones. However, the training time that has to be expended for such improvements is rather large on average and also varies dramatically when slow criteria are used. The systematic differences between the criteria *classes* are only small.

For training setups similar to the one used in this work, the following rules can be used for selecting a stopping criterion:

1. Use fast stopping criteria unless small improvements of network performance (e.g. 4%) are worth large increases of training time (e.g. factor 4).
2. To maximize the probability of finding a "good" solution (as opposed to maximizing the average quality of solutions), use a GL criterion.
3. To maximize the average quality of solutions, use a PQ criterion if the network overfits only very little or an UP criterion otherwise.

3 Where and how well does this trick work?

As no mathematical analysis of the properties of stopping criteria is possible today (see Section 4 for the state of the art), we resort to an experimental evaluation.

We want to find out which criteria will achieve how much generalization using how much training time on which kinds of problems. To achieve broad coverage, we use 12 different network topologies, 12 different learning tasks, and 14 different stopping criteria. To keep the experiment feasible, only one training algorithm is used.

3.1 Concrete questions

To derive and evaluate the stopping criteria selection rules presented above we need to answer the following questions:

1. *Training time:* How long will training take with each criterion, i.e., how *fast* or *slow* are they?
2. *Efficiency:* How much of this training time will be redundant, i.e., will occur after the to-be-chosen validation error minimum has been seen?
3. *Effectiveness:* How good will the resulting network performance be?
4. *Robustness:* How sensitive are the above qualities of a criterion to changes of the learning problem, network topology, or initial conditions?
5. *Tradeoffs:* Which criteria provide the best time-performance tradeoff?
6. *Quantification:* How can the tradeoff be quantified?

The answers will directly lead to the rules already presented above in Section 2.2. To find the answers to the questions we record for a large number of runs when each criterion would stop and what the associated network performance would be.

3.2 Experimental setup

Approach: To measure network performance, we partition each dataset into two disjoint parts: *Training data* and *test data*. The training data is further subdivided into a *training set* of examples used to adjust the network weights and a *validation set* of examples used to estimate network performance during training as required by the stopping criteria. The validation set is never used for weight adjustment. This decision was made in order to obtain pure stopping

criteria results. In contrast, in a real application one would include the validation set examples in the training set and retrain from scratch after a reasonable stopping time has been computed.

Stopping criteria: The stopping criteria examined were GL_1, GL_2, GL_3, GL_5, $PQ_{0.5}$, $PQ_{0.75}$, PQ_1, PQ_2, PQ_3, UP_2, UP_3, UP_4, UP_6, and UP_8. All criteria where evaluated simultaneously, i.e., each single training run returned one result for each of the criteria. This approach reduces the variance of the estimation.

Learning tasks: Twelve different problems were used, all from the PROBEN1 NN benchmark set [19]. All problems are real datasets from realistic application domains; they form a sample of a broad class of domains, but none of them exhibits extreme nonlinearity. The problems have between 8 and 120 inputs, between 1 and 19 outputs, and between 214 and 7200 examples. All inputs and outputs are normalized to range $0 \ldots 1$. Nine of the problems are classification tasks using 1-of-n output encoding (*cancer, card, diabetes, gene, glass, heart, horse, soybean*, and *thyroid*), three are approximation tasks (*building, flare*, and *hearta*).

Datasets and network architectures: The examples of each problem were partitioned into training (50%), validation (25%), and test set (25% of examples) in three different random ways, resulting in 36 datasets. Each of these datasets was trained with 12 different feedforward network topologies: one hidden layer networks with 2, 4, 8, 16, 24, or 32 hidden nodes and two hidden layer networks with 2+2, 4+2, 4+4, 8+4, 8+8, or 16+8 hidden nodes in the first+second hidden layer, respectively; all these networks were fully connected including all possible shortcut connections. For each of the network topologies and each dataset, two runs were made with linear output units and one with sigmoidal output units using the activation function $f(x) = x/(1 + |x|)$.

Training algorithm: All runs were done using the RPROP training algorithm [21] using the squared error function and the parameters $\eta^+ = 1.1$, $\eta^- = 0.5$, $\Delta_0 \in 0.05 \ldots 0.2$ randomly per weight, $\Delta_{max} = 50$, $\Delta_{min} = 0$, initial weights $-0.5 \ldots 0.5$ randomly. RPROP is a fast backpropagation variant that is about as fast as quickprop [6] but more stable without adjustment of the parameters. RPROP requires epoch learning, i.e., the weights are updated only once per epoch. Therefore, the algorithm is fast without parameter tuning for small training sets but not recommendable for large training sets. Lack of parameter tuning helps to avoid the common methodological error of tuning parameters using the test error.

3.3 Experiment results

Altogether, 1296 training runs were made for the comparison, giving 18144 stopping criteria performance records for the 14 criteria. 270 of these records (or 1.5%) from 125 different runs reached the 3000 epoch limit instead of using the stopping criterion itself.

The results for each stopping criterion averaged over all 1296 runs are shown in Table 1. Figure 3 describes the variance embedded in the means given in the

table. I will now explain and then interpret the entries in both, table and figure. Note that the discussion is biased by the particular collection of criteria chosen for the study.

Definitions: For each run, we define $E_{va}(C)$ as the minimum validation error found until criterion C indicates to stop; it is the error after epoch number $t_m(C)$ (read: "time of minimum"). $E_{te}(C)$ is the corresponding test error and characterizes network performance. Stopping occurs after epoch $t_s(C)$ (read: "time of stop"). A *best* criterion $\hat{C}$ of a particular run is one with minimum t_s of all those (among the examined) with minimum E_{va}, i.e., a criterion that found the best validation error fastest. There may be several best, because multiple criteria may stop at the same epoch. Note that there is no single criterion $\hat{C}$ because $\hat{C}$ changes from run to run. C is called *good* in a particular run if $E_{va}(C) = E_{va}(\hat{C})$, i.e., if it is among those that found the lowest validation set error, no matter how fast or slow.

3.4 Discussion: Answers to the questions

We now discuss the questions raised in Section 3.1.

1. *Training time:* The *slowness* of a criterion C in a run, relative to another criterion x is $S_x(C) := t_s(C)/t_s(x)$, i.e., the relative total training time. As we see, the times relative to a fixed criterion as shown in column $S_{GL_2}(C)$ vary by more than factor 4. Therefore, the decision for a particular stopping criterion influences training times dramatically, even if one considers only the range of criteria used here. In contrast, even the slowest criteria train only about 2.5 times

	training time		efficiency and effectiveness			
C	$S_{\hat{c}}(C)$	$S_{GL_2}(C)$	$r(C)$	$B_{\hat{c}}(C)$	$B_{GL_2}(C)$	$P_g(C)$
UP_2	0.792	0.766	0.277	1.055	1.024	0.587
GL_1	0.956	0.823	0.308	1.044	1.010	*0.680
UP_3	1.010	1.264	0.419	*1.026	1.003	0.631
GL_2	1.237	1.000	0.514	1.034	1.000	*0.723
UP_4	1.243	1.566	0.599	*1.020	0.997	0.666
$PQ_{0.5}$	1.253	1.334	0.663	1.027	1.002	0.658
$PQ_{0.75}$	1.466	1.614	0.863	1.021	0.998	0.682
GL_3	1.550	1.450	*0.712	1.025	0.994	*0.748
PQ_1	1.635	1.796	1.038	1.018	0.994	0.704
UP_6	1.786	2.381	1.125	*1.012	0.990	0.737
GL_5	2.014	2.013	1.162	1.021	0.991	*0.772
PQ_2	2.184	2.510	1.636	1.012	0.990	0.768
UP_8	2.485	3.259	1.823	*1.010	0.988	0.759
PQ_3	2.614	3.095	2.140	1.009	0.988	0.800

Table 1. Behavior of stopping criteria. S_{GL_2} is normalized training time, B_{GL_2} is normalized test error (both relative to GL_2). r is the training time redundancy, P_g is the probability of finding a good solution. For further description please refer to the text.

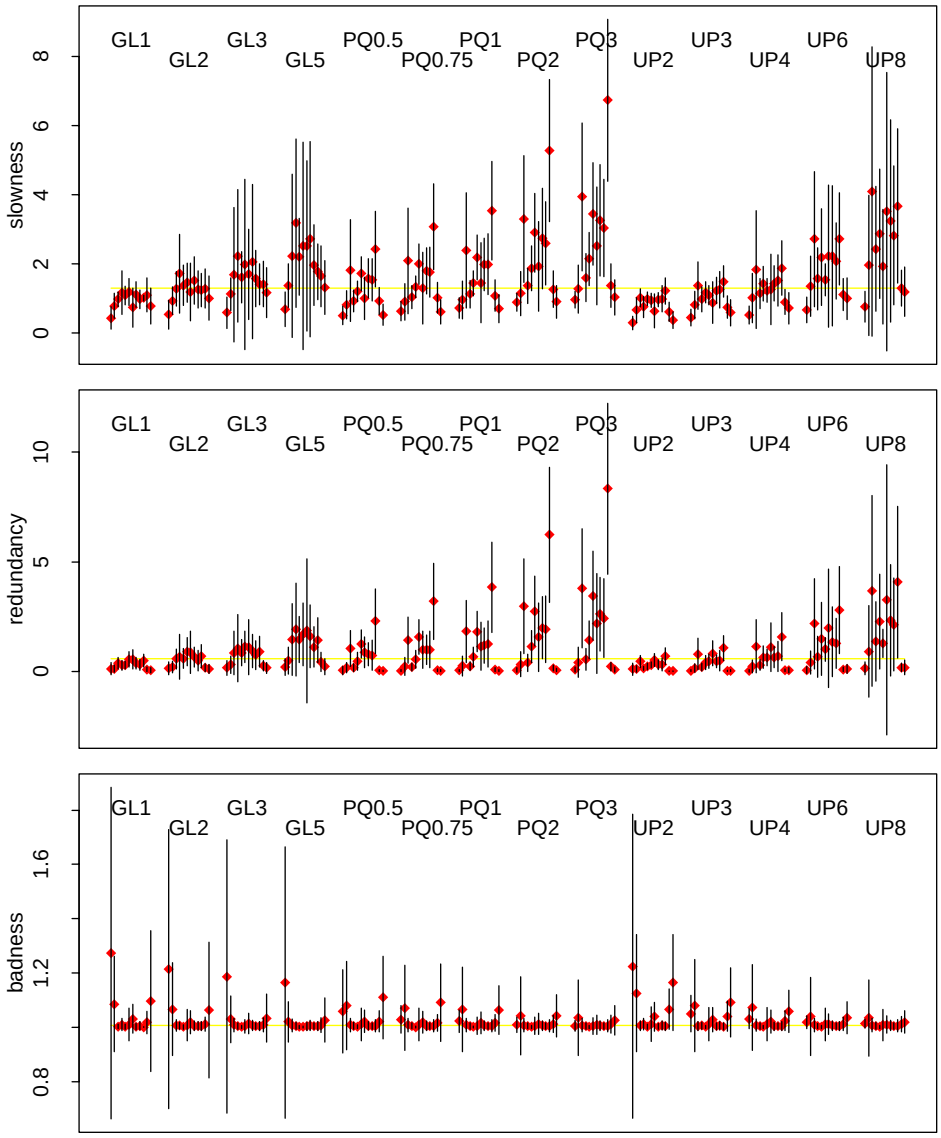

Fig. 3. Variance of slowness $S_{\hat{G}}(C)$ (top), redundancy $r(C)$ (middle), and badness $B_{\hat{G}}(C)$ (bottom) for each pair of learning problem and stopping criterion. In each of the 168 columns, the dot represents the mean computed from 108 runs: learning problem and stopping criterion are fixed, while three other parameters are varied (12 topologies $\times$ 3 runs $\times$ 3 dataset variants). The length of the line is twice the standard deviation within these 108 values. Within each block of dot-line plots, the plots represent (in order) the problems building, cancer, card, diabetes, flare, gene, glass, heart, hearta, horse, soybean, thyroid. The horizontal line marks the median of the means. Note: When comparing the criteria groups, remember that overall the PQ criteria chosen are slower than the others. It is unfair to compare, for example, $PQ_{0.5}$ to GL_1 and UP_2.

as long as the fastest criterion of each run that finds the same result, as indicated in column $S_{\hat{C}}(C)$. This shows that the training times are not completely unreasonable even for the slower criteria, but do indeed pay off to some degree.

2. *Efficiency:* The *redundancy* of a criterion can be defined as $r(C) := (t_s(C)/t_m(C)) - 1$. It characterizes how long the training continues after the final solution has been seen. $r(C) = 0$ would be perfect, $r(C) = 1$ means that the criterion trains twice as long as necessary. Low values indicate efficient criteria. As we see, the slower a criterion is, the less efficient it tends to get. Even the fastest criteria "waste" about one fifth of their overall training time. The slower criteria train twice as long as necessary to find the same solution.

3. *Effectiveness:* We define the *badness* of a criterion C in a run relative to another criterion x as $B_x(C) := E_{te}(C)/E_{te}(x)$, i.e., its relative error on the test set. $P_g(C)$ is the fraction of the 1296 runs in which C was a good criterion. This is an estimate of the probability that C is good in a run. As we see from the P_g column, even the fastest criteria are fairly effective. They reach a result as good as the best (of the same run) in about 60% of the cases. On the other hand, even the slowest criteria are not at all infallible; they achieve about 80%. However, P_g says nothing about how *far* from the optimum the non-good runs are. Columns $B_{\hat{C}}(C)$ and $B_{GL_2}(C)$ indicate that these differences are usually rather small: column $B_{GL_2}(C)$ shows that even the criteria with the lowest error achieve only about 1% lower error on the average than the relatively fast criterion GL_2. In column $B_{\hat{C}}(C)$ we see that several only modestly slow criteria have just about 2% higher error on the average than the best criteria of the same run. For obtaining the lowest possible generalization error, independent of training time, it appears that one has to use an extreme criterion such as GL_{50} or even use a conjunction of all three criteria classes with high parameter values.

4. *Robustness:* We call a criterion *robust* to the degree that its performance is independent of the learning problem and the learning environment (network topology, initial conditions etc.). Optimal robustness would mean that in Figure 3 all dots within a block are at the same height (problem independence) and all lines have length zero (environment independence). Note that slowness and badness are measured relative to the best criterion of the same program run. We observe the following:

- With respect to slowness and redundancy, slower criteria are much less robust than faster ones. In particular the PQ criteria are quite sensitive to the learning problem, with the card and horse problems being worst in this experimental setting.
- With respect to badness, the picture is completely different: slower criteria tend to be slightly *more* robust than slower ones. PQ criteria are a little more robust than the others while GL criteria are significantly less robust. All criteria are more or less instable for the building, cancer, and thyroid problems. In particular, all GL criteria have huge problems with the building problem, whose dataset 1 is the only one that is partitioned non-randomly; it uses chronological order of examples, see [19]. The slower variants of the other criteria types are nicely robust in this case.

– Similar statements apply when one analyzes the influence of only large or only small network topologies separately (not shown in any figure or table). One notable exception was the fact that for networks with very few hidden nodes the PQ criteria are more cost-effective than both the GL and the UP criteria for minimizing $B_{\hat{C}}(C)$. The explanation may be that such small networks do not overfit severely; in this case it is advantageous to take training progress into account as an additional factor to determine when to stop training.

Overall, fast criteria improve the predictability of the training time, while slow ones improve the predictability of the solution quality.

5. *Best tradeoffs:* Despite the common overall trend, some criteria may be more cost-effective than others, i.e., provide better tradeoffs between training time and resulting network performance. Column $B_{\hat{c}}$ of the table suggests that the best tradeoffs between test error and training time are (in order of increasing willingness to spend lots of training time) UP_3, UP_4, and UP_6, if one wants to minimize the expected network performance from a single run. These criteria are also robust. If on the other hand one wants to make several runs and pick the network that seems to be best (based on its validation error), P_g is the relevant metric and the GL criteria are preferable. The best tradeoffs are marked with a star in the table. Figure 4 illustrates these results. The upper curve corresponds

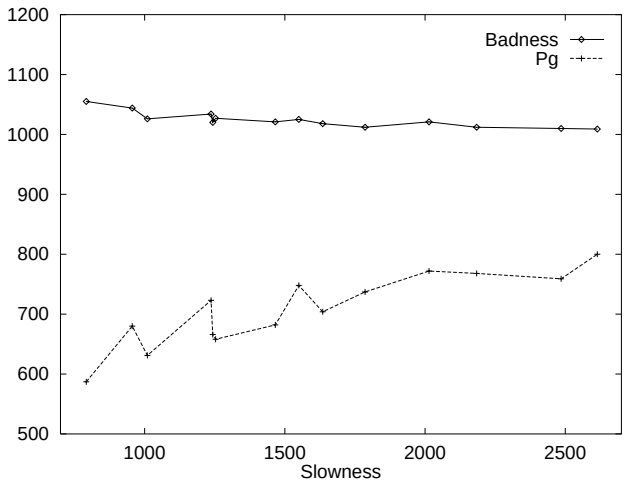

Fig. 4. Badness $B_{\hat{C}}(C)$ and P_g against slowness $S_{\hat{C}}(C)$ of criteria

to column $B_{\hat{C}}$ of the table (plotted against column $S_{\hat{C}}$); local minima indicate criteria with the best tradeoffs. The lower curve corresponds to column P_g; local maxima indicate the criteria with the best tradeoffs. All measurements are scaled by 1000.

6. *Quantification:* From columns $S_{GL_2}(C)$ and $B_{GL_2}(C)$ we can quantify the tradeoff involved in the selection of a stopping criterion as follows: In the range of criteria examined we can roughly trade a 4% decrease in test error (from 1.024 to 0.988) for an about fourfold increase in training time (from 0.766 to 3.095). Within this range, some criteria are somewhat better than others, but there is no panacea.

3.5 Generalization of these results

It is difficult to say whether or how these results apply to different contexts than those of the above evaluation. Speculating though, I would expect that the behavior of the stopping criteria

- is similar for other learning rules, unless they frequently make rather extreme steps in parameter space,
- is similar for other error functions, unless they are discontinuous,
- is similar for other learning tasks, as long as they are in the same ballpark with respect to their nonlinearity, number of inputs and outputs, and amount of available training data.

Note however, that at least with respect to the learning task deviations do occur (see Figure 3). More research is needed to describe which properties of the learning tasks lead to which differences in stopping criteria behavior.

4 Why this works

Detailed theoretical analyses of the error curves cannot yet be done for the most interesting cases such as sigmoidal multi-layer perceptrons trained on a modest number of examples; today they are possible for restricted scenarios only [1, 2, 3, 24] and do usually not aim at finding the optimal stopping criterion in a way comparable to the present work. However, a simplification of the analysis performed by Wang et al. [24] or the alternative view induced by the bias/variance decomposition of the error as described by Geman et al. [10] can give some insights why early stopping behaves as it does.

At the beginning of training (phase I), the error is dominated by what Wang et al. call the *approximation error* — the network has hardly learned anything and is still very biased. During training this part of the error is further and further reduced. At the same time, however, another component of the error increases: the *complexity error* that is induced by the increasing variance of the network model as the possible magnitude and diversity of the weights grows. If we train long enough, the error will be dominated by the complexity error (phase III). Therefore, there is a phase during training, when the approximation and complexity (or: bias and variance) components of the error compete but none of them dominates (phase II). See Amari et al. [1, 2] for yet another view of the training process, using a geometrical interpretation. The task of early

stopping as described in the present work is to detect when phase II ends and the dominance of the variance part begins.

Published theoretical results on early stopping appear to provide some nice techniques for practical application: Wang et al. [24] offer a method for computing the stopping point based on complexity considerations — without using a separate validation set at all. This could save precious training examples. Amari et al. [1, 2] compute the optimal split proportion of training data into training and validation set.

On the other hand, unfortunately, the practical applicability of these theoretical analyses is severely restricted. Wang et al.'s analysis applies to networks where only output weights are being trained; no hidden layer training is captured. It is unclear to what degree the results apply to the multi-layer networks considered here. Amari et al.'s analysis applies to the asymptotic case of very many training examples. The analysis does not give advice on stopping criteria; it shows that early stopping is not useful when very many examples are available but does not cover the much more frequent case when training examples are scarce.

There are several other theoretical works on early stopping, but none of them answers our practical questions. Thus, given these theoretic results, one is still left with making a good stopping decision for practical cases of multilayer networks with only few training examples and faced with a complicated evolution of the validation set error as shown in Figure 2. This is why the present empirical investigation was necessary.

The jagged form of the validation error curve during phase II arises because neither bias nor variance change monotonically, let alone smoothly. The bias error component may change abruptly because training algorithms never perform gradient descent, but take finite steps in parameter space that sometimes have severe results. The observed variance error component may change abruptly because, first, the validation set error is only an estimate of the actual generalization error and, second, the effect of a parameter change may be very different in different parts of parameter space.

Quantitatively, the different error minima that occur during phase II are quite close together in terms of size, but may be rather far apart in terms of training epoch. The exact validation error behavior seems rather unpredictable when only a short left section of the error curve is given. The behavior is also very different for different training situations.

For these reasons no class of stopping criteria has any big advantage over another (on average, for the mix of situations considered here), but scaling the same criterion to be slower always tends to gain a little generalization.

References

1. S. Amari, N. Murata, K.-R. Müller, M. Finke, and H. Yang. Statistical theory of overtraining - is cross-validation effective? In *[23]*, pages 176–182, 1996.
2. S. Amari, N. Murata, K.-R. Müller, M. Finke, and H. Yang. Aymptotic statistical theory of overtraining and cross-validation. *IEEE Trans. on Neural Networks*, 8(5):985–996, September 1997.
3. Pierre Baldi and Yves Chauvin. Temporal evolution of generalization during learning in linear networks. *Neural Computation*, 3:589–603, 1991.
4. Jack D. Cowan, Gerald Tesauro, and J. Alspector, editors. *Advances in Neural Information Processing Systems 6*, San Mateo, CA, 1994. Morgan Kaufman Publishers Inc.
5. Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *[22]*, pages 598–605, 1990.
6. 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.
7. Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation learning architecture. In *[22]*, pages 524–532, 1990.
8. Emile Fiesler (efiesler@idiap.ch). Comparative bibliography of ontogenic neural networks. (submitted for publication), 1994.
9. William Finnoff, Ferdinand Hergert, and Hans Georg Zimmermann. Improving model selection by nonconvergent methods. *Neural Networks*, 6:771–783, 1993.
10. Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.
11. 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.
12. Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *[11]*, pages 164–171, 1993.
13. Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *[16]*, pages 950–957, 1992.
14. Asriel U. Levin, Todd K. Leen, and John E. Moody. Fast pruning using principal components. In *[4]*, 1994.
15. 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.
16. John E. Moody, Stephen J. Hanson, and Richard P. Lippmann, editors. *Advances in Neural Information Processing Systems 4*, San Mateo, CA, 1992. Morgan Kaufman Publishers Inc.
17. N. Morgan and H. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *[22]*, pages 630–637, 1990.
18. Steven J. Nowlan and Geoffrey E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.
19. 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.
20. Russel Reed. Pruning algorithms — a survey. *IEEE Transactions on Neural Networks*, 4(5):740–746, 1993.

21. 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.
22. David S. Touretzky, editor. *Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990. Morgan Kaufman Publishers Inc.
23. D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors. *Advances in Neural Information Processing Systems 8*, Cambridge, MA, 1996. MIT Press.
24. Changfeng Wang, Santosh S. Venkatesh, and J. Stephen Judd. Optimal stopping and effective machine complexity in learning. In *[4]*, 1994.
25. Andreas S. Weigend, David E. Rumelhart, and Bernardo A. Huberman. Generalization by weight-elimination with application to forecasting. In *[15]*, pages 875–882, 1991.

This article was processed using the LaTeX macro package with LLNCS style