FLAT MINIMA

Neural Computation 9(1):1–42 (1997)

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

March 1996

Abstract

We present a new algorithm for finding low complexity neural networks with high generalization capability. The algorithm searches for a "flat" minimum of the error function. A flat minimum is a large connected region in weight-space where the error remains approximately constant. An MDL-based, Bayesian argument suggests that flat minima correspond to "simple" networks and low expected overfitting. The argument is based on a Gibbs algorithm variant and a novel way of splitting generalization error into underfitting and overfitting error. Unlike many previous approaches, ours does not require Gaussian assumptions and does not depend on a "good" weight prior – instead we have a prior over input/output functions, thus taking into account net architecture and training set. Although our algorithm requires the computation of second order derivatives, it has backprop's order of complexity. Automatically, it effectively prunes units, weights, and input lines. Various experiments with feedforward and recurrent nets are described. In an application to stock market prediction, flat minimum search outperforms (1) conventional backprop, (2) weight decay, (3) "optimal brain surgeon" / "optimal brain damage". We also provide pseudo code of the algorithm (omitted from the NC-version).

1 BASIC IDEAS / OUTLINE

Our algorithm tries to find a large region in weight space with the property that each weight vector from that region leads to *similar* small error. Such a region is called a "flat minimum" (Hochreiter & Schmidhuber, 1995). To get an intuitive feeling for why a flat minimum is interesting, consider this: a "sharp" minimum (see figure 2) corresponds to weights which have to be specified with high precision. A flat minimum (see figure 1) corresponds to weights many of which can be given with low precision. In the terminology of the theory of minimum description (message) length (MML, Wallace, 1968; MDL, Rissanen, 1978), fewer bits of information are required to describe a flat minimum (corresponding to a "simple" or low complexity-network). The MDL principle suggests that low network complexity corresponds to high generalization performance. Similarly, the standard Bayesian view favors "fat" maxima of the posterior weight distribution (maxima with a lot of probability mass — see, e.g., Buntine & Weigend, 1991). We will see: **flat minima are fat maxima**.

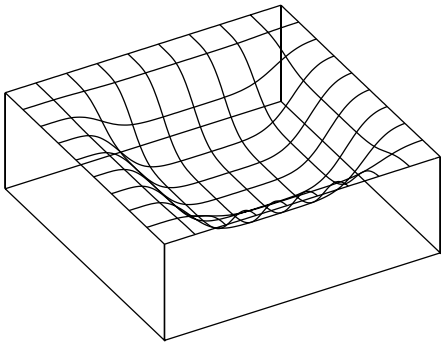

Figure 1: *Example of a "flat" minimum.*

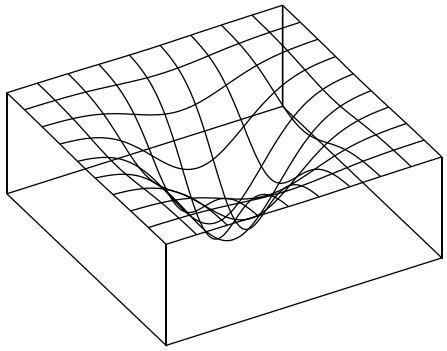

Figure 2: *Example of a "sharp" minimum.*

Unlike, e.g., Hinton and van Camp's method (1993), our algorithm does not depend on the choice of a "good" *weight* prior. It finds a flat minimum by searching for weights that minimize both training error and weight precision. This requires the computation of the Hessian. However, by using an efficient second order method (Pearlmutter, 1994; Møller, 1993), we obtain conventional backprop's order of computational complexity. Automatically, the method effectively reduces numbers of units, weights, and input lines, as well as output sensitivity with respect to remaining weights and units. Unlike, e.g., simple weight decay, our method automatically treats/prunes units and weights in different layers in different reasonable ways.

Outline.

- Section 2 formally introduces basic concepts, such as error measures, flat minima etc.
- Section 3 describes the novel algorithm called "flat minimum search" (FMS).
- Section 4 formally derives the algorithm.
- Section 5 reports experimental generalization results with feedforward and recurrent networks. For instance, in an application to stock market prediction, flat minimum search outperforms the following, widely used competitors: (1) conventional backprop, (2) weight decay, (3) "optimal brain surgeon" / "optimal brain damage".
- Section 6 mentions relations to previous work.
- Section 7 mentions limitations of the algorithm and outlines future work.

- The appendix presents a detailed theoretical justification of our approach.

  Using a variant of the Gibbs algorithm, appendix A.1 defines generalization, underfitting and overfitting error in a novel way. By defining an appropriate prior over input-output functions, we postulate that the most probable network is a "flat" one.

  Appendix A.2 formally justifies the error function minimized by our algorithm.

  Appendix A.3 describes an efficient implementation of the algorithm.

  Appendix A.4 finally presents pseudo code of the algorithm.

2 TASK / ARCHITECTURE / BOXES

Generalization task. The task is to approximate an unknown function $f \subset X \times Y$ mapping a finite set of possible inputs $X \subset R^N$ to a finite set of possible outputs $Y \subset R^K$. A data set D is obtained from f (see appendix A.1). All training information is given by a finite set $D_0 \subset D$. D_0 is called the *training set*. The pth element of D_0 is denoted by an input/target pair (x_p, y_p).

Architecture/ Net functions. For simplicity, we will focus on a standard feedforward net (but in the experiments, we will use recurrent nets as well). The net has N input units, K output units, L weights, and differentiable activation functions. It maps input vectors $x \in R^N$ to output vectors $o(w, x) \in R^K$, where w is the L-dimensional weight vector, and the weight on the connection from unit j to i is denoted w_{ij}. The *net function* induced by w is denoted $net(w)$: for $x \in R^N$, $net(w)(x) = o(w, x) = \big(o^1(w, x), o^2(w, x), \ldots, o^{K-1}(w, x), o^K(w, x)\big)$, where $o^i(w, x)$ denotes the i-th component of $o(w, x)$, corresponding to output unit i.

Training error. We use squared error $E(net(w), D_0) := \sum_{(x_p, y_p) \in D_0} \parallel y_p - o(w, x_p) \parallel^2$, where $\parallel . \parallel$ denotes the Euclidean norm.

Tolerable error. To define a region in weight space with the property that each weight vector from that region leads to small error and similar output, we introduce the tolerable error E_{tol}, a positive constant (see appendix A.1 for a formal definition of E_{tol}). "Small" error is defined as being smaller than E_{tol}. $E(net(w), D_0) > E_{tol}$ implies "underfitting".

Boxes. Each weight w satisfying $E(net(w), D_0) \leq E_{tol}$ defines an "acceptable minimum" (compare $M(D_0)$ in appendix A.1). **We are interested in a large region of connected acceptable minima, where each weight w within this region leads to almost identical net functions $net(w)$. Such a region is called a flat minimum.** We will see that flat minima correspond to low expected generalization error. To simplify the algorithm for finding a large connected region (see below), we do not consider maximal connected regions but focus on so-called *"boxes"* within regions: for each acceptable minimum w, its *box* M_w in weight space is a L-dimensional hypercuboid with center w. For simplicity, each edge of the box is taken to be parallel to one weight axis. Half the length of the box edge in direction of the axis corresponding to weight w_{ij} is denoted by $\Delta w_{ij}(X)$. The $\Delta w_{ij}(X)$ are the maximal (positive) values such that for all L-dimensional vectors κ whose components κ_{ij} are restricted by $|\kappa_{ij}| \leq \Delta w_{ij}(X)$, we have: $E(net(w), net(w + \kappa), X) \leq \epsilon$, where $E(net(w), net(w + \kappa), X) = \sum_{x \in X} \parallel o(w, x) - o(w + \kappa, x) \parallel^2$, and ϵ is a small positive constant defining tolerable output changes (see also equation (1)). Note that $\Delta w_{ij}(X)$ depends on ϵ. Since our algorithm does not use ϵ, however, it is notationally suppressed. $\Delta w_{ij}(X)$ gives the precision of w_{ij}. M_w's *box volume* is defined by $V(\Delta w(X)) := 2^L \prod_{i,j} \Delta w_{ij}(X)$, where $\Delta w(X)$ denotes the vector with components $\Delta w_{ij}(X)$. Our goal is to find large boxes within flat minima.

3 THE ALGORITHM

Let $X_0 = \{x_p \mid (x_p, y_p) \in D_0\}$ denote the inputs of the training set. We approximate $\Delta w(X)$ by $\Delta w(X_0)$, where $\Delta w(X_0)$ is defined like $\Delta w(X)$ in the previous section (replacing X by X_0). For simplicity, in what follows, we will abbreviate $\Delta w(X_0)$ by Δw. Starting with a random initial weight vector, flat minimum search (FMS) tries to find a w that not only has low $E(net(w), D_0)$ but

also defines a box M_w with maximal box volume $V(\Delta w)$ and, consequently, minimal $\tilde{B}(w, X_0) := -\log(\frac{1}{2^L} V(\Delta w)) = \sum_{i,j} -\log \Delta w_{ij}$. Note the relationship to MDL: $\tilde{B}$ is the number of bits required to describe the weights, whereas the number of bits needed to describe the y_p, given w (with $(x_p, y_p) \in D_0$), can be bounded by fixing E_{tol} (see appendix A.1). In the next section we derive the following algorithm. We use gradient descent to minimize $E(w, D_0) = E(net(w), D_0) + \lambda B(w, X_0)$, where $B(w, X_0) = \sum_{x_p \in X_0} B(w, x_p)$, and

$$B(w, x_p) = \frac{1}{2} \left(-L \log \epsilon + \sum_{i,j} \log \sum_k \left(\frac{\partial o^k(w, x_p)}{\partial w_{ij}} \right)^2 + L \log \sum_k \left(\sum_{i,j} \frac{|\frac{\partial o^k(w, x_p)}{\partial w_{ij}}|}{\sqrt{\sum_k (\frac{\partial o^k(w, x_p)}{\partial w_{ij}})^2}} \right)^2 \right) . \tag{1}$$

Here $o^k(w, x_p)$ is the activation of the kth output unit (given weight vector w and input x_p), ϵ is a constant, and λ is the regularization constant (or hyperparameter) which controls the trade-off between regularization and training error (see appendix A.1). To minimize $B(w, X_0)$, for each $x_p \in X_0$ we have to compute

$$\frac{\partial B(w, x_p)}{\partial w_{uv}} = \sum_{k,i,j} \frac{\partial B(w, x_p)}{\partial (\frac{\partial o^k(w, x_p)}{\partial w_{ij}})} \frac{\partial^2 o^k(w, x_p)}{\partial w_{ij} \partial w_{uv}} \text{ for all } u, v . \tag{2}$$

It can be shown that by using Pearlmutter's and Møller's efficient second order method, the gradient of $B(w, x_p)$ can be computed in $O(L)$ time (see details in A.3). **Therefore, our algorithm has the same order of computational complexity as standard backprop.**

4 DERIVATION OF THE ALGORITHM

Outline. We are interested in weights representing nets with tolerable error but flat outputs (see section 2 and appendix A.1). To find nets with flat outputs, two conditions will be defined to specify $B(w, x_p)$ for $x_p \in X_0$ and, as a consequence, $B(w, X_0)$ (see section 3). The first condition ensures flatness. The second condition enforces "equal flatness" in all weight space directions, to obtain low variance of the net functions induced by weights within a box. The second condition will be justified using an MDL-based argument. In both cases, linear approximations will be made (to be justified in A.2).

Formal details. We are interested in weights causing tolerable error (see "acceptable minima" in section 2) that can be perturbed without causing significant output changes, thus indicating the presence of many neighboring weights leading to the same net function. By searching for the boxes from section 2, we are actually searching for low-error weights whose perturbation does not significantly change the net function.

In what follows we treat the input x_p as fixed: for convenience, we suppress x_p, i.e. we abbreviate $o^k(w, x_p)$ by $o^k(w)$. Perturbing the weights w by δw (with components δw_{ij}), we obtain $ED(w, \delta w) := \sum_k (o^k(w + \delta w) - o^k(w))^2$, where $o^k(w)$ expresses o^k's dependence on w (in what follows, however, w often will be suppressed for convenience, i.e. we abbreviate $o^k(w)$ by o^k). Linear approximation (justified in A.2) gives us **"Flatness Condition 1":**

$$ED(w, \delta w) \approx ED_l(\delta w) := \sum_k (\sum_{i,j} \frac{\partial o^k}{\partial w_{ij}} \delta w_{ij})^2 \leq ED_{l,max}(\delta w) := \sum_k (\sum_{i,j} |\frac{\partial o^k}{\partial w_{ij}}| |\delta w_{ij}|)^2 \leq \epsilon, \tag{3}$$

where $\epsilon > 0$ defines tolerable output changes within a box and is small enough to allow for linear approximation (it does not appear in $B(w, x_p)$'s and $B(w, D_0)$'s gradient, see section 3). ED_l is ED's linear approximation, and $ED_{l,max}$ is $max\{ED_l(w, \delta v) | \ \forall_{ij}: \ \delta v_{ij} = \pm \delta w_{ij}\}$. Flatness condition 1 is a "robustness condition" (or "fault tolerance condition", or "perturbation tolerance condition" – see, e.g., Minai & Williams, 1994; Murray & Edwards, 1993; Neti et al., 1992; Matsuoka, 1992; Bishop, 1993; Kerlirzin & Vallet, 1993; Carter et al., 1990).

Many boxes M_w satisfy flatness condition 1. To select a particular, very flat M_w, the following **"Flatness Condition 2"** uses up degrees of freedom left by inequality (3):

$$\forall i,j,u,v : (\delta w_{ij})^2 \sum_k (\frac{\partial o^k}{\partial w_{ij}})^2 = (\delta w_{uv})^2 \sum_k (\frac{\partial o^k}{\partial w_{uv}})^2 \ . \tag{4}$$

Flatness condition 2 enforces equal "directed errors"
$ED_{ij}(w, \delta w_{ij}) = \sum_k (o^k(w_{ij} + \delta w_{ij}) - o^k(w_{ij}))^2 \approx \sum_k (\frac{\partial o^k}{\partial w_{ij}} \delta w_{ij})^2$, where $o^k(w_{ij})$ has the obvious meaning, and δw_{ij} is the i,j-th component of δw. Linear approximation is justified by the choice of ϵ in inequality (3). As will be seen in the MDL-justification to be presented below, *flatness condition 2 favors the box which minimizes the mean perturbation error within the box. This corresponds to minimizing the variance of the net functions induced by weights within the box* (recall that $ED(w, \delta w)$ is quadratic).

How to derive the algorithm from flatness conditions 1 and 2. We first solve equation (4) for $|\delta w_{ij}| = |\delta w_{uv}| \sqrt{\frac{\sum_k \left(\frac{\partial o^k}{\partial w_{uv}}\right)^2}{\sum_k \left(\frac{\partial o^k}{\partial w_{ij}}\right)^2}}$ (fixing u, v for all i, j). Then we insert the $|\delta w_{ij}|$ (with fixed u, v) into inequality (3) (replacing the second "$\leq$" in (3) by "$=$", since we search for the box with maximal volume). This gives us an equation for the $|\delta w_{uv}|$ (which depend on w, but this is notationally suppressed):

$$|\delta w_{uv}| = \frac{\sqrt{\epsilon}}{\sqrt{\sum_k (\frac{\partial o^k}{\partial w_{uv}})^2} \sqrt{\sum_k \left(\sum_{i,j} \frac{|\frac{\partial o^k}{\partial w_{ij}}|}{\sqrt{\sum_k (\frac{\partial o^k}{\partial w_{ij}})^2}} \right)^2}} \ . \tag{5}$$

The $|\delta w_{ij}|$ (u, v is replaced by i, j) approximate the Δw_{ij} from section 2. The box M_w is approximated by AM_w, the box with center w and edge lengths $2\delta w_{ij}$. M_w's volume $V(\Delta w)$ is approximated by AM_w's box volume $V(\delta w) := 2^L \prod_{ij} |\delta w_{ij}|$. Thus, $\tilde{B}(w, x_p)$ (see section 3) can be approximated by $B(w, x_p) := -\log \frac{1}{2^L} V(\delta w) = \sum_{i,j} -\log |\delta w_{ij}|$. This immediately leads to the algorithm given by equation (1).

How can the above approximations be justified? The learning process itself enforces their validity (see A.2). Initially, the conditions above are valid only in a very small environment of an "initial" acceptable minimum. But during search for new acceptable minima with more associated box volume, the corresponding environments are enlarged. Appendix A.2 will prove this for feedforward nets (experiments indicate that this appears to be true for recurrent nets as well).

Comments. Flatness condition 2 influences the algorithm as follows: (1) The algorithm prefers to increase the δw_{ij}'s of weights whose current contributions are not important to compute the target output. (2) The algorithm enforces equal sensitivity of all output units with respect to weights of connections to hidden units. Hence, output units tend to share hidden units, i.e., different hidden units tend to contribute equally to the computation of the target. The contributions of a particular hidden unit to different output unit activations tend to be equal, too.

Flatness condition 2 is essential: flatness condition 1 by itself corresponds to nothing more but first order derivative reduction (ordinary sensitivity reduction). However, as mentioned above, what we really want is to minimize the variance of the net functions induced by weights near the actual weight vector.

Automatically, the algorithm treats units and weights in different layers differently, and takes the nature of the activation functions into account.

MDL-JUSTIFICATION OF FLATNESS CONDITION 2

Let us assume a sender wants to send a description of the function induced by w to a receiver who knows the inputs x_p but not the targets y_p, where $(x_p, y_p) \in D_0$. The MDL principle

suggests that the sender wants to minimize the expected description length of the net function. Let $ED_{mean}(w, X_0)$ denote the mean value of ED on the box. Expected description length is approximated by $\mu ED_{mean}(w, X_0) + B(w, X_0) + c$, where c, μ are positive constants. One way of seeing this is to apply Hinton and van Camp's "bits back" argument to a uniform weight prior (ED_{mean} corresponds to the output variance). However, we prefer to use a different argument: we encode each weight w_{ij} of the box center w by a bitstring according to the following procedure (Δw_{ij} is given):
(0) Define a variable interval $I_{ij} \subset R$.
(1) Make I_{ij} equal to the interval constraining possible weight values.
(2) While $I_{ij} \not\subset [w_{ij} - \Delta w_{ij}, w_{ij} + \Delta w_{ij}]$:
Divide I_{ij} into 2 equally-sized disjunct intervals I_1 and I_2.
If $w_{ij} \in I_1$ then $I_{ij} \leftarrow I_1$; write '1'.
If $w_{ij} \in I_2$ then $I_{ij} \leftarrow I_2$; write '0'.
The final set $\{I_{ij}\}$ corresponds to a "bit-box" within our box. This "bit-box" contains M_w's center w and is described by a bitstring of length $\tilde{B}(w, X_0) + c$, where the constant c is independent of the box M_w. From $ED(w, w_b - w)$ (w_b is the center of the "bit-box") and the bitstring describing the "bit-box", the receiver can compute w as follows: he selects an initialization weight vector within the "bit-box" and uses gradient descent to decrease $B(w_a, X_0)$ until $ED(w_a, w_b - w_a) = ED(w, w_b - w)$, where w_a in the bit-box denotes the receiver's current approximation of w (w_a is constantly updated by the receiver). This is like "FMS without targets" – recall that the receiver knows the inputs x_p. Since w corresponds to the weight vector with the highest degree of local flatness within the "bit-box", the receiver will find the correct w.

$ED(w, w_b - w)$ is described by a Gaussian distribution with mean zero. Hence, the description length of $ED(w, w_b - w)$ is $\mu ED(w, w_b - w)$ (Shannon, 1948). w_b, the center of the "bit-box", cannot be known before training. However, we do know the *expected* description length of the net function, which is $\mu ED_{mean} + \tilde{B}(w, X_0) + c$ (c is a constant independent of w). Let us approximate ED_{mean}: $ED_{l,mean}(w, \delta w) := \frac{1}{V(\delta w)} \int_{AM_w} ED_l(w, \delta v) d\delta v =$

$$\frac{1}{V(\delta w)} 2^L \frac{1}{3} \sum_{i,j} \left((\delta w_{ij})^3 \sum_k \left(\frac{\partial o^k}{\partial w_{ij}} \right)^2 \prod_{u,v \text{with } u,v \neq i,j} \delta w_{uv} \right) = \frac{1}{3} \sum_{i,j} (\delta w_{ij})^2 \sum_k \left(\frac{\partial o^k}{\partial w_{ij}} \right)^2 .$$

Among those w that lead to equal $B(w, X_0)$ (the negative logarithm of the box volume plus $L \log 2$), we want to find those with minimal description length of the function induced by w. Using Lagrange multipliers (viewing the δw_{ij} as variables), **it can be shown that $ED_{l,mean}$ is minimal under the condition $B(w, X_0) = constant$ iff flatness condition 2 holds**. To conclude: with given box volume, we need flatness condition 2 to minimize the expected description length of the function induced by w.

5 EXPERIMENTAL RESULTS

5.1 EXPERIMENT 1 – noisy classification.

Task. The first task is taken from Pearlmutter and Rosenfeld (1991). The task is to decide whether the x-coordinate of a point in 2-dimensional space exceeds zero (class 1) or doesn't (class 2). Noisy training/test examples are generated as follows: data points are obtained from a Gaussian with zero mean and stdev 1.0, bounded in the interval $[-3.0, 3.0]$. The data points are misclassified with probability 0.05. Final input data is obtained by adding a zero mean Gaussian with stdev 0.15 to the data points. In a test with 2,000,000 data points, it was found that the procedure above leads to 9.27 per cent misclassified data. *No* method will misclassify less than 9.27 per cent, due to the inherent noise in the data (including the test data). The training set is based on 200 fixed data points (see figure 3). The test set is based on 120,000 data points.

Results. 10 conventional backprop (BP) nets were tested against 10 equally initialized networks trained by flat minimum search (FMS). *After 1,000 epochs, the weights of our nets essentially stopped changing (automatic "early stopping"), while backprop kept changing weights to learn the outliers in the data set and overfit.* In the end, our approach left a single hidden unit h with a

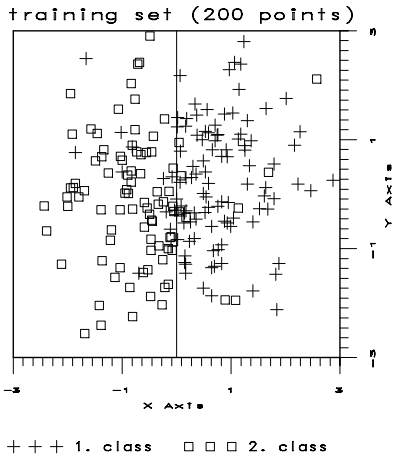

Figure 3: *The 200 input examples of the training set. Crosses represent data points from class 1. Squares represent data points from class 0.*

	Backprop		FMS			Backprop		FMS	
	MSE	dto	MSE	dto		MSE	dto	MSE	dto
1	0.220	1.35	0.193	0.00	6	0.219	1.24	0.187	0.04
2	0.223	1.16	0.189	0.09	7	0.215	1.14	0.187	0.07
3	0.222	1.37	0.186	0.13	8	0.214	1.10	0.185	0.01
4	0.213	1.18	0.181	0.01	9	0.218	1.21	0.190	0.09
5	0.222	1.24	0.195	0.25	10	0.214	1.21	0.188	0.07

Table 1: *10 comparisons of conventional backprop (BP) and flat minimum search (FMS). The second row (labeled "MSE") shows mean squared error on the test set. The third row ("dto") shows the difference between the percentage of misclassifications and the optimal percentage (9.27). The remaining rows provide the analogous information for FMS, which clearly outperforms backprop.*

maximal weight of 30.0 or -30.0 from the x-axis input. Unlike with backprop, the other hidden units were effectively pruned away (outputs near zero). So was the y-axis input (zero weight to h). It can be shown that this corresponds to an "optimal" net with minimal numbers of units and weights. Table 1 illustrates the superior performance of our approach.

Parameters:
Learning rate: 0.1.
Architecture: (2-20-1).
Number of training epochs: 400,000.
With FMS: $E_{tol} = 0.0001$.
See section 5.6 for parameters common to all experiments.

5.2 EXPERIMENT 2 – recurrent nets.

Time-varying inputs. The method works for continually running fully recurrent nets as well. At every time step, a recurrent net with sigmoid activations in $[0, 1]$ sees an input vector from a stream of randomly chosen input vectors from the set $\{(0,0),(0,1),(1,0),(1,1)\}$. The task is to switch on the first output unit whenever an input $(1, 0)$ had occurred two time steps ago, and to

switch on the second output unit without delay in response to any input $(0, 1)$. The task can be solved by a single hidden unit.

Non-weight-decay-like results. With conventional recurrent net algorithms, after training, both hidden units were used to store the input vector. Not so with our new approach. We trained 20 networks. All of them learned perfect solutions. Like with weight decay, most weights to the output decayed to zero. But *unlike* with weight decay, **strong inhibitory** connections (-30.0) switched off one of the hidden units, effectively pruning it away.

Parameters:
Learning rate: 0.1.
Architecture: (2-2-2).
Number of training examples: 1,500.
$E_{tol} = 0.0001$.
See section 5.6 for parameters common to all experiments.

5.3 EXPERIMENT 3 – stock market prediction (1).

Task. We predict the DAX[1] (the German stock market index) using fundamental indicators. Following Rehkugler and Poddig (1990), the net sees the following indicators: (a) German interest rate (*"Umlaufsrendite"*), (b) industrial production divided by money supply, (c) business sentiments (*"IFO Geschäftsklimaindex"*). The input (scaled in the interval [-3.4,3.4]) is the difference between data from the current quarter and last year's corresponding quarter. The goal is to predict the sign of next year's corresponding DAX difference.

Details. The training set consists of 24 data vectors from 1966 to 1972. Positive DAX tendency is mapped to target 0.8, otherwise the target is -0.8. The test set consists of 68 data vectors from 1973 to 1990. Flat minimum search (FMS) is compared against: (1) Conventional backprop (BP8) with 8 hidden units, (2) Backprop with 4 hidden units (BP4) (4 hidden units are chosen because pruning methods favor 4 hidden units, but 3 is not enough), (3) Optimal brain surgeon (OBS; Hassibi & Stork, 1993),) with a few improvements (see section 5.6), (4) Weight decay (WD) according to Weigend et. al (1991) (WD and OBS were chosen because they are well-known and widely used).

Performance measure. Since wrong predictions lead to loss of money, performance is measured as follows. The sum of incorrectly predicted DAX changes is subtracted from the sum of correctly predicted DAX changes. The result is divided by the sum of absolute DAX changes.

Results. See table 2. **Our method outperforms the other methods.**

MSE is irrelevant. Note that MSE is *not* a reasonable performance measure for this task. For instance, although FMS typically makes more correct classifications than WD, FMS' MSE often exceeds WD's. This is because WD's wrong classifications tend to be close to 0, while FMS often prefers large weights yielding strong output activations — FMS' few false classifications tend to contribute a lot to MSE.

Parameters:
Learning rate: 0.01.
Architecture: (3-8-1), except BP4 with (3-4-1).
Number of training examples: 20,000,000.
Method specific parameters:
FMS: $E_{tol} = 0.13$; $\Delta\lambda = 0.001$.
WD: like with FMS, but $w_0 = 0.2$.
OBS: $E_{tol} = 0.015$ (the same result was obtained with higher E_{tol} values, e.g. 0.13).
See section 5.6 for parameters common to all experiments.

[1] Raw DAX version according to *Statistisches Bundesamt* (federal office of statistics). Other data are from the same source (except for business sentiment). Collected by Christian Puritscher, for a diploma thesis in industrial management at LMU, Munich.

Method	train MSE	test MSE	removed w	u	performance max	min	mean
BP8	0.003	0.945			47.33	25.74	37.76
BP4	0.043	1.066			42.02	42.02	42.02
OBS	0.089	1.088	14	3	48.89	27.17	41.73
WD	0.096	1.102	22	4	44.47	36.47	43.49
FMS	0.040	1.162	24	4	47.74	39.70	43.62

Table 2: *Comparisons of conventional backprop (BP4, BP8), optimal brain surgeon (OBS), weight decay (WD), and flat minimum search (FMS). All nets except BP4 start out with 8 hidden units. Each value is a mean of 7 trials. Column "MSE" shows mean squared error. Column "w" shows the number of pruned weights, column "u" shows the number of pruned units, the final 3 rows ("max", "min", "mean") list maximal, minimal and mean performance (see text) over 7 trials. Note that test MSE is insignificant for performance evaluations (this is due to targets 0.8/-0.8, as opposed to the "real" DAX targets).* **Our method outperforms all other methods.**

5.4 EXPERIMENT 4 – stock market prediction (2).

Task. We predict the DAX again, using the basic set-up of the experiment in section 5.3. However, the following modifications are introduced:

- There are two additional inputs: (d) dividend rate, (c) foreign orders in manufacturing industry.
- Monthly predictions are made. The net input is the difference between the current month's data and last month's data. The goal is to predict the sign of next month's corresponding DAX difference.
- There are 228 training examples and 100 test examples.
- The target is the percentage of DAX change scaled in the interval [-1,1] (outliers are ignored).
- Performance of WD and FMS is also tested on networks "spoiled" by conventional backprop ("WDR" and "FMSR" – the "R" stands for *R*etraining).

Results are shown in table 3. **Average performance of our method exceeds the ones of weight decay, OBS, and conventional backprop.**

Table 3 also shows superior performance of our approach when it comes to retraining "spoiled" networks (note that OBS is a retraining method by nature). FMS led to the best improvements in generalization performance.

Parameters:
Learning rate: 0.01.
Architecture: (5-8-1).
Number of training examples: 20,000,000.
Method specific parameters:
FMS: $E_{tol} = 0.235$; $\Delta\lambda = 0.0001$; if $E_{\text{average}} < E_{tol}$ then $\Delta\lambda$ is set to 0.001.
WD: like with FMS, but $w_0 = 0.2$.
FMSR: like with FMS, but $E_{tol} = 0.15$; number of retraining examples: 5,000,000.
WDR: like with FMSR, but $w_0 = 0.2$.
OBS: $E_{tol} = 0.235$. See section 5.6 for parameters common to all experiments.

Method	train MSE	test MSE	removed w	removed u	performance max	performance min	performance mean
BP	0.181	0.535			57.33	20.69	41.61
OBS	0.219	0.502	15	1	50.78	32.20	40.43
WDR	0.180	0.538	0	0	62.54	13.64	41.17
FMSR	0.180	0.542	0	0	64.07	24.58	41.57
WD	0.235	0.452	17	3	54.04	32.03	40.75
FMS	0.240	0.472	19	3	54.11	31.12	44.40

Table 3: *Comparisons of conventional backprop (BP), optimal brain surgeon (OBS), weight decay after spoiling the net with BP (WDR), flat minimum search after spoiling the net with BP (FMSR), weight decay (WD), flat minimum search (FMS). All nets start out with 8 hidden units. Each value is a mean of 10 trials. Column "MSE" shows mean squared error. Column "w" shows the number of pruned weights, column "u" shows the number of pruned units, the final 3 rows ("max", "min", "mean") list maximal, minimal and mean performance (see text) over 10 trials (note again that MSE is an irrelevant performance measure for this task).* **Flat minimum search outperforms all other methods.**

5.5 EXPERIMENT 5 – stock market prediction (3).

Task. This time, we predict the DAX using weekly *technical* (as opposed to fundamental) indicators. The data (DAX values and 35 technical indicators) was provided by *Bayerische Vereinsbank*.

Data analysis. To analyze the data, we computed: (1) The pairwise correlation coefficients of the 35 technical indicators. (2) The maximal pairwise correlation coefficients of all indicators and all linear combinations of two indicators. This analysis revealed that only 4 indicators are not highly correlated. For such reasons, our nets see only the 8 most recent DAX-changes and the following technical indicators: (a) the DAX value, (b) change of 24-week relative strength index ("RSI") – the relation of increasing tendency to decreasing tendency, (c) "5 week statistic", (d) "MACD" (smoothened difference of exponentially weighted 6 week and 24 week DAX).

Input data. The final network input is obtained by scaling the values (a-d) and the 8 most recent DAX-changes in $[-2, 2]$. The training set consists of 320 data points (July 1985 to August 1991). The targets are the actual DAX changes scaled in $[-1, 1]$.

Comparison. The following methods are applied to the training set: (1) Conventional backprop (BP), (2) optimal brain surgeon / optimal brain damage (OBS/OBD), (3) weight decay (WD) according to Weigend et al., (4) flat minimum search (FMS). The resulting nets are evaluated on a test set consisting of 100 data points (August 1991 to July 1993).

Performance is measured like in section 5.3.

Results. Table 4 shows the results. **Again, our method outperforms the other methods**.

Parameters:

Learning rate: 0.01.

Architecture: (12-9-1).

Training time: 10,000,000 examples.

Method specific parameters:

OBS/OBD: $E_{tol} = 0.34$.

FMS: $E_{tol} = 0.34$; $\Delta\lambda = 0.003$. If $E_{\text{average}} < E_{tol}$ then $\Delta\lambda$ is set to 0.03.

WD: like with FMS, but $w_0 = 0.2$.

See section 5.6 for parameters common to all experiments.

Method	train MSE	test MSE	removed w	removed u	performance max	performance min	performance mean
BP	0.13	1.08			28.45	-16.7	8.08
OBS	0.38	0.912	55	1	27.37	-6.08	10.70
WD	0.51	0.334	110	8	26.84	-6.88	12.97
FMS	0.46	0.348	103	7	29.72	18.09	21.26

Table 4: *Comparisons of conventional backprop (BP), optimal brain surgeon (OBS), weight decay (WD), flat minimum search (FMS). All nets start out with 9 hidden units. Each value is a mean of 10 trials. Column "MSE" shows mean squared error. Column "w" shows the number of pruned weights, column "u" shows the number of pruned units, the final 3 rows ("max", "min", "mean") list maximal, minimal and mean performance (see text) over 10 trials (note again that MSE is an irrelevant performance measure for this task).* **Flat minimum search outperforms all other methods.**

5.6 DETAILS / PARAMETERS

With exception of the experiment in section 5.2, all units are sigmoid in the range of $[-1.0, 1.0]$. Weights are constrained to $[-30, 30]$ and initialized in [-0.1,0.1]. The latter ensures high first order derivatives in the beginning of the learning phase. WD is set up to hardly punish weights below $w_0 = 0.2$. E_{average} is the average error on the training set, approximated using exponential decay: $E_{\text{average}} \leftarrow \gamma E_{\text{average}} + (1-\gamma)E(net(w), D_0)$, where $\gamma = 0.85$.

FMS details. To control $B(w, D_0)$'s influence during learning, its gradient is normalized and multiplied by the length of $E(net(w), D_0)$'s gradient (same for weight decay, see below). λ is computed like in (Weigend et al., 1991) and initialized with 0. Absolute values of first order derivatives are replaced by 10^{-20} if below this value. We ought to judge a weight w_{ij} as being pruned if δw_{ij} (see equation (5) in section 4) exceeds the length of the weight range. However, the unknown scaling factor ϵ (see inequality (3) and equation (5) in section 4) is required to compute δw_{ij}. Therefore, we judge a weight w_{ij} as being pruned if, with arbitrary ϵ, δw_{ij} is much bigger than the corresponding δ's of the other weights (typically, there are clearly separable classes of weights with high and low δ's, which differ from each other by a factor ranging from 10^2 to 10^5).

If all weights to and from a particular unit are very close to zero, the unit is lost: due to tiny derivatives, the weights will never again increase significantly. Sometimes, it is necessary to bring lost units back into the game. For this purpose, every n_{init} time steps (typically, n_{init} = 500,000), all weights w_{ij} with $0 \leq w_{ij} < 0.01$ are randomly re-initialized in $[0.005, 0.01]$; all weights w_{ij} with $0 \geq w_{ij} > -0.01$ are randomly initialized in $[-0.01, -0.005]$, and λ is set to 0.

Weight decay details. We used Weigend et al.'s weight decay term: $D(w) = \sum_{i,j} \frac{w_{ij}^2/w_0}{1+w_{ij}^2/w_0}$. Like with FMS, $D(w, w_0)$'s gradient was normalized and multiplied by the length of $E(net(w), D_0)$'s gradient. λ was adjusted like with FMS. Lost units were brought back like with FMS.

Modifications of OBS. Typically, most weights exceed 1.0 after training. Therefore, higher order terms of δw in the Taylor expansion of the error function do not vanish. Hence, OBS is not fully theoretically justified. Still, we used OBS to delete high weights, assuming that higher order derivatives are small if second order derivatives are. To obtain reasonable performance, we modified the original OBS procedure (notation following Hassibi and Stork, 1993):

- To detect the weight that deserves deletion, we use both $L_q = \frac{w_q^2}{[H^{-1}]_{qq}}$ (the original value used by Hassibi and Stork) and $T_q := \frac{\partial E}{\partial w_q} w_q + \frac{1}{2}\frac{\partial^2 E}{\partial w_q^2} w_q^2$. Here H denotes the Hessian and H^{-1} its approximate inverse. We delete the weight causing minimal training set error (after tentative deletion).
- Like with OBD (LeCun et al., 1990), to prevent numerical errors due to small eigenvalues

of H, we do: if $L_q < 0.00001$ or $T_q < 0.00001$ or $\| I - H^{-1}H \| > 10.0$ (bad approximation of H^{-1}), we only delete the weight detected in the previous step – the other weights remain the same. Here $\| . \|$ denotes the sum of the absolute values of all components of a matrix.

- If OBS' adjustment of the remaining weights leads to at least one absolute weight change exceeding 5.0, then δw is scaled such that the maximal absolute weight change is 5.0. This leads to better performance (also due to small eigenvalues).

- If $E_{\text{average}} > E_{tol}$ after weight deletion, then the net is retrained until either $E_{\text{average}} < E_{tol}$ or the number of training examples exceeds 800,000. Practical experience indicates that the choice of E_{tol} hardly influences the result.

- OBS is stopped if $E_{\text{average}} > E_{tol}$ after retraining. The most recent weight deletion is countermanded.

6 RELATION TO PREVIOUS WORK

Most previous algorithms for finding low complexity networks with high generalization capability are based on different prior assumptions. They can be broadly classified into two categories (see Schmidhuber (1994a), however, for an exception):

(1) Assumptions about the prior weight distribution. Hinton and van Camp (1993) and Williams (1994) assume that pushing the posterior weight distribution close to the weight prior leads to "good" generalization (see more details below). Weight decay (e.g., Hanson & Pratt, 1989; Krogh & Hertz, 1992) can be derived, e.g., from Gaussian or Laplace weight priors. Nowlan and Hinton (1992) assume that a distribution of networks with many similar weights generated by Gaussian mixtures is "better" *a priori*. MacKay's weight priors (1992b) are implicit in additional penalty terms, which embody the assumptions made. The problem with the approaches above is this: there may not be a "good" weight prior for *all* possible architectures and training sets. With FMS, however, we don't have to select a "good" weight prior — instead we choose a prior over input/output functions. This automatically takes the net architecture and the training set into account.

(2) Prior assumptions about how theoretical results on early stopping and network complexity carry over to practical applications. Such assumptions are implicit in methods based on validation sets (Mosteller & Tukey, 1968; Stone, 1974; Eubank, 1988; Hastie & Tibshirani, 1993), e.g., "generalized cross validation" (Craven & Wahba, 1979; Golub et al., 1979), "final prediction error" (Akaike, 1970), "generalized prediction error" (Moody & Utans, 1994; Moody, 1992). See also Holden (1994), Wang et al. (1994), Amari and Murata (1993), and Vapnik's "structural risk minimization" (Guyon et al., 1992; Vapnik, 1992).

Constructive algorithms / pruning algorithms. Other architecture selection methods are less flexible in the sense that they can be used only either before or after weight adjustments. Examples are "sequential network construction" (Fahlman & Lebiere, 1990; Ash, 1989; Moody, 1989), input pruning (Moody, 1992; Refenes et al., 1994), unit pruning (White, 1989; Mozer & Smolensky, 1989; Levin et al., 1994), weight pruning, e.g. "optimal brain damage" (LeCun et al., 1990), "optimal brain surgeon" (Hassibi & Stork, 1993).

Hinton and van Camp (1993). They minimize the sum of two terms: the first is conventional error plus variance, the other is the distance $\int p(w \mid D_0) \log \frac{p(w|D_0)}{p(w)} dw$ between posterior $p(w \mid D_0)$ and weight prior $p(w)$. They have to choose a "good" *weight* prior. But, as mentioned above, perhaps there is no "good" weight prior for *all* possible architectures and training sets. With FMS, however, we don't depend on a "good" weight prior — instead we have a prior over input/output functions, thus taking into account net architecture and training set. Furthermore, Hinton and van Camp have to compute variances of weights and unit activations, which (in general) *cannot* be done using linear approximation. Intuitively speaking, their weight variances are related to our Δw_{ij}. Our approach, however, *does* justify linear approximation, as seen in appendix A.2.

Wolpert (1994a). His (purely theoretical) analysis suggests an interesting different additional error term (taking into account local flatness in all directions): the logarithm of the Jacobi determinant of the functional from weight space to the space of possible nets. This term is small if the net output (based on the current weight vector) is locally flat in weight space (if many neighboring weights lead to the same net function in the space of possible net functions). It is not clear, however, how to derive a practical algorithm (e.g., a pruning algorithm) from this.

Murray and Edwards (1993). They obtain additional error terms consisting of weight squares and second order derivatives. Unlike our approach, theirs *explicitly* prefers weights near zero. In addition, their approach appears to require much more computation time (due to second order derivatives in the error term).

7 LIMITATIONS / FINAL REMARKS / FUTURE RESEARCH

How to adjust λ? Given recent trends in neural computing (see, e.g., MacKay, 1992a, 1992b), it may seem like a step backwards that λ is adapted using an ad-hoc heuristic from Weigend et al., 1991. However, for determining λ in MacKay's style, one would have to compute the Hessian of the cost function. Since our term $B(w, X_0)$ includes first order derivatives, adjusting λ would require the computation of third order derivatives. This is impracticable. Also, to optimize the regularizing parameter λ (see MacKay, 1992b), we need to compute the function $\int d^L w \exp(-\lambda B(w, X_0))$, but it is not obvious how: the "quick and dirty version" (MacKay, 1992a) cannot deal with the unknown constant ϵ in $B(w, X_0)$.

Future work will investigate how to adjust λ without too much computational effort. In fact, as will be seen in appendix A.1, the choices of λ and E_{tol} are correlated — the optimal choice of E_{tol} may indeed correspond to the optimal choice of λ.

Generalized boxes? The boxes found by the current version of FMS are axis-aligned. This may cause an under-estimate of flat minimum volume. Although our experiments indicate that box search works very well, it will be interesting to compare alternative approximations of flat minimum volumes.

Multiple initializations? First, consider this FMS "alternative": run conventional backprop starting with several random initial guesses, and pick the flattest minimum with largest volume. This does not work: conventional backprop changes the weights according to steepest descent — it runs away from flat ranges in weight space! Using an "FMS *committee*" (multiple runs with different initializations), however, would lead to a better approximation of the posterior. This is left for future work.

Notes on generalization error. If the prior distribution of targets $p(f)$ (see appendix A.1) is uniform (or if the distribution of prior distributions is uniform), no algorithm can obtain a lower expected generalization error than training error reducing algorithms (see, e.g., Wolpert, 1994b). Typical target distributions in the real world are *not* uniform, however – the real world appears to favor problem solutions with low algorithmic complexity. See, e.g., Schmidhuber (1994a). MacKay (1992a) suggests to search for alternative priors if the generalization error indicates a "poor regulariser". He also points out that with a "good" approximation of the non-uniform prior, more probable posterior hypothesis do not necessarily have a lower generalization error. For instance, there may be noise on the test set, or two hypotheses representing the same function may have different posterior values, and the expected generalization error ought to be computed over the whole posterior and not for a single solution. Schmidhuber (1994b) proposes a general, "self-improving" system whose entire life is viewed as a single training sequence and which continually attempts to incrementally modify its priors based on experience with previous problems — see also Schmidhuber (1996). It remains to be seen, however, whether this will lead to practicable algorithms.

Ongoing work on low-complexity coding. FMS can also be useful for *unsupervised learning*. In recent work, we postulate that a "generally useful" code of given input data fulfills three

MDL-inspired criteria: (1) It conveys information about the input data. (2) It can be computed from the data by a low-complexity mapping. (3) The data can be computed from the code by a low-complexity mapping. To obtain such codes, we simply train an auto-associator with FMS (after training, codes are represented across the hidden units). In initial experiments, depending on data and architecture, this always led to well-known kinds of codes considered useful in previous work by numerous researchers: we sometimes obtained factorial codes, sometimes local codes, and sometimes sparse codes. In most cases, the codes were of the low-redundancy, binary kind. Initial experiments with a speech data benchmark problem (vowel recognition) already showed the *true* usefulness of codes obtained by FMS: feeding the codes into standard, supervised, overfitting backprop classifiers, we obtained much better generalization performance than competing approaches.

APPENDIX – THEORETICAL JUSTIFICATION

Contents:

Note: Appendices A.3.2 and A.4 were omitted from the version for *Neural Computation*. An alternative version of the entire appendix (but with some minor errors) can be found in Hochreiter & Schmidhuber (1994).

A.1. FLAT NETS: THE MOST PROBABLE HYPOTHESES

Short Guide Through Appendix A.1. We introduce a novel kind of generalization error that can be split into an overfitting error and an underfitting error. To find hypotheses causing low generalization error, we first select a subset of hypotheses causing low underfitting error. We are interested in those of its elements causing low overfitting error.

More Detailed Guide Through Appendix A.1.

- After listing relevant definitions we will introduce a somewhat unconventional variant of the Gibbs algorithm, designed to take into account that FMS uses only the training data D_0 to determine $G(. \mid D_0)$, a distribution over the set of hypotheses expressing our prior belief in hypotheses (here we do not care where the data came from — this will be treated later).

- This variant of the Gibbs algorithm will help us to introduce the concept of "expected extended generalization error", which can be split into an "overfitting error" (relevant for measuring whether the learning algorithm focuses too much on the training set) and an "underfitting error" (relevant for measuring whether the algorithm sufficiently approximates the training set). To obtain these errors, we measure the Kullback-Leibler distance between posterior $p(. \mid D_0)$ after training on the training set and posterior $p_{D_0}(. \mid D)$ after (hypothetical) training on all data (here the subscript D_0 indicates that for learning D, $G(. \mid D_0)$ is used as prior belief in hypotheses, too). The overfitting error measures the information conveyed by $p(. \mid D_0)$, but not by $p_{D_0}(. \mid D)$. The underfitting error measures the information conveyed by $p_{D_0}(. \mid D)$, but not by $p(. \mid D_0)$.

- We then introduce the "tolerable error level" and the set of "acceptable minima". The latter contains hypotheses with low underfitting error, assuming that D_0 indeed conveys information about the test set (every training set error reducing algorithm makes this assumption). In the remainder of the appendix, we will focus only on hypotheses within the set of acceptable minima.

- We introduce the "relative overfitting error", which is the relative contribution of a hypothesis to the mean overfitting error on the set of acceptable minima. The relative overfitting error measures the overfitting error of hypotheses with low underfitting error. The goal is to find a hypothesis with low overfitting error and, consequently, with low generalization error.

- The relative overfitting error is approximated based on the trade-off between low training set error and large values of $G(. \mid D_0)$. The distribution $G(. \mid D_0)$ is restricted to the set of acceptable minima, to obtain the distribution $G_{M(D_0)}(. \mid D_0)$.

- We then assume the data is obtained from a target chosen according to a given prior distribution. Using previously introduced distributions, we derive the expected test set error and

the expected relative overfitting error. We want to reduce the latter by choosing a certain $G_{M(D_0)}(. \mid D_0)$ and $G(. \mid D_0)$.

- The special case of noise free data is considered.
- To be able to minimize the expected relative overfitting error, we need to adopt a certain prior belief $p(f)$. The only unknown distributions required to determine $G_{M(D_0)}(. \mid D_0)$ are $p(D_0 \mid f)$ and $p(D \mid f)$ — they describe how (noisy) data is obtained from the target. We have to make the following assumptions: the choice of prior belief is "appropriate", the noise on data drawn from the target has mean 0, and small noise is more probable than large noise (the noise assumptions ensure that reducing the training error — by choosing some h from $M(D_0)$ — reduces the expected underfitting error). We don't need Gaussian assumptions, though.
- We show that FMS approximates our special variant of the Gibbs algorithm: the prior is approximated locally in weight space, and flat $net(w)$ are approximated by flat $net(w')$ with w' near w in weight space.

Definitions. Let $A = \{(x, y) \mid x \in X, y \in Y\}$ be the set of all possible input/output pairs (pairs of vectors). Let NET be the set of functions that can be implemented by the network. For every net function $g \in NET$ we have $g \subset A$. Elements of NET are parameterized with a parameter vector w from the set of possible parameters W. $net(w)$ is a function which maps a parameter vector w onto a net function g (*net* is surjective.) Let T be the set of target functions f, where $T \subset NET$. Let H be the set of hypothesis functions h, where $H \subset T$. For simplicity, take all sets to be finite, and let all functions map each $x \in X$ to some $y \in Y$. Values of functions with argument x are denoted by $g(x), net(w)(x), f(x), h(x)$. We have $(x, g(x)) \in g; (x, net(w)(x)) \in net(w); (x, f(x)) \in f; (x, h(x)) \in h$.

Let $D = \{(x_p, y_p) \mid 1 \leq p \leq m\}$ be the data, where $D \subset A$. D is divided into a training set $D_0 = \{(x_p, y_p) \mid 1 \leq p \leq n\}$ and a test set $D \setminus D_0 = \{(x_p, y_p) \mid n < p \leq m\}$. For the moment, we are not interested in how D was obtained.

We use squared error $E(D, h) := \sum_{p=1}^{m} \parallel y_p - h(x_p) \parallel^2$, where $\parallel . \parallel$ is the Euclidean norm. $E(D_0, h) := \sum_{p=1}^{n} \parallel y_p - h(x_p) \parallel^2$. $E(D \setminus D_0, h) := \sum_{p=n+1}^{m} \parallel y_p - h(x_p) \parallel^2$. $E(D, h) = E(D_0, h) + E(D \setminus D_0, h)$ holds.

Learning. We use a variant of the Gibbs formalism (see Opper & Haussler, 1991, or Levin et al., 1990). Consider a stochastic learning algorithm (random weight initialization, random learning rate). The learning algorithm attempts to reduce training set error by randomly selecting a hypothesis with low $E(D_0, h)$, according to some *conditional* distribution $G(. \mid D_0)$ over H. $G(. \mid D_0)$ is chosen in advance, but in contrast to traditional Gibbs (which deals with unconditional distributions on H), we may take a look at the training set before selecting G. For instance, one training set may suggest linear functions as being more probable than others, another one splines, etc. The unconventional Gibbs variant is appropriate because FMS uses only X_0 (the set of first components of D_0's elements, see section 3) to compute the flatness of $net(w')$. The trade-off between the desire for low $E(D_0, h)$ and the a priori belief in a hypothesis according to $G(. \mid D_0)$ is governed by a positive constant β (interpretable as the inverse temperature from statistical mechanics, or the amount of stochasticity in the training algorithm).

We obtain $p(h \mid D_0)$, the learning algorithm applied to data D_0:

$$p(h \mid D_0) = \frac{G(h \mid D_0) \exp(-\beta E(D_0, h))}{Z(D_0, \beta)}, \tag{6}$$

where

$$Z(D_0, \beta) = \sum_{h \in H} G(h \mid D_0) \exp(-\beta E(D_0, h)). \tag{7}$$

$Z(D_0, \beta)$ is the "error momentum generating function", or the "weighted accessible volume in configuration space" or "the partition function" (from statistical mechanics).

For theoretical purposes, assume we know D and may use it for learning. To learn, we use the same distribution $G(h \mid D_0)$ as above (prior belief in some hypotheses h is based exclusively on the training set). There is a reason why we do not use $G(h \mid D)$ instead: $G(h \mid D)$ does not allow for making a distinction between a better prior belief in hypotheses and a better approximation of the test set data. However, we are interested in how $G(h \mid D_0)$ performs on the test set data $D \setminus D_0$. We obtain

$$p_{D_0}(h \mid D) = \frac{G(h \mid D_0) \exp(-\beta E(D,h))}{Z_{D_0}(D,\beta)}, \tag{8}$$

where

$$Z_{D_0}(D,\beta) = \sum_{h \in H} G(h \mid D_0) \exp(-\beta E(D,h)). \tag{9}$$

The subscript D_0 indicates that the prior belief is chosen based on D_0 only.

Expected extended generalization error. We define the *expected extended generalization error* $E_G(D, D_0)$ on the unseen test exemplars $D \setminus D_0$:

$$E_G(D,D_0) := \sum_{h \in H} p(h \mid D_0) E(D \setminus D_0, h) - \sum_{h \in H} p_{D_0}(h \mid D) E(D \setminus D_0, h). \tag{10}$$

Here $E_G(D, D_0)$ is the mean error on $D \setminus D_0$ after learning with D_0, minus the mean error on $D \setminus D_0$ after learning with D. The second (negative) term is a lower bound (due to non-zero temperature) for the error on $D \setminus D_0$ after learning the training set D_0. Note: for the zero temperature limit $\beta \to \infty$ we get (summation convention explained at the end of this paragraph) $E_G(D, D_0) = \sum_{h \in H, D_0 \subset h} \frac{G(h|D_0)}{Z(D_0)} E(D \setminus D_0, h)$, where $Z(D_0) = \sum_{h \in H, D_0 \subset h} G(h \mid D_0)$. In this case, the generalization error depends on $G(h \mid D_0)$, restricted to those hypotheses h compatible with D_0 ($D_0 \subset h$). For $\beta \to 0$ (full stochasticity), we get $E_G(D, D_0) = 0$.
Summation convention: in general, $\sum_{h \in H, D_0 \subset h}$ denotes summation over those h satisfying $h \in H$ and $D_0 \subset h$. In what follows, we will keep an analogous convention: the first symbol is the running index, for which additional expressions specify conditions.

Overfitting and underfitting error. Let us separate the generalization error into an overfitting error E_o and an underfitting error E_u (in analogy to Wang et al., 1994; and Guyon et al., 1992). We will see that *overfitting and underfitting error correspond to the two different error terms in our algorithm: decreasing one term is equivalent to decreasing E_o, decreasing the other is equivalent to decreasing E_u.* Using the Kullback-Leibler distance (Kullback, 1959), we measure the information conveyed by $p(. \mid D_0)$, but not by $p_{D_0}(. \mid D)$ (see figure 4). We may view this as information about $G(. \mid D_0)$: since there are more h which are compatible with D_0 than there are h which are compatible with D, $G(. \mid D_0)$'s influence on $p(h \mid D_0)$ is stronger than its influence on $p_{D_0}(h \mid D)$. To get the non-stochastic bias (see definition of E_G), we divide this information by β and obtain the overfitting error:

$$\begin{aligned} E_o(D,D_0) \quad := \quad & \frac{1}{\beta} \sum_{h \in H} p(h \mid D_0) \ln \frac{p(h \mid D_0)}{p_{D_0}(h \mid D)} = \\ & \sum_{h \in H} p(h \mid D_0) E(D \setminus D_0, h) + \frac{1}{\beta} \ln \frac{Z_{D_0}(D,\beta)}{Z(D_0,\beta)}. \end{aligned} \tag{11}$$

Analogously, we measure the information conveyed by $p_{D_0}(. \mid D)$, but not by $p(. \mid D_0)$ (see figure 5). This information is about $D \setminus D_0$. To get the non-stochastic bias (see definition of E_G), we divide this information by β and obtain the underfitting error:

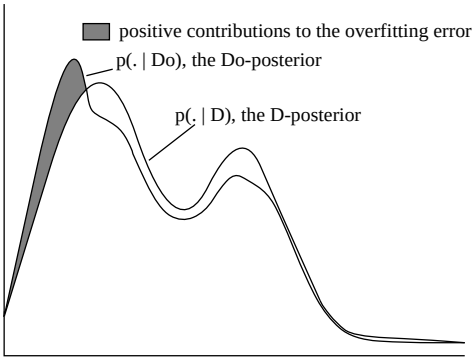

Figure 4: *Positive contributions to the overfitting error $E_o(D, D_0)$, after learning the training set with a large β.*

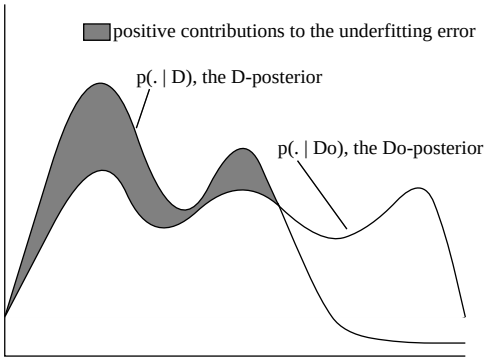

Figure 5: *Positive contributions to the underfitting error $E_u(D_0, D)$, after learning the training set with a small β. Again, we use the D-posterior from figure 4, assuming it is almost fully determined by $E(D, h)$ (even if β is smaller than in figure 4).*

$$E_u(D, D_0) := \frac{1}{\beta} \sum_{h \in H} p_{D_0}(h \mid D) \ln \frac{p_{D_0}(h \mid D)}{p(h \mid D_0)} = \tag{12}$$
$$- \sum_{h \in H} p_{D_0}(h \mid D) E(D \setminus D_0, h) + \frac{1}{\beta} \ln \frac{Z(D_0, \beta)}{Z_{D_0}(D, \beta)}.$$

Peaks in $G(. \mid D_0)$ which do not match peaks of $p_{D_0}(. \mid D)$ produced by $D \setminus D_0$ lead to overfitting error. Peaks of $p_{D_0}(. \mid D)$ produced by $D \setminus D_0$ which do not match peaks of $G(. \mid D_0)$ lead to underfitting error. Overfitting and underfitting error tell us something about the shape of $G(. \mid D_0)$ with respect to $D \setminus D_0$, i.e., to what degree is the prior belief in h compatible with $D \setminus D_0$.

Why are they called "overfitting" and "underfitting" error? Positive contributions to the overfitting error are obtained where peaks of $p(. \mid D_0)$ do not match (or are higher than) peaks of $p_{D_0}(. \mid D)$: there some h will have large probability after training on D_0 but will have lower probability after training on all data D. This is either because D_0 has been approximated too closely, or because of sharp peaks in $G(. \mid D_0)$ — the learning algorithm specializes either on D_0 or on $G(. \mid D_0)$ ("overfitting"). The specialization on D_0 will become even worse if D_0 is corrupted by noise — the case of noisy D_0 will be treated later. Positive contributions to the underfitting error are obtained where peaks of $p_{D_0}(. \mid D)$ do not match (or are higher than) peaks of $p(. \mid D_0)$: there some h will have large probability after training on all data D, but will have lower probability after training on D_0. This is either due to a poor D_0 approximation (note that $p(. \mid D_0)$ is almost fully determined by $G(. \mid D_0)$), or to insufficient information about D conveyed by D_0 ("underfitting"). Either the algorithm did not learn "enough" of D_0, or D_0 does not tell us anything about D. In the latter case, there is nothing we can do — we have to focus on the case where we did not learn enough about D_0.

Analysis of overfitting and underfitting error. $E_G(D, D_0) = E_o(D, D_0) + E_u(D, D_0)$ holds. Note: for zero temperature limit $\beta \to \infty$ we obtain $Z_{D_0}(D) = \sum_{h \in H, D \subset h} G(h \mid D_0)$ and $Z(D_0) = \sum_{h \in H, D_0 \subset h} G(h \mid D_0)$.
$E_o(D, D_0) = \sum_{h \in H, D_0 \subset h} \frac{G(h|D_0)}{Z(D_0)} E(D \setminus D_0, h) = E_G(D, D_0)$. $E_u(D, D_0) = 0$, i.e., there is *no* underfitting error. For $\beta \to 0$ (full stochasticity) we get $E_u(D, D_0) = 0$ and $E_o(D, D_0) = 0$ (recall that E_G is not the conventional but the extended expected generalization error).

Since $D_0 \subset D$, $Z_{D_0}(D, \beta) < Z(D_0, \beta)$ holds. In what follows, averages after learning on D_0 are denoted by $<_{D_0} . >$, and averages after learning on D are denoted by $<_D . >$.

Since $Z_{D_0}(D,\beta) = \sum_{h\in H} G(h \mid D_0)\exp(-\beta E(D_0,h))\exp(-\beta E(D\setminus D_0,h))$, we have
$\frac{Z_{D_0}(D,\beta)}{Z(D_0,\beta)} = \sum_{h\in H} p(h \mid D_0)\exp(-\beta E(D\setminus D_0,h)) = <_{D_0}\exp(-\beta E(D\setminus D_0,.))>$.
Analogously, we have $\frac{Z(D_0,\beta)}{Z_{D_0}(D,\beta)} = <_D \exp(\beta E(D\setminus D_0,.))>$.
Thus, $E_o(D,D_0) = <_{D_0} E(D\setminus D_0,.)> + \frac{1}{\beta}\ln <_{D_0}\exp(-\beta E(D\setminus D_0,.))>$, and
$E_u(D,D_0) = -<_D E(D\setminus D_0,.)> + \frac{1}{\beta}\ln <_D \exp(\beta E(D\setminus D_0,.))>$.[2] With large β, after learning on D_0, E_o measures the difference between average test set error and a minimal test set error. With large β, after learning on D, E_u measures the difference between average test set error and a maximal test set error. So assume we do have a large β (large enough to exceed the minimum of $\frac{1}{\beta}\ln\frac{Z_{D_0}(D,\beta)}{Z(D_0,\beta)}$). *We have to assume that D_0 indeed conveys information about the test set*: preferring hypotheses h with small $E(D_0,h)$ by using a larger β leads to smaller test set error (without this assumption no error decreasing algorithm would make sense). E_u can be decreased by enforcing less stochasticity (by further increasing β), but this will increase E_o. Likewise, decreasing β (enforcing more stochasticity) will decrease E_o but increase E_u. Increasing β decreases the maximal test set error after learning D more than it decreases the average test set error, thus decreasing E_u, and vice versa. Decreasing β increases the minimal test set error after learning D_0 more than it increases the average test set error, thus decreasing E_o, and vice versa. This is the above-mentioned trade-off between stochasticity and fitting the training set, governed by β.

Tolerable error level / Set of acceptable minima. Let us implicitly define a tolerable error level $E_{tol}(\alpha,\beta)$ which, with confidence $1-\alpha$, is the upper bound of the training set error after learning.

$$p(E(D_0,h) \le E_{tol}(\alpha,\beta)) = \sum_{h\in H, E(D_0,h)\le E_{tol}(\alpha,\beta)} p(h \mid D_0) = 1-\alpha. \tag{13}$$

With $(1-\alpha)$-confidence, we have $E(D_0,h) \le E_{tol}(\alpha,\beta)$ after learning. $E_{tol}(\alpha,\beta)$ decreases with increasing β,α. Now we define $M(D_0) := \{h \in H \mid E(D_0,h) \le E_{tol}(\alpha,\beta)\}$, which is the set of acceptable minima — see section 2. *The set of acceptable minima is a set of hypotheses with low underfitting error.* With probability $1-\alpha$, the learning algorithm selects a hypothesis from $M(D_0) \subset H$. Note: for the zero temperature limit $\beta\to\infty$ we have
$E_{tol}(\alpha) = 0$ and $M(D_0) = \{h \in H \mid D_0 \subset h\}$. By fixing a small E_{tol} (or a large β), E_u will be forced to be low.

We would like to have an algorithm decreasing (1) training set error (this corresponds to decreasing underfitting error), and (2) an additional error term, which should be designed to ensure low overfitting error, given a fixed small E_{tol}. The remainder of this section will lead to an answer for the question: *how to design this additional error term?* Since low underfitting is obtained by selecting a hypothesis from $M(D_0)$, in what follows we will focus on $M(D_0)$ only. Using an appropriate choice of prior belief, at the end of this section, we will finally see that the overfitting error can be reduced by an error term expressing preference for flat nets.

[2] We have $-\frac{\partial \ln Z(D_0,\beta)}{\partial\beta} = <_{D_0} E(D_0,.)>$ and $-\frac{\partial \ln Z_{D_0}(D,\beta)}{\partial\beta} = <_D E(D,.)>$. Furthermore, $\frac{\partial^2 \ln Z(D_0,\beta)}{\partial\beta^2} = <_{D_0}(E(D_0,.) - <_{D_0} E(D_0,.)>)^2>$ and $\frac{\partial^2 \ln Z_{D_0}(D,\beta)}{\partial\beta^2} = <_D (E(D,.) - <_D E(D,.)>)^2>$. See also Levin et al. (1990). Using these expressions, it can be shown: by increasing β (starting from $\beta = 0$), we will find a β that minimizes $\frac{1}{\beta}\ln\frac{Z_{D_0}(D,\beta)}{Z(D_0,\beta)} < 0$. Increasing β further makes this expression go to 0.

Relative overfitting error. Let us formally define the *relative overfitting error* E_{ro}, which is the relative contribution of some $h \in M(D_0)$ to the mean overfitting error of hypotheses set $M(D_0)$:

$$E_{ro}(D, D_0, M(D_0), h) = p_{M(D_0)}(h \mid D_0) E(D \setminus D_0, h), \tag{14}$$

where

$$p_{M(D_0)}(h \mid D_0) := \frac{p(h \mid D_0)}{\sum_{h \in M(D_0)} p(h \mid D_0)} \tag{15}$$

for $h \in M(D_0)$, and zero otherwise.

For $h \in M(D_0)$, we approximate $p(h \mid D_0)$ as follows. We assume that $G(h \mid D_0)$ is large where $E(D_0, h)$ is large (trade-off between low $E(D_0, h)$ and $G(h \mid D_0)$). Then $p(h \mid D_0)$ has large values (due to large $G(h \mid D_0)$) where $E(D_0, h) \approx E_{tol}(\alpha, \beta)$ (assuming $E_{tol}(\alpha, \beta)$ is small). We get
$p(h \mid D_0) \approx \frac{G(h|D_0) \exp(-\beta E_{tol}(\alpha,\beta))}{Z(D_0,\beta)}$. The relative overfitting error can now be approximated by

$$E_{ro}(D, D_0, M(D_0), h) \approx \frac{G(h \mid D_0)}{\sum_{h \in M(D_0)} G(h \mid D_0)} E(D \setminus D_0, h). \tag{16}$$

To obtain a distribution over $M(D_0)$, we introduce $G_{M(D_0)}(. \mid D_0)$, the normalized distribution $G(. \mid D_0)$ restricted to $M(D_0)$. For approximation (16) we have

$$E_{ro}(D, D_0, M(D_0), h) \approx G_{M(D_0)}(h \mid D_0) E(D \setminus D_0, h). \tag{17}$$

Prior belief in f and D. Assume D was obtained from a target function f. Let $p(f)$ be the prior on targets and $p(D \mid f)$ the probability of obtaining D with a given f. We have

$$p(f \mid D_0) = \frac{p(D_0 \mid f) p(f)}{p(D_0)}, \tag{18}$$

where $p(D_0) = \sum_{f \in T} p(D_0 \mid f) p(f)$.

The data is drawn from a target function with added noise (the noise-free case is treated below). We don't make any assumptions about the nature of the noise — it does not have to be Gaussian (like, e.g., in MacKay's work, 1992b).

We want to select a $G(. \mid D_0)$ which makes E_{ro} small, i.e., those $h \in M(D_0)$ with small $E(D \setminus D_0, h)$ should have high probabilities $G(h \mid D_0)$.

We don't know $D \setminus D_0$ during learning. D is assumed to be drawn from a target f. We compute the expectation of E_{ro}, given D_0. The probability of the test set $D \setminus D_0$, given D_0, is

$$p(D \setminus D_0 \mid D_0) = \sum_{f \in T} p(D \setminus D_0 \mid f) p(f \mid D_0), \tag{19}$$

where we assume $p(D \setminus D_0 \mid f, D_0) = p(D \setminus D_0 \mid f)$ (we don't remember which exemplars were already drawn). The **expected test set error** $E(., h)$ for some h, given D_0, is

$$\sum_{D \setminus D_0} p(D \setminus D_0 \mid D_0) E(D \setminus D_0, h) = \sum_{f \in T} p(f \mid D_0) \sum_{D \setminus D_0} p(D \setminus D_0 \mid f) E(D \setminus D_0, h). \tag{20}$$

The expected relative overfitting error $E_{ro}(., D_0, M(D_0), h)$ is obtained by inserting equation (20) into equation (17):

$$E_{ro}(., D_0, M(D_0), h) \approx G_{M(D_0)}(h \mid D_0) \sum_{f \in T} p(f \mid D_0) \sum_{D \setminus D_0} p(D \setminus D_0 \mid f) E(D \setminus D_0, h). \tag{21}$$

Minimizing expected relative overfitting error. We define a $G_{M(D_0)}(. \mid D_0)$ such that $G_{M(D_0)}(. \mid D_0)$ has its largest value near small expected test set error $E(.,.)$ (see (17) and (20)). This definition leads to a low expectation of $E_{ro}(., D_0, M(D_0), .)$ (see equation (21)). Define

$$G_{M(D_0)}(h \mid D_0) := \delta\left(argmin_{h' \in M(D_0)}\left(E(., h')\right) - h\right) \ , \tag{22}$$

where δ is the Dirac delta function, which we will use with loose formalism — the context will make clear how the delta function is used.

Using equation (20) we get

$$G_{M(D_0)}(h \mid D_0) = \delta\left(argmin_{h' \in M(D_0)}\left(\sum_{f \in T} p(f \mid D_0) \sum_{D \setminus D_0} p(D \setminus D_0 \mid f) E(D \setminus D_0, h')\right) - h\right) \ . \tag{23}$$

$G_{M(D_0)}(. \mid D_0)$ determines the hypothesis h from $M(D_0)$ that leads to lowest expected test set error. Consequently, we achieve the lowest expected relative overfitting error.

$G_{M(D_0)}$ helps us to define G:

$$G(h \mid D_0) := \frac{\zeta + G_{M(D_0)}(h \mid D_0)}{\sum_{h \in H}(\zeta + G_{M(D_0)}(h \mid D_0))} \ , \tag{24}$$

where $G_{M(D_0)}(h \mid D_0) = 0$ for $h \notin M(D_0)$, and where ζ is a small constant ensuring positive probability $G(h \mid D_0)$ for all hypotheses h.

To appreciate the importance of the prior $p(f)$ in the definition of $G_{M(D_0)}$ (see also equation (29)), in what follows, we will focus on the noise-free case.

The special case of noise-free data. Let $p(D_0 \mid f)$ be equal to $\delta(D_0 \subset f)$ (up to an normalizing constant):

$$p(f \mid D_0) = \frac{\delta(D_0 \subset f) p(f)}{\sum_{f \in T, D_0 \subset f} p(f)} \tag{25}$$

Assume $p(D \setminus D_0 \mid f) = \frac{\delta(D \setminus D_0 \subset f)}{\sum_{D \setminus D_0} \delta(D \setminus D_0 \subset f)}$. Let F be the number of elements in X. $p(D \setminus D_0 \mid f) = \frac{\delta(D \setminus D_0 \subset f)}{2^{F-n}}$. We expand $\sum_{D \setminus D_0} p(D \setminus D_0 \mid f) E(D \setminus D_0, h)$ from equation (20):

$$\begin{aligned} \frac{1}{2^{F-n}} \sum_{D \setminus D_0 \subset f} E(D \setminus D_0, h) \quad &= \quad \frac{1}{2^{F-n}} \sum_{D \setminus D_0 \subset f} \ \sum_{(x,y) \in D \setminus D_0} E((x,y), h) = \\ &\frac{1}{2^{F-n}} \sum_{(x,y) \in f \setminus D_0} E((x,y), h) \sum_{i=1}^{F-n} \binom{F-n-1}{i-1} = \frac{1}{2} E(f \setminus D_0, h). \end{aligned} \tag{26}$$

Here $E((x,y), h) = \| \ y - h(x) \ \|^2$, $E(f \setminus D_0, h) = \sum_{(x,y) \in f \setminus D_0} \| \ y - h(x) \ \|^2$, and $\sum_{i=1}^{F-n} \binom{F-n-1}{i-1} = 2^{F-n-1}$. The factor $\frac{1}{2}$ results from considering the *mean* test set error (where the test set is drawn from f), whereas $E(f \setminus D_0, h)$ is the maximal test set error (obtained by using a maximal test set). From (20) and (26), we obtain the expected test set error $E(., h)$ for some h, given D_0:

$$\sum_{D \setminus D_0} p(D \setminus D_0 \mid D_0) E(D \setminus D_0, h) = \frac{1}{2} \sum_{f \in T} p(f \mid D_0) E(f \setminus D_0, h). \tag{27}$$

From (27) and (17), we obtain the expected $E_{ro}(., D_0, M(D_0), h)$:

$$E_{ro}(., D_0, M(D_0), h) \approx \frac{1}{2} \ G_{M(D_0)}(h \mid D_0) \sum_{f \in T} p(f \mid D_0) E(f \setminus D_0, h). \tag{28}$$

For $G_{M(D_0)}(h \mid D_0)$ we obtain in this noise free case

$$G_{M(D_0)}(h \mid D_0) = \delta \left(argmin_{h' \in M(D_0)} \left(\sum_{f \in T} p(f \mid D_0) E(f \setminus D_0, h') \right) - h \right) . \tag{29}$$

The lowest expected test set error measured by $\frac{1}{2} \sum_{f \in T} p(f \mid D_0) E(f \setminus D_0, h)$. See equation (27).

Noisy data and noise-free data: conclusion. For both the noise-free and the noisy case, equation (18) shows that given D_0 and h, the expected test set error depends on prior target probability $p(f)$.

Choice of prior belief. Now we select some $p(f)$, our prior belief in target f. We introduce a formalism similar to Wolpert's (Wolpert, 1994a). $p(f)$ is defined as the probability of obtaining $f = net(w)$ by choosing a w randomly according to $p(w)$.

Let us first have a look at Wolpert's formalism: $p(f) = \int dw p(w) \delta(net(w) - f)$. By restricting W to W_{inj}, he obtains an injective function $net_{inj} : W_{inj} \rightarrow NET : net_{inj}(w) = net(w)$, which is net restricted to W_{inj}. net_{inj} is surjective (because net is surjective):

$$\begin{aligned} p(f) \quad = \quad & \int_W p(w) \frac{\delta(net_{inj}(w) - f)}{|\det \, net'_{inj}(w)|} |\det \, net'_{inj}(w)| dw = \\ & \int_{NET} p(net_{inj}^{-1}(g)) \frac{\delta(g - f)}{|\det \, net'_{inj}(net_{inj}^{-1}(g))|} dg = \\ & \frac{p(net_{inj}^{-1}(f))}{|\det \, net'_{inj}(net_{inj}^{-1}(f))|}, \end{aligned} \tag{30}$$

where $|\det \, net'_{inj}(w)|$ is the absolute Jacobian determinant of net_{inj}, evaluated at w. If there is a locally flat $net(w) = f$ (flat around w), then $p(f)$ is high.

However, we prefer to follow another path. Our algorithm (flat minimum search) tends to prune a weight w_i if $net(w)$ is very flat in w_i's direction. It prefers regions where $\det net'(w) = 0$ (where many weights lead to the same net function). Unlike Wolpert's approach, ours distinguishes the probabilities of targets $f = net(w)$ with $\det net'(w) = 0$. The advantage is: we do not only search for $net(w)$ which are flat in one direction but for $net(w)$ which are flat in many directions (this corresponds to a higher probability of the corresponding targets). Define

$$net^{-1}(g) := \{w \in W \mid net(w) = g\} \tag{31}$$

and

$$p(net^{-1}(g)) := \sum_{w \in net^{-1}(g)} p(w). \tag{32}$$

We have

$$p(f) = \frac{\sum_{g \in NET} p(net^{-1}(g)) \delta(g - f)}{\sum_{f \in T} \sum_{g \in NET} p(net^{-1}(g)) \delta(g - f)} = \frac{p(net^{-1}(f))}{\sum_{f \in T} p(net^{-1}(f))}. \tag{33}$$

net partitions W into equivalence classes. To obtain $p(f)$, we compute the probability of w being in the equivalence class $\{w \mid net(w) = f\}$, if randomly chosen according to $p(w)$. An equivalence class corresponds to a net function, i.e., net maps all w of an equivalence class to the same net function.

Relation to FMS algorithm. FMS (from section 3) works locally in weight space W. Let w' be the actual weight vector found by FMS (with $h = net(w')$). Recall the definition of

$G_{M(D_0)}(h \mid D_0)$ (see (22) and (23)): we want to find a hypothesis h which best approximates those f with large $p(f)$ (the test data has high probability of being drawn from such targets). We will see that those $f = net(w)$ with flat $net(w)$ locally have high probability $p(f)$. Furthermore we will see that a w' close to w with flat $net(w)$ has flat $net(w')$, too. To approximate such targets f, the only thing we can do is find a w' close to many w with $net(w) = f$ and large $p(f)$. To justify this approximation (see definition of $p(f \mid D_0)$ while recalling that $h \in G_{M(D_0)}$), we assume (1) that the noise has mean 0, and (2) that small noise is more likely than large noise (e.g., Gaussian, Laplace, Cauchy distributions).

To restrict $p(f) = p(net(w))$ to a local range in W, we define regions of equal net functions $F(w) = \{\bar{w} \mid \forall_\tau 0 \leq \tau \leq 1, w + \tau(\bar{w} - w) \in W : net(w) = net(w + \tau(\bar{w} - w))\}$.
Note: $F(w) \subset net^{-1}(net(w))$. If $net(w)$ is flat along long distances in many directions $\bar{w} - w$, then $F(w)$ has many elements. Locally in weight space, at w' with $h = net(w')$, for $\gamma > 0$ we define: if the minimum $w = argmin_{\bar{w}}\{\parallel \bar{w} - w' \parallel \mid \parallel \bar{w} - w' \parallel < \gamma, net(\bar{w}) = f\}$ exists, then $p_{w',\gamma}(f) = c\, p(F(w))$, where c is a constant. If this minimum does not exist, then $p_{w',\gamma}(f) = 0$. $p_{w',\gamma}(f)$ locally approximates $p(f)$. During search for w' (corresponding to a hypothesis $h = net(w')$), to locally decrease the expected test set error (see equation (20)), we want to enter areas where many large $F(w)$ are near w' in weight space. We wish to decrease the test set error, which is caused by drawing data from highly probable targets f (those with large $p_{w',\gamma}(f)$). We do not know, however, which w's are mapped to target's f by $net(.)$. Therefore, we focus on $F(w)$ (w near w' in weight space), instead of $p_{w',\gamma}(f)$. Assume $\parallel w' - w \parallel$ is small enough to allow for a Taylor expansion, and that $net(w')$ is flat in direction $(\bar{w} - w')$:
$net(w) = net(w' + (w - w')) = net(w') + \nabla net(w')(w - w') + \frac{1}{2}(w - w')H(net(w'))(w - w') + \ldots$, where $H(net(w'))$ is the Hessian of $net(.)$ evaluated at w', $\nabla net(w)(\bar{w} - w') = \nabla net(w')(\bar{w} - w') + O(w - w')$, and $(\bar{w} - w')H(net(w))(\bar{w} - w') = (\bar{w} - w')H(net(w'))(\bar{w} - w') + O(w - w')$ (analogously for higher order derivatives). We see: in a small environment of w', there is flatness in direction $(\bar{w} - w')$, too. Likewise, if $net(w')$ is not flat in any direction, this property also holds within a small environment of w'. Only near w' with flat $net(w')$, there may exist w with large $F(w)$. Therefore, it is reasonable to search for a w' with $h = net(w')$, where $net(w')$ is flat within a large region. This means to search for the h determined by $G_{M(D_0)}(. \mid D_0)$ of equation (22). Since $h \in M(D_0)$, $E(D_0, net(w')) \leq E_{tol}$ holds: we search for a w' living within a large connected region, where for all w within this region $E(net(w'), net(w), X) = \sum_{x \in X} \parallel net(w')(x) - net(w)(x) \parallel^2 \leq \epsilon$, where ϵ is defined in section 2. **To conclude:** we decrease the relative overfitting error and the underfitting error by searching for a **flat minimum** (see definition of flat minima in section 2).

Practical realization of the Gibbs variant.
(1) Select α and $E_{tol}(\alpha, \beta)$, thus implicitly choosing β.
(2) Compute the set $M(D_0)$.
(3) Assume we know how data is obtained from target f, i. e. we know $p(D_0 \mid f)$, $p(D \setminus D_0 \mid f)$, and the prior $p(f)$. Then we can compute $G_{M(D_0)}(. \mid D_0)$ and $G(. \mid D_0)$.
(4) Start with $\beta = 0$ and increase β until equation (13) holds. Now we know the β from the implicit choice above.
(5) Since we know all we need to compute $p(h \mid D_0)$, select some h according to this distribution.

Three comments on certain FMS limitations.
1. *FMS only approximates* the Gibbs variant given by the definition of $G_{M(D_0)}(h \mid D_0)$ (see (22) and (23)).
We only *locally* approximate $p(f)$ in weight space. If $f = net(w)$ is locally flat around w then there exist units or weights which can be given with low precision (or can be removed). If there are other weights w_i with $net(w_i) = f$, then one may assume that there are also points in weight space near such w_i where weights can be given with low precision (think of, e.g., symmetrical exchange of weights and units). We assume the local approximation of $p(f)$ is good. The most probable targets represented by flat $net(w)$ are approximated by a hypothesis h which is also

represented by a flat $net(w')$ (where w' is near w in weight space). To allow for approximation of $net(w)$ by $net(w')$, we have to assume that the hypothesis set H is dense in the target set T. If $net(w')$ is flat in many directions then there are many $net(w) = f$ that share this flatness and are well-approximated by $net(w')$. The only reasonable thing FMS can do is to make $net(w')$ as flat as possible in a large region around w', to approximate the $net(w)$ with large prior probability (recall that flat regions are approximated by axis-aligned boxes, as discussed in section 7, paragraph entitled "Generalized boxes?"). This approximation is fine if $net(w')$ is smooth enough in "unflat" directions (small changes in w' should not result in quite different net functions).

2. *Concerning point (3) above:*
$p(f \mid D_0)$ depends on $p(D_0 \mid f)$ (how the training data is drawn from the target, see (18)). $G_{M(D_0)}(h \mid D_0)$ depends on $p(f \mid D_0)$ and $p(D \setminus D_0 \mid f)$ (how the test data is drawn from the target). Since we do not know how the data is obtained, the quality of the approximation of the Gibbs algorithm may suffer from noise which has not mean 0, or from large noise being more probable than small noise.

Of course, if the choice of prior belief does not match the true target distribution, the quality of $G_{M(D_0)}(h \mid D_0)$'s approximation will suffer as well.

3. *Concerning point (5) above:*
FMS outputs only a single h instead of $p(h \mid D_0)$. This issue is discussed in section 7 (paragraph entitled "multiple initializations?").

To conclude: Our FMS algorithm from section 3 only *approximates* the Gibbs algorithm variant. Two important assumptions are made: The first is that an appropriate choice of prior belief has been made. The second is that the noise on the data is not too "weird" (mean 0, small noise more likely). *The two assumptions are necessary for any algorithm based on an additional error term besides the training error.* The approximations are: $p(f)$ is approximated locally in weight space, and flat $net(w)$ are approximated by flat $net(w')$ with w' near w's. Our Gibbs variant takes into account that FMS uses only X_0 for computing flatness.

A.2. WHY DOES THE HESSIAN DECREASE?

Outline. This section shows that second order derivatives of the output function vanish during flat minimum search. This justifies the linear approximations in section 4.

Intuition. We show that the algorithm tends to suppress the following values: (1) unit activations, (2) first order activation derivatives, (3) the sum of all contributions of an arbitrary unit activation to the net output. Since weights, inputs, activation functions, and their first and second order derivatives are bounded, the entries in the Hessian decrease where the corresponding $|\delta w_{ij}|$ increase.

Formal details. We consider a strictly layered feedforward network with K output units and g layers. We use the same activation function f for all units. For simplicity, in what follows we focus on a single input vector x_p. x_p (and occasionally w itself) will be notationally suppressed. We have

$$\frac{\partial y^l}{\partial w_{ij}} = f'(s_l) \left\{ \begin{array}{cc} y^j & \text{for } i = l \\ \sum_m w_{lm} \frac{\partial y^m}{w_{ij}} & \text{for } i \neq l \end{array} \right\} , \tag{34}$$

where y^a denotes the activation of the a-th unit, and $s_l = \sum_m w_{lm} y^m$.

The last term of equation (1) (the "regulator") expresses output sensitivity (to be minimized) with respect to simultaneous perturbations of all weights. "Regulation" is done by equalizing the sensitivity of the output units with respect to the weights. The "regulator" does not influence the same particular units or weights for each training example. It may be ignored for the purposes of this section. Of course, the same holds for the first (constant) term in (1). We are left with the second term. With (34) we obtain:

$$\sum_{i,j} \log \sum_k \left(\frac{\partial o^k}{\partial w_{ij}}\right)^2 =$$

$$2 \sum_{\text{unit } k \text{ in the } g \text{ th layer}} (\text{fan-in of unit } k) \log |f'(s_k)| +$$

$$2 \sum_{\text{unit } j \text{ in the } (g-1)\text{th layer}} (\text{fan-out of unit } j) \log |y^j| +$$

$$\sum_{\text{unit } j \text{ in the } (g-1)\text{th layer}} (\text{fan-in of unit } j) \log \sum_k \left(f'(s_k) w_{kj}\right)^2 +$$

$$2 \sum_{\text{unit } j \text{ in the } (g-1)\text{th layer}} (\text{fan-in of unit } j) \log |f'(s_j)| +$$

$$2 \sum_{\text{unit } j \text{ in the } (g-2)\text{th layer}} (\text{fan-out of unit } j) \log |y^j| +$$

$$\sum_{\text{unit } j \text{ in the } (g-2)\text{th layer}} (\text{fan-in of unit } j) \log \sum_k \left(f'(s_k) \sum_l f'(s_l) w_{kl} w_{lj}\right)^2 +$$

$$2 \sum_{\text{unit } j \text{ in the } (g-2)\text{th layer}} (\text{fan-in of unit } j) \log |f'(s_j)| +$$

$$2 \sum_{\text{unit } j \text{ in the } (g-3)\text{th layer}} (\text{fan-out of unit } j) \log |y^j| +$$

$$\sum_{i,j, \text{ where unit } i \text{ in a layer} < (g-2)} \log \sum_k \left(f'(s_k) \sum_{l_1} f'(s_{l_1}) w_{kl_1} \sum_{l_2} w_{l_1 l_2} \frac{\partial y^{l_2}}{\partial w_{ij}}\right)^2 \qquad (35)$$

Let us have a closer look at this equation. We observe:
(1) Activations of units decrease in proportion to their fan-outs.
(2) First order derivatives of the activation functions decrease in proportion to their fan-ins.
(3) A term of the form $\sum_k \left(f'(s_k) \sum_{l_1} f'(s_{l_1}) w_{kl_1} \sum_{l_2} \ldots \sum_{l_r} f'(s_{l_r}) w_{l_{r-1} l_r} w_{l_r j}\right)^2$ expresses the sum of unit j's squared contributions to the net output. Here r ranges over $\{0, 1, \ldots, g-2\}$, and unit j is in the $(g-1-r)$th layer (for the special case $r = 0$, we get $\sum_k \left(f'(s_k) w_{kj}\right)^2$). These terms also decrease in proportion to unit j's fan-in. Analogously, equation (35) can be extended to the case of additional layers.

Comment. Let us assume that $f'(s_j) = 0$ and $f(s_j) = 0$ is "difficult to achieve" (can be achieved only by fine-tuning all weights on connections to unit j). Instead of minimizing $|f(s_j)|$ or $|f'(s_j)|$ by adjusting the net input of unit j (this requires fine-tuning of many weights), our algorithm prefers pushing weights w_{kl} on connections to output units towards zero (other weights are less affected). On the other hand, if $f'(s_j) = 0$ and $f(s_j) = 0$ is **not** "difficult to achieve", then, **unlike weight decay, our algorithm does not necessarily prefer weights close to zero.** Instead, it prefers (possibly very strong) weights which push $f(s_j)$ or $f'(s_j)$ towards zero (e.g., with sigmoid units active in [0,1]: strong inhibitory weights are preferred; with Gaussian units: high absolute weight values are preferred). See the experiment in section 5.2.

How does this influence the Hessian? The entries in the Hessian corresponding to output o^k can be written as follows:

$$\frac{\partial^2 o^k}{\partial w_{ij} \partial w_{uv}} = \frac{f''(s_k)}{(f'(s_k))^2} \frac{\partial o^k}{\partial w_{ij}} \frac{\partial o^k}{\partial w_{uv}} + f'(s_k) \left(\sum_l w_{kl} \frac{\partial^2 y^l}{\partial w_{ij} \partial w_{uv}} + \bar{\delta}_{ik} \frac{\partial y^j}{\partial w_{uv}} + \bar{\delta}_{uk} \frac{\partial y^v}{\partial w_{ij}}\right) , \qquad (36)$$

where $\bar{\delta}$ is the Kronecker-Delta. Searching for big boxes, we run into regions of acceptable minima with o^k's close to target (section 2). Thus, by scaling the targets, $\frac{f''(s_k)}{(f'(s_k))^2}$ can be bounded. Therefore, the first term in equation (36) decreases during learning.

According to the analysis above, the first order derivatives in the second term of (36) are pushed towards zero. So are the w_{kl} of the sum in the second term of (36).

The only remaining expressions of interest are second order derivatives of units in layer $(g-1)$. The $\frac{\partial^2 y^l}{\partial w_{ij}\partial w_{uv}}$ are bounded if (a) the weights, (b) the activation functions, (c) their first and second order derivatives, and (d) the inputs are bounded. This is indeed the case, as will be shown for networks with one or two hidden layers:

Case 1: For unit l in a single hidden layer ($g = 3$), we obtain

$$|\frac{\partial^2 y^l}{\partial w_{ij}\partial w_{uv}}| = |\bar{\delta}_{li}\bar{\delta}_{lu} f''(s_l) y^j y^v| < C_1 \ , \tag{37}$$

where y^j, y^v are the components of an input vector x_p, and C_1 is a positive constant.

Case 2: For unit l in the third layer of a net with 2 hidden layers ($g = 4$), we obtain

$$|\frac{\partial^2 y^l}{\partial w_{ij}\partial w_{uv}}| = |f''(s_l)(w_{li}y^j + \bar{\delta}_{il}y^j)(w_{lu}y^v + \bar{\delta}_{ul}y^v) +$$
$$f'(s_l)\left(w_{li}\bar{\delta}_{iu}f''(s_i)y^j y^v + \bar{\delta}_{il}\bar{\delta}_{uj}f'(s_j)y^v + \bar{\delta}_{ul}\bar{\delta}_{iv}f'(s_v)y^j\right)| < C_2 \ , \tag{38}$$

where C_2 is a positive constant. Analogously, the boundedness of second order derivatives can be shown for additional hidden layers.

Conclusion: As desired, our algorithm makes the $H^k_{ij,uv}$ decrease where $|\delta w_{ij}|$ or $|\delta w_{uv}|$ increase.

A.3. EFFICIENT IMPLEMENTATION OF THE ALGORITHM

Outline. We first explicitly compute the derivatives of (1). Then we show how to use Pearlmutter and Møller's algorithm to speed up the computation of second order terms (A.3.2).

For simplicity, in what follows we focus on a single input vector x_p. Again, x_p (and occasionally w itself) will be notationally suppressed.

A.3.1 EXPLICIT DERIVATIVE OF EQUATION (1)

The derivative of the right-hand side of (1) is:

$$\frac{\partial B(w,x_p)}{\partial w_{uv}} = \sum_{i,j} \frac{\sum_k \frac{\partial o^k}{\partial w_{ij}} \frac{\partial^2 o^k}{\partial w_{ij}\partial w_{uv}}}{\sum_m (\frac{\partial o^m}{\partial w_{ij}})^2} +$$
$$L \frac{\sum_k \left(\sum_{i,j} \frac{|\frac{\partial o^k}{\partial w_{ij}}|}{\sqrt{\sum_m (\frac{\partial o^m}{\partial w_{ij}})^2}} \sum_{i,j} \left(\mathrm{sign}(\frac{\partial o^k}{\partial w_{ij}}) \frac{\frac{\partial^2 o^k}{\partial w_{ij}\partial w_{uv}} \sum_m (\frac{\partial o^m}{\partial w_{ij}})^2 - \frac{\partial o^k}{\partial w_{ij}} \sum_m \frac{\partial o^m}{\partial w_{ij}} \frac{\partial^2 o^m}{\partial w_{ij}\partial w_{uv}}}{(\sum_m (\frac{\partial o^m}{\partial w_{ij}})^2)^{\frac{3}{2}}} \right) \right)}{\sum_k \left(\sum_{i,j} \frac{|\frac{\partial o^k}{\partial w_{ij}}|}{\sqrt{\sum_m (\frac{\partial o^m}{\partial w_{ij}})^2}} \right)^2} \ . \tag{39}$$

To compute (2), we need

$$\frac{\partial B(w,x_p)}{\partial (\frac{\partial o^k}{\partial w_{ij}})} = \frac{\frac{\partial o^k}{\partial w_{ij}}}{\sum_m (\frac{\partial o^m}{\partial w_{ij}})^2} +$$
$$L \frac{\sum_m \left(\sum_{l,r} \left(\frac{|\frac{\partial o^m}{\partial w_{lr}}|}{\sqrt{\sum_{\bar{m}} (\frac{\partial o^{\bar{m}}}{\partial w_{lr}})^2}} \right) \mathrm{sign}(\frac{\partial o^m}{\partial w_{ij}}) \frac{\bar{\delta}_{mk} \sum_{\bar{m}} (\frac{\partial o^{\bar{m}}}{\partial w_{ij}})^2 - \frac{\partial o^m}{\partial w_{ij}} \frac{\partial o^k}{\partial w_{ij}}}{(\sum_{\bar{m}} (\frac{\partial o^{\bar{m}}}{\partial w_{ij}})^2)^{\frac{3}{2}}} \right)}{\sum_m \left(\sum_{l,r} \frac{|\frac{\partial o^m}{\partial w_{lr}}|}{\sqrt{\sum_{\bar{m}} (\frac{\partial o^{\bar{m}}}{\partial w_{lr}})^2}} \right)^2} \ , \tag{40}$$

where $\bar{\delta}$ is the Kronecker-Delta. Using the nabla operator and (40), we can compress (39):

$$\nabla_{uv} B(w, x_p) = \sum_k H^k (\nabla_{\frac{\partial o^k}{\partial w_{ij}}} B(w, x_p)) \ , \tag{41}$$

where H^k is the Hessian of the output o^k. Since the sums over l, r in (40) need to be computed only once (the results are reusable for all i, j), $\nabla_{\frac{\partial o^k}{\partial w_{ij}}} B(w, x_p)$ can be computed in $O(L)$ time. The product of the Hessian and a vector can be computed in $O(L)$ time (see next section). With constant number of output units, **the computational complexity of our algorithm is** $O(L)$.

A.3.2. FAST MULTIPLICATION BY THE HESSIAN

Pearlmutter (1994) and Møller (1993) compute the product of a vector and the Hessian of the error in $O(L)$ time. Using Pearlmutter's notation, we do the same with the Hessian of the output. An operator R is defined as follows:

$$R_y\{g(x)\} \equiv \frac{\partial}{\partial t} g(x + ty) \mid_{t=0} \ . \tag{42}$$

The Hessian of the kth output o^k of a feedforward net is computed in 3 successive passes:

1. First backward pass ($y^l = o^k$):

$$\frac{\partial y^l}{\partial y^i} = \left\{ \begin{array}{cc} 1 & \text{for } i = l \\ \sum_j w_{ji} \frac{\partial y^l}{\partial s_j} & \text{for } i \neq l \end{array} \right\} \ , \tag{43}$$

$$\frac{\partial y^l}{\partial s_i} = f_i'(s_i) \frac{\partial y^l}{\partial y^i} \ , \tag{44}$$

$$\frac{\partial y^l}{\partial w_{ji}} = y^i \frac{\partial y^l}{\partial s_j} \ . \tag{45}$$

2. First forward pass:

$$R\{s_i\} = \sum_j (w_{ij} R\{y^j\} + \frac{\partial B(w, x_p)}{\partial (\frac{\partial o^k}{\partial w_{ij}})} y^j) \ , \tag{46}$$

$$R\{y^i\} = \left\{ \begin{array}{cc} 0 & \text{for } y^i \text{ input} \\ R\{s_i\} f_i'(s_i) & \text{otherwise} \end{array} \right\} \ . \tag{47}$$

3. Second backward pass ($y^l = o^k$):

$$R\{\frac{\partial y^l}{\partial y^i}\} = \left\{ \begin{array}{cc} 0 & \text{for } y^i \text{ in layers not below } y^l \\ \sum_j \left(w_{ji} R\{\frac{\partial y^l}{\partial s_j}\} + \frac{\partial B(w, x_p)}{\partial (\frac{\partial o^k}{\partial w_{ji}})} \frac{\partial y^l}{\partial s_j} \right) & \text{for } y^i \text{ in layers below } y^l \end{array} \right\} \ , \tag{48}$$

$$R\{\frac{\partial y^l}{\partial s_i}\} = f_i'(s_i) R\{\frac{\partial y^l}{\partial y^i}\} + R\{s_i\} f_i''(s_i) \frac{\partial y^l}{\partial y^i} \ , \tag{49}$$

$$R\{\frac{\partial y^l}{\partial w_{ji}}\} = y^i R\{\frac{\partial y^l}{\partial s_j}\} + R\{y^i\} \frac{\partial y^l}{\partial s_j} \ . \tag{50}$$

The elements of the vector $H^k(\nabla_{\frac{\partial o^k}{\partial w_{ij}}} B(w, x_p))$ are $R\{\frac{\partial o^k}{\partial w_{ji}}\}$ (see (41)). Using the technique in (Pearlmutter, 1994), recurrent networks can be dealt with as well.

A.4. PSEUDO CODE OF THE ALGORITHM

Below the algorithm in pseudo code (using fast multiplication as in appendix A.3.2). Comments are marked by "**". *Note:* the pseudo code was omitted from the version for *Neural Computation*. *We recommend not to blindly reimplement the algorithm from the pseudo code, but to make a serious effort to understand it, by consulting the body of the paper as well. We believe that this will greatly facilitate proper, problem-specific use of the algorithm.*

Notation. In what follows, the variable integers i and j stand for units.
Variables k, m, k_1 are reserved for output units only.
g is the number of layers, where the gth layer is the output layer and the 1st layer is the input layer.
The current pattern consists of input vector x and target vector t (see section 2).
$x[j]$ is the component of the input vector corresponding to input unit j.
$t[k]$ is the component of the output vector corresponding to output unit k.
$w[i][j]$ is the real-valued weight on the connection from unit j to unit i (see w_{ij} in section 2).
$s[j]$ is the net input of unit j (see equation (34) and text thereafter).
f_j is the activation function of unit j, f'_j is the first derivative, f''_j is the second derivative of f_j (see appendix A.2).
$y[j]$ is the activation of the j-th unit (see appendix A.2).
$error[k]$ is the error of the k-th output unit.
$ky[k][j]$ is $\frac{\partial y[k]}{\partial y[j]}$ (see equation (43)).
$ks[k][j]$ is $\frac{\partial y[k]}{\partial s[j]}$ (see equation (44)).
$ykw[k][i][j]$ is $\frac{\partial y[k]}{\partial w[i][j]}$ (see equation (45)).
$yw[i][j]$ is $\frac{\partial E}{\partial w[i][j]} = \nabla_w E$, the gradient of the quadratic error.
abs(x) denotes the absolute value of real x.
kron$(m = k)$ returns 1 if $m = k$ and 0 otherwise.
sign(x) returns the sign of real x.
$t1[][], t2[], t3, t4$ are variables (used to compute the right hand side of equation (40)).
$t1[i][j] = \sum_{k \text{ is output unit}} \left(\frac{\partial y[k]}{\partial w[i][j]}\right)^2 = \sum_{k \text{ is output unit}} (ykw[k][i][j])^2$.
$t2[k] = \sum_{w[i][j]} \frac{\text{abs}\left(\frac{\partial y[k]}{\partial w[i][j]}\right)}{\sqrt{\sum_{k \text{ is output unit}} \left(\frac{\partial y[k]}{\partial w[i][j]}\right)^2}} = \sum_{w[i][j]} \frac{\text{abs}(ykw[k][i][j])}{\sqrt{t1[i][j]}}$ (see the sums over l, r in (40)).
$t3 = \sum_{k \text{ is output unit}} \left(\sum_{w[i][j]} \frac{\text{abs}\left(\frac{\partial y[k]}{\partial w[i][j]}\right)}{\sqrt{\sum_{k \text{ is output unit}} \left(\frac{\partial y[k]}{\partial w[i][j]}\right)^2}} \right)^2 = \sum_{k \text{ is output unit}} (t2[k])^2$ (see the denumerator in the second line of equation (40)).
$t4 = \sum_{m \text{ is output unit}} \sum_{w[l][r]} \frac{\text{abs}\left(\frac{\partial y[m]}{\partial w[l][r]}\right)}{\sqrt{\sum_{k_1 \text{ is output unit}} \left(\frac{\partial y[k_1]}{\partial w[l][r]}\right)^2}}$
$\text{sign}\left(\frac{\partial y[m]}{\partial w[i][j]}\right) \frac{\text{kron}(m=k) \sum_{k_1 \text{ is output unit}} \left(\frac{\partial y[k_1]}{\partial w[i][j]}\right)^2 - \frac{\partial y[m]}{\partial w[i][j]} \frac{\partial y[k]}{\partial w[i][j]}}{\left(\sum_{k_1 \text{ is output unit}} \left(\frac{\partial y[k_1]}{\partial w[i][j]}\right)^2\right)^{\frac{3}{2}}} =$
$\sum_{m \text{ is output unit}} t2[m] \ \text{sign}(ykw[m][i][j]) \frac{\text{kron}(m=k) t1[i][j] - ykw[m][i][j] \ ykw[k][i][j]}{(t1[i][j])^{\frac{3}{2}}}$ (see the numerator in the second line of equation (40)).

weights stands for the number of weights that make a significant contribution to the computation of the current pattern.
$\delta w[i][j]$ is an approximation of $w[i][j]$'s precision (approximation because ϵ is unknown).
$insignificant[i][j]$ marks whether or not $w[i][j]$ provides a significant contribution to the computation of the current pattern.

$b[k][i][j]$ is $\frac{\partial B}{\partial\left(\frac{\partial y[k]}{\partial w[i][j]}\right)}$ (see equation (40)).
$rs[k][i]$ is $R\{s[i]\}$ (see equation (46)).
$ry[k][i]$ is $R\{y[i]\}$ (see equation (47)).
$rdks[i]$ is $R\{\frac{\partial y[k]}{\partial s[i]}\}$ (see equation (49)).
$rdky[i]$ is $R\{\frac{\partial y[k]}{\partial y[i]}\}$ (see equation (48)).
$rdw[i][j]$ is $R\{\frac{\partial y[k]}{\partial w[i][j]}\} = \nabla_w B = \sum_k H^k(\nabla_{\frac{\partial o^k}{\partial w_{ij}}} B)$, the gradient of the additional error term B (see equation (50)).
E is the current pattern's quadratic error (see section 2 and appendix A.1).
α is the learning rate for the quadratic error.
λ is the learning rate for the additional error term (the following values are used to make *lambda* updates according to Weigend et al., 1991, see also section 5.6).
$\Delta\lambda$ is a parameter needed for updating λ.
σ is a parameter needed for updating λ, typical value is 0.5.
E_o is the most recent epoch's average error.
E_n is the current epoch's average error.
E_a is the exponentially weighted average epoch error.
γ is the parameter for the exponentially weighted error — a typical value is 0.9 or 0.99, depending on training set size.
E_{tol} is the tolerable error level – it depends on the task to be solved (see section 2 and appendix A.1, equation (13)).
exemplars is the number of exemplars observed during training.
epochlength is the length of an epoch, measured in number of presented training patterns.
$epochs = exemplars \ \% \ epochlength$ is the current number of epochs so far, where % represents integer division.
$lyw, lrdw, scale$ are variables required for normalizing B' gradient to the length of E's gradient.
$wasalive[i][j]$ is TRUE if $w[i][j]$ was alive for at least one pattern presented during the previous epoch, and FALSE otherwise.
$alive[i][j]$ is TRUE if $wasalive[i][j]$ is TRUE *and* if $alive[i][j]$ was always TRUE during all epochs since $w[i][j]$ was set alive for the last time (otherwise $alive[i][j]$ is FALSE).
$\leftarrow$ denotes the assignment operator.

Additional comments.

- For simplicity, the description of the algorithm neglects bias weights and "true units".
- Targets should be scaled to bound first order derivatives of output units – see text after equation (36) in A.2 (e.g., for sigmoids active in $[0, 1]$ scale targets to range $[0.2, 0.8]$).
- Removing more than one weight ($alive[i][j]$ $\leftarrow$FALSE) at a time may cause the error to increase. Removing only one weight at a time leads to smoother performance improvement.
- To prevent accidental weight removal in case of small training sets, we recommend not to use too many near-zero inputs — weights from such inputs may be evaluated $insignificant$ despite being significant.
- Likewise, the random weight initialization in the beginning of the learning phase may cause accidental weight removal due to small, random, initial derivatives. This can be prevented by keeping all weights alive for a certain initial time interval.
- Initially, K's value does not yet have a sensible interpretation. One may start with a large K and decrease it as the error decreases.
- For each pattern, there is a minimal $\delta w[i][j]$ (stored in δ_{min}). δ_{min} represents weight precision required for significant weights.

Speeding up the algorithm. It makes sense to separate the algorithm into two phases. Phase 1 is conventional backprop, phase 2 is FMS. The backprop phase consists of the forward pass, the first backward pass, and the weight update based on $\lambda = 0$ (marked in the algorithm). Start with phase 1. Switch to phase 2 if $E_a < 0.9\ E_{tol}$. Switch again to phase 1 if $E_a > 1.1\ E_{tol}$ (the values 0.9 and 1.1 can be changed).
Two-phase learning does not sensitively depend on λ (but avoid λ values that are always too small). Two-phase learning is justified because weights with large $\delta w[i][j]$ (and small $\frac{\partial E}{\partial w[i][j]}$) hardly influence E's gradient — it makes sense to let FMS focus on low-precision weights, and let backprop focus on the others.

ALGORITHM.

Initialization. Set $K = 10^2$ or $K = 10^3$ (the exponent is the difference between the numbers of significant digits required for maximal and minimal precision).
Set ϵ to an arbitrary small value.
Set $exemplars$ and $epochs$ to 0.
Initialize $w[i][j]$ for all i, j.
Initialize λ, α (typically, $\lambda = 0$), and provide a criterion for stopping the learning procedure.
Set $\gamma, \sigma, \Delta\lambda$, for instance, $\gamma = 0.9$ or $0.99, \sigma = 0.5, \Delta\lambda = 0.01\ \alpha$ or $\Delta\lambda = 0.001\ \alpha$.
Set E_{tol} to some desired error value after learning.
Set E, E_a, E_o, E_n to 0.
Set $epochlength$.
Set $exemplars, epochs$ to 0.
Set $alive[i][j]$ = TRUE for all i, j.
Set $wasalive[i][j]$ = FALSE for all i, j.
Set δ_{min} to a large value.
Set $alive[i][j]$ = FALSE for non-existing $w[i][j]$.

While training not stopped do
begin(while)

select pattern pair (x, t).
** The following variables can be set after they were used for the last time in the previous loop. **
set all components of $s[], yw[][], ky[][], t1[][], t2[], rs[][], rdw[][], rdky[]$ to 0
set all components of $insignificant[][]$ to FALSE
set $t3, t4, E$ to 0

** **Forward pass.** **

for all input units j do
begin
| $y[j] \leftarrow x[j]$
end

```
for u = 2 to g do
begin
|     for all units i in layer u do
|     begin
|     |     for all units j in layer u − 1 do
|     |     begin
|     |     |     if (alive[i][j]) do
|     |     |     begin
|     |     |     |     s[i] ← s[i] + w[i][j] y[j]
|     |     |     end
|     |     end
|     |     y[i] ← f_i(s[i])
|     end
end
```

** **Compute the error.** **

```
for all output units k do
begin
|     error[k] ← t[k] − y[k]
|     E ← E + (error[k])^2
end

E_n ← E_n + E
```

** Compute the error gradient **

** **1. backward pass.** **

```
for all output units k do
begin
|     ks[k][k] ← f'_k(s[k])
|     for u = 1 to g − 1 do
|     begin
|     |     for all units j in layer g − u do
|     |     begin
|     |     |     (IF u ≠ 1 THEN: for all units i in layer g − u + 1 ELSE: i = k) do
|     |     |     begin
|     |     |     |     if (alive[i][j]) do
|     |     |     |     begin
|     |     |     |     |     ykw[k][i][j] ← y[j] ks[k][i]
|     |     |     |     |     set abs(ykw[k][i][j]) > 1E-5 ** to avoid division overflow **
|     |     |     |     |     yw[i][j] ← yw[i][j] + ykw[k][i][j] error[k]
|     |     |     |     |     ky[k][j] ← ky[k][j] + w[i][j] ks[k][i]
|     |     |     |     |     t1[i][j] ← t1[i][j] + (ykw[k][i][j])^2
|     |     |     |     end
|     |     |     end
|     |     |     ks[k][j] ← f'_j(s[j])ky[k][j]
|     |     end
|     end
end
```

** **End of conventional backprop (phase 1).** **

** **compute** $b[k][i][j] = \frac{\partial B}{\partial\left(\frac{\partial o^k}{\partial w_{ij}}\right)}$ **

** we recommend to introduce additional local variables for inner loops, such as $h1 = \sqrt{t1[i][j]}$ and $h2 = ykw[k][i][j]\ /\ t1[i][j]$ **

```
for all output units k do
begin
|     for all i, j, such that (alive[i][j]) do
|     begin
|     |     t2[k] ← t2[k] + abs(ykw[k][i][j]) / √(t1[i][j])
|     end
|     t3 ← t3 + (t2[k])^2
end
```

** some weights are insignificant to compute the current pattern **

```
for all i, j, such that (alive[i][j]) do
begin
|     δw[i][j] ← √ϵ / (√(t1[i][j]) √t3)
|     if (δw[i][j] < δ_min ) do
|     begin
|     |     δ_min ← δw[i][j]
|     end
end

weights ← 0

for all i, j, such that (alive[i][j]) do
begin
|     if (δw[i][j] > K δ_min ) do
|     begin
|     |     insignificant[i][j] ← TRUE
|     |     for all output units k do
|     |     begin
|     |     |     t1[i][j] ← t1[i][j] − (ykw[k][i][j])^2
|     |     end
|     end
|     else do
|     begin
|     |     weights ← weights + 1
|     |     wasalive[i][j] ← TRUE
|     end
end
```

** update variables after having marked the current pattern's insignificant weights **
$t3 \leftarrow 0$
for all output units k do
begin
| $t2[k] \leftarrow 0$
| for all i, j, such that ($alive[i][j]$ AND NOT $insignificant[i][j]$) do
| begin
| | $t2[k] \leftarrow t2[k] + \text{abs}(ykw[k][i][j])/\sqrt{t1[i][j]}$
| end
| $t3 \leftarrow t3 + (t2[k])^2$
end

for all output units k do
begin
| for all i, j, such that ($alive[i][j]$ AND NOT $insignificant[i][j]$) do
| begin
| | $t4 \leftarrow 0$
| | for all output units m do
| | begin
| | | $t4 \leftarrow t4 \; + \; t2[m] \;\; \text{sign}(ykw[m][i][j])$
| | | $(\; \text{kron}(m = k) \; t1[i][j] - ykw[m][i][j] \;\; ykw[k][i][j] \;) \;\; / \;\; (t1[i][j])^{\frac{3}{2}}$
| | end
| | $b[k][i][j] \leftarrow ykw[k][i][j] \; / \; t1[i][j] + weights \; t4 \; / \; t3$
| end
end

** **Forward pass.** **

for all output units k do
begin
| for all input units j do
| begin
| | $ry[k][j] \leftarrow 0$
| end
| for $u = 2$ to g do
| begin
| | (IF $u \neq g$ THEN: for all units i in layer u ELSE: $i = k$) do
| | begin
| | | for all units j in layer $u - 1$ do
| | | begin
| | | | if ($alive[i][j]$ AND NOT $insignificant[i][j]$) do
| | | | begin
| | | | | $rs[k][i] \leftarrow rs[k][i] + w[i][j] \; ry[k][j] + b[k][i][j] \; y[j]$
| | | | end
| | | end
| | | $ry[k][i] \leftarrow rs[k][i] \; f_i'(s[i])$
| | end
| end
end

** **2. backward pass.** **

for all output units k do
begin
| $rdks[k] \leftarrow rs[k][k]\ f_k''(s[k])$
| for $u = 1$ to $g - 1$ do
| begin
| | for all units j in layer $g - u$ do
| | begin
| | | $rdky[j] \leftarrow 0$
| | | (IF $u \neq 1$ THEN: for all units i in layer $g - u + 1$ ELSE: $i = k$) do
| | | begin
| | | | if ($alive[i][j]$ AND NOT $insignificant[i][j]$) do
| | | | begin
| | | | | $rdky[j] \leftarrow rdky[j] + w[i][j]\ rdks[i] + b[k][i][j]\ ks[k][i]$
| | | | | $rdw[i][j] \leftarrow rdw[i][j] + \ y[j]\ rdks[i] + ry[k][j]\ ks[k][i]$
| | | | end
| | | end
| | | $rdks[j] \leftarrow f_j'(s[j])\ rdky[j] + rs[k][j]\ f_j''(s[j])\ ky[k][j]$
| | end
| end
end

** **Normalize B's gradient to the length of E's gradient.** **

$lyw \leftarrow 0$
$lrdw \leftarrow 0$

for all i, j, such that ($alive[i][j]$ AND NOT $insignificant[i][j]$) do
begin
| $lyw \leftarrow lyw + (yw[i][j])^2$
| $lrdw \leftarrow lrdw + (rdw[i][j])^2$
end

$scale = \sqrt{lyw}/\sqrt{lrdw}$ ** $scale\ = \parallel yw \parallel\ /\ \parallel rdw \parallel$ **

** **End of B's gradient computation (phase 2).** **

** **Weight update.** **

for all i, j, such that ($alive[i][j]$ AND NOT $insignificant[i][j]$) do
begin
| $w[i][j] \leftarrow w[i][j] + \alpha\ yw[i][j] - \lambda\ scale\ rdw[i][j]$
end

** **Update learning parameters.** **

$exemplars \leftarrow exemplars + 1$

```
if (exemplars mod epochlength = 0) do ** "mod" is the modulo function **
begin
|     epochs ← epochs + 1
|     E_n ← E_n / epochlength
|     E_a ← γ E_a + (1 − γ) E_n
|     ** lambda update according to Weigend et al.(1991). **
|     if (E_n ≤ E_tol OR E_n ≤ E_o) do
|     begin
|     |     λ ← λ + Δλ
|     end
|     else do
|     begin
|     |     if (E_n ≤ E_a) do
|     |     begin
|     |     |     λ ← λ − Δλ
|     |     |     if (λ < 0) do
|     |     |     begin
|     |     |     |     λ ← 0
|     |     |     end
|     |     end
|     |     else do
|     |     begin
|     |     |     λ ← σ λ
|     |     end
|     end
|     E_o ← E_n
|     E_n ← 0
|     ** update weights that are alive, **
|     ** a weight is alive if it was alive (marked by wasalive[][]) **
|     ** for at least one pattern presented during the previous epoch. **
|     for all i, j, such that (alive[i][j]) do
|     begin
|     |     if (wasalive[i][j] = FALSE) do
|     |     begin
|     |     |     alive[i][j] = FALSE
|     |     end
|     |     wasalive[i][j] = FALSE
|     end
|     if (epochs mod 100 = 0 OR E_a > 2.0 E_tol) do
|     ** weights are re-animated if the average error is too large; weights are also **
|     ** re-animated every 100-th epoch, to enable faster reduction of quadratic error **
|     ** (due to weight changes, some previously dead weight may turn out to deserve **
|     ** to live again); one may use values other than 100 and 2.0 **
|     begin
|     |     for all i, j such that w[i][j] exists do
|     |     begin
|     |     |     alive[i][j] = TRUE
|     |     end
|     end
end

decide whether to stop learning or not

end(while)
```

Acknowledgments

We would like to thank David Wolpert and anonymous referees for numerous comments that helped to improve previous drafts of this paper. This work was supported by *DFG grant SCHM 942/3-1* from "Deutsche Forschungsgemeinschaft".

References

Akaike, H. (1970). Statistical predictor identification. *Ann. Inst. Statist. Math.*, 22:203–217.

Amari, S. and Murata, N. (1993). Statistical theory of learning curves under entropic loss criterion. *Neural Computation*, 5(1):140–153.

Ash, T. (1989). Dynamic node creation in backpropagation neural networks. *Connection Science*, 1(4):365–375.

Bishop, C. M. (1993). Curvature-driven smoothing: A learning algorithm for feed-forward networks. *IEEE Transactions on Neural Networks*, 4(5):882–884.

Buntine, W. L. and Weigend, A. S. (1991). Bayesian back-propagation. *Complex Systems*, 5:603–643.

Carter, M. J., Rudolph, F. J., and Nucci, A. J. (1990). Operational fault tolerance of CMAC networks. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 340–347. San Mateo, CA: Morgan Kaufmann.

Craven, P. and Wahba, G. (1979). Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numer. Math.*, 31:377–403.

Eubank, R. L. (1988). Spline smoothing and nonparametric regression. In Farlow, S., editor, *Self-Organizing Methods in Modeling*. Marcel Dekker, New York.

Fahlman, S. E. and Lebiere, C. (1990). The cascade-correlation learning algorithm. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 525–532. San Mateo, CA: Morgan Kaufmann.

Golub, G., Heath, H., and Wahba, G. (1979). Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21:215–224.

Guyon, I., Vapnik, V., Boser, B., Bottou, L., and Solla, S. A. (1992). Structural risk minimization for character recognition. In Moody, J. E., Hanson, S. J., and Lippman, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 471–479. San Mateo, CA: Morgan Kaufmann.

Hanson, S. J. and Pratt, L. Y. (1989). Comparing biases for minimal network construction with back-propagation. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 1*, pages 177–185. San Mateo, CA: Morgan Kaufmann.

Hassibi, B. and Stork, D. G. (1993). Second order derivatives for network pruning: Optimal brain surgeon. In S. J. Hanson, J. D. C. and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 164–171. San Mateo, CA: Morgan Kaufmann.

Hastie, T. J. and Tibshirani, R. J. (1990). Generalized additive models. *Monographs on Statisics and Applied Probability*, 43.

Hinton, G. E. and van Camp, D. (1993). Keeping neural networks simple. In *Proceedings of the International Conference on Artificial Neural Networks, Amsterdam*, pages 11–18. Springer.

Hochreiter, S. and Schmidhuber, J. (1994). Flat minimum search finds simple nets. Technical Report FKI-200-94, Fakultät für Informatik, Technische Universität München.

Hochreiter, S. and Schmidhuber, J. (1995). Simplifying nets by discovering flat minima. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 529–536. MIT Press, Cambridge MA.

Holden, S. B. (1994). *On the Theory of Generalization and Self-Structuring in Linearly Weighted Connectionist Networks*. PhD thesis, Cambridge University, Engineering Department.

Kerlirzin, P. and Vallet, F. (1993). Robustness in multilayer perceptrons. *Neural Computation*, 5(1):473–482.

Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In Moody, J. E., Hanson, S. J., and Lippman, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 950–957. San Mateo, CA: Morgan Kaufmann.

Kullback, S. (1959). *Statistics and Information Theory*. J. Wiley and Sons, New York.

LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. San Mateo, CA: Morgan Kaufmann.

Levin, A. U., Leen, T. K., and Moody, J. E. (1994). Fast pruning using principal components. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 35–42. Morgan Kaufmann, San Mateo, CA.

Levin, E., Tishby, N., and Solla, S. (1990). A statistical approach to learning and generalization in layered neural networks. *Proceedings of the IEEE*, 78(10):1568–1574.

MacKay, D. J. C. (1992a). Bayesian interpolation. *Neural Computation*, 4:415–447.

MacKay, D. J. C. (1992b). A practical Bayesian framework for backprop networks. *Neural Computation*, 4:448–472.

Matsuoka, K. (1992). Noise injection into inputs in back-propagation learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):436–440.

Minai, A. A. and Williams, R. D. (1994). Perturbation response in feedforward networks. *Neural Networks*, 7(5):783–796.

Møller, M. F. (1993). Exact calculation of the product of the Hessian matrix of feed-forward network error functions and a vector in O(N) time. Technical Report PB-432, Computer Science Department, Aarhus University, Denmark.

Moody, J. E. (1989). Fast learning in multi-resolution hierarchies. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 1*. San Mateo, CA: Morgan Kaufmann.

Moody, J. E. (1992). The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In Moody, J. E., Hanson, S. J., and Lippman, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 847–854. San Mateo, CA: Morgan Kaufmann.

Moody, J. E. and Utans, J. (1994). Architecture selection strategies for neural networks: Application to corporate bond rating prediction. In Refenes, A. N., editor, *Neural Networks in the Capital Markets*. John Wiley & Sons.

Mosteller, F. and Tukey, J. W. (1968). Data analysis, including statistics. In Lindzey, G. and Aronson, E., editors, *Handbook of Social Psychology, Vol. 2*. Addison-Wesley.

Mozer, M. C. and Smolensky, P. (1989). Skeletonization: A technique for trimming the fat from a network via relevance assessment. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 1*, pages 107–115. San Mateo, CA: Morgan Kaufmann.

Murray, A. F. and Edwards, P. J. (1993). Synaptic weight noise during MLP learning enhances fault-tolerance, generalisation and learning trajectory. In S. J. Hanson, J. D. C. and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 491–498. San Mateo, CA: Morgan Kaufmann.

Neti, C., Schneider, M. H., and Young, E. D. (1992). Maximally fault tolerant neural networks. In *IEEE Transactions on Neural Networks*, volume 3, pages 14–23.

Nowlan, S. J. and Hinton, G. E. (1992). Simplifying neural networks by soft weight sharing. *Neural Computation*, 4:173–193.

Opper, M. and Haussler, D. (1991). Calculation of the learning curve of Bayes optimal classification algorithm for learning a perceptron with noise. In *Computational Learning Theory: Proceedings of the Forth Annual Workshop*. Morgan Kaufmann.

Pearlmutter, B. A. (1994). Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160.

Pearlmutter, B. A. and Rosenfeld, R. (1991). Chaitin-Kolmogorov complexity and generalization in neural networks. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 925–931. San Mateo, CA: Morgan Kaufmann.

Refenes, A. N., Francis, G., and Zapranis, A. D. (1994). Stock performance modeling using neural networks: A comparative study with regression models. *Neural Networks*.

Rehkugler, H. and Poddig, T. (1990). Statistische Methoden versus Künstliche Neuronale Netzwerke zur Aktienkursprognose. Technical Report 73, University Bamberg, Fakultät Sozial- und Wirtschaftswissenschaften.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465–471.

Schmidhuber, J. (1996). A general method for multi-agent learning and incremental self-improvement in unrestricted environments. In Yao, X., editor, *Evolutionary Computation: Theory and Applications*. Scientific Publ. Co., Singapore.

Schmidhuber, J. H. (1994a). Discovering problem solutions with low Kolmogorov complexity and high generalization capability. Technical Report FKI-194-94, Fakultät für Informatik, Technische Universität München. Short version in A. Prieditis and S. Russell, eds., *Machine Learning: Proceedings of the Twelfth International Conference*, pages 488-496, Morgan Kaufmann, San Francisco, CA, 1995.

Schmidhuber, J. H. (1994b). On learning how to learn learning strategies. Technical Report FKI-198-94, Fakultät für Informatik, Technische Universität München.

Shannon, C. E. (1948). A mathematical theory of communication (parts I and II). *Bell System Technical Journal*, XXVII:379–423.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Roy. Stat. Soc.*, 36:111–147.

Vapnik, V. (1992). Principles of risk minimization for learning theory. In Moody, J. E., Hanson, S. J., and Lippman, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 831–838. San Mateo, CA: Morgan Kaufmann.

Wallace, C. S. and Boulton, D. M. (1968). An information theoretic measure for classification. *Computer Journal*, 11(2):185–194.

Wang, C., Venkatesh, S. S., and Judd, J. S. (1994). Optimal stopping and effective machine complexity in learning. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 303–310. Morgan Kaufmann, San Mateo, CA.

Weigend, A. S., Rumelhart, D. E., and Huberman, B. A. (1991). Generalization by weight-elimination with application to forecasting. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 875–882. San Mateo, CA: Morgan Kaufmann.

White, H. (1989). Learning in artificial neural networks: A statistical perspective. *Neural Computation*, 1(4):425–464.

Williams, P. M. (1994). Bayesian regularisation and pruning using a Laplace prior. Technical report, School of Cognitive and Computing Sciences, University of Sussex, Falmer, Brighton.

Wolpert, D. H. (1994a). Bayesian backpropagation over i-o functions rather than weights. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 200–207. San Mateo, CA: Morgan Kaufmann.

Wolpert, D. H. (1994b). The relationship between PAC, the statistical physics framework, the Bayesian framework, and the VC framework. Technical Report SFI-TR-03-123, Santa Fe Institute, NM 87501.