FLAT MINIMUM SEARCH
FINDS SIMPLE NETS
Technical Report FKI-200-94

Sepp Hochreiter         Jürgen Schmidhuber
hochreit@informatik.tu-muenchen.de    schmidh@informatik.tu-muenchen.de

Fakultät für Informatik, H2
Technische Universität München
80290 München, Germany

December 31, 1994

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 argument shows that flat minima correspond to low expected over-fitting. 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".

1 INTRODUCTION
Our algorithm finds a large region in weight space with the property that each weight vector in that region leads to similar small error. Such a region is called a "flat minimum" [15]. To get an intuitive feeling for why a flat minimum is interesting, consider this [see also [50]): 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 length (MDL [45, 39]), 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.

Unlike e.g. Hinton and van Camp's method [14], our algorithm does not depend on the choice of a "good" 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 Pearlmutter's and Møller's efficient second order method [35, 26], 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, the method treats/prunes units and weights in different layers in different reasonable ways.

Outline. Section 2 introduces basic concepts. Section 3 describes the novel algorithm. Excellent experimental generalization results are reported in section 4. Section 5 mentions relations to previous work. A detailed theoretical justification of our approach is presented in the appendix.
2 TASK / ARCHITECTURE / BOXES

Generalization task. The task is to approximate an unknown relation $D \subset X \times Z$ between a set of possible inputs $X \subset \mathbb{R}^N$ and a set of possible outputs $Z \subset \mathbb{R}^K$. $D$ is taken to be a function. A relation $D$ is obtained from $\tilde{D}$ by adding noise to the outputs (see appendix A.1). All training information is given by a finite relation $D_0 \subset D$. $D_0$ is called the training set. The $p\text{th}$ element of $D_0$ is denoted by an input/target pair $(x_p, d_p)$.

Architecture. 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, $W$ weights, and differentiable activation functions. It maps input vectors $x_p \in \mathbb{R}^N$ to output vectors $o_p \in \mathbb{R}^K$. The weight from unit $j$ to $i$ is denoted by $w_{ij}$. The $W$-dimensional weight vector is denoted by $w$.

Training error. We use mean squared error $E_q(w, D_0) := \frac{1}{|D_0|} \sum_{(x_p, d_p) \in D_0} \| d_p - o_p \|^2$, where $\| \cdot \|$ denotes the Euclidian norm, and $|\cdot|$ denotes the cardinality of a set.

Tolerable error. To define a region in weight space with the property that each weight vector from that region has “similar small error”, we introduce the tolerable error $E_{tol}$, a positive constant. “Small” error is defined as being smaller than $E_{tol}$. $E_q(w, D_0) > E_{tol}$ implies “underfitting”.

Boxes. Each weight $w$ satisfying $E_q(w, D_0) \leq E_{tol}$ defines an “acceptable minimum”. We are interested in a large region of connected acceptable minima. Such a region is called a flat minimum. They are associated with low expected generalization error (see appendix A.2). 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 $W$-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 $\Delta w_{ij}$, which is the maximal (positive) value such that for all pairs $(i, j)$, all positive $k_{ij} \leq \Delta w_{ij}$ can be added to or subtracted from the corresponding $w_{ij}$ simultaneously without violating $E_q(w, D_0) \leq E_{tol}$ ($\Delta w_{ij}$ gives the precision of $w_{ij}$). $M_w$’s box volume is defined by $V(\Delta w) := 2^W \prod_{i,j} \Delta w_{ij}$, where $\Delta w$ denotes the vector with components $\Delta w_{ij}$.

3 THE ALGORITHM

Starting with a random initial weight vector, flat minimum search (FMS) finds a $w$ defining a box $M_w$ with maximal box volume $V(\Delta w)$ and minimal $\tilde{B}(w, D_0) := -\log(\frac{1}{|D_0|} V(\Delta w)) = \sum_{i,j} \log(\Delta w_{ij})$. Note the relationship to MDL ($\tilde{B}$ is the number of bits required to describe the weights, see appendix A.3).
In A.3, we derive the following algorithm. We use gradient descent to minimize \( E(w, D_h) = E_q(w, D_h) + \lambda B(w, D_h) \), where

\[
B(w, D_h) = \frac{1}{2} \left( -W \log \epsilon + \sum_{i,j} \log \left( \sum_k \left( \frac{1}{\sqrt{\sum_l (\frac{\partial o_k}{\partial w_{ij}})^2}} \right)^2 \right) \right).
\]

(1)

Here \( o_k \) is the activation of the \( k \)th output unit, \( \epsilon \) is a constant, and \( \lambda \) is a positive variable ensuring either \( E_q(w, D_h) \leq E_{q\ell} \) or an expected decrease of \( E_q(\cdot, D_h) \) during learning (see [47] for adjusting \( \lambda \)). To minimize \( B(w, D_h) \), we have to compute

\[
\frac{\partial B(w, D_h)}{\partial w_{uv}} = \sum_{k,i,j} \frac{\partial B(w, D_h)}{\partial (\frac{\partial o_k}{\partial w_{ij}})} \frac{\partial^2 o_k}{\partial w_{uv}} \frac{\partial w_{uv}}{\partial w_{ij}} \text{ for all } u, v.
\]

(2)

See details in appendix A.5.1. It can be shown that by using Pearlmutter’s and Møller’s efficient second-order method [35, 26], the gradient of \( B(w, D_h) \) can be computed in \( O(W) \) time (see details in A.5). Therefore, our algorithm has the same order of computational complexity as standard backprop.

4 EXPERIMENTAL RESULTS

4.1 EXPERIMENT 1 – noisy classification.

The first experiment is taken from Pearlmutter and Rosenfeld [36]. 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 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. The training set is based on 200 fixed data points (see figure 3). The test set is based on 120,000 data points.

![training set (200 points)](image)

Figure 3: The 200 input examples of the training set. Crosses represent data points from class 1. Squares represent data points from class 0.
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 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 4.6 for parameters common to all experiments.

### 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.

<table>
<thead>
<tr>
<th></th>
<th>Backprop</th>
<th>FMS</th>
<th></th>
<th>Backprop</th>
<th>FMS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSE</td>
<td>dto</td>
<td>MSE</td>
<td>dto</td>
<td>MSE</td>
</tr>
<tr>
<td>1</td>
<td>0.220</td>
<td>1.35</td>
<td>0.193</td>
<td>0.00</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>0.223</td>
<td>1.16</td>
<td>0.189</td>
<td>0.09</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>0.222</td>
<td>1.37</td>
<td>0.186</td>
<td>0.13</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0.213</td>
<td>1.18</td>
<td>0.181</td>
<td>0.01</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>0.222</td>
<td>1.24</td>
<td>0.195</td>
<td>0.25</td>
<td>10</td>
</tr>
</tbody>
</table>

4.2 EXPERIMENT 2 – recurrent nets.

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 4.6 for parameters common to all experiments.
4.3 EXPERIMENT 3 - stock market prediction (1).

We predict the DAX\(^1\) (the German stock market index) using fundamental indicators. Following Rehkugler and Poddig\([3,8]\), the net sees the following indicators: (a) German interest rate ("Umlaufrendite"), (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.

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\([1,2]\)), with a few improvements (see section 4.6), (4) Weight decay (WD) according to [47] (WD and OBS were chosen because they are well-known and widely used). Three different confidence measures are used: network output exceeding 0.0 (0.6, 0.9) is interpreted as a prediction of positive tendency. Network output below 0.0 (-0.6, -0.9) is interpreted as a prediction of negative tendency. Other outputs don't count.

Since wrong predictions lead to loss of money, performance is measured as follows. The sum of confidently but incorrectly predicted DAX changes is subtracted from the sum of confidently and correctly predicted DAX changes. The result is divided by the sum of absolute DAX changes. The most profitable network predicts everything confidently and correctly. With two networks with equal performance, the one making fewer confident predictions is preferable.

Results. See table 2. Our method outperforms the other methods. Only with confidence measure 0.9, BP4 and BP8 exhibit better performance per prediction, but considerably fewer predictions are made. Not much money can be made this way. Using FMS, the number of confident predictions does not change a lot if the confidence measure is changed. Thus, prediction quality does not depend strongly on the confidence measure to be chosen in advance.

Table 3 shows upper performance bounds for early stopping methods. They are obtained by cheating: test set performance is monitored during learning the training set. Column "b" shows the minimal number of millions of training examples needed to achieve small test set error, column "e" shows the corresponding maximal number (where test set error starts growing again). The corresponding interval \(I\) tends to be relatively small (only the best result within \(I\) is shown). Mean performance on \(I\) approximately equals the performance of WD and FMS based on confidence measures 0.0/0.6, and is below the performance of WD and FMS based on confidence measure 0.9. The probability of early stopping within \(I\), but without cheating, is small.

\(^1\)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.

<table>
<thead>
<tr>
<th>nr.</th>
<th>train</th>
<th>test</th>
<th>removed</th>
<th>performance-(\text{nr-pred.})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSE F1 F2 F3</td>
<td>MSE F1 F2 F3</td>
<td>w u</td>
<td>confidence: 0.0/0.6/0.9</td>
</tr>
<tr>
<td>BP8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.004 0 0 0</td>
<td>1.002 25 25 22</td>
<td></td>
<td>25.74/68 31.38/65 22.59/29</td>
</tr>
<tr>
<td>2</td>
<td>0.003 0 0 0</td>
<td>1.002 29 25 23</td>
<td></td>
<td>44.61/68 34.76/62 20.22/23</td>
</tr>
<tr>
<td>3</td>
<td>0.003 0 0 0</td>
<td>0.999 30 25 21</td>
<td></td>
<td>47.33/68 30.97/59 21.31/30</td>
</tr>
<tr>
<td>4</td>
<td>0.003 0 0 0</td>
<td>0.860 29 22 17</td>
<td></td>
<td>39.26/68 30.92/56 24.97/35</td>
</tr>
<tr>
<td>5</td>
<td>0.003 0 0 0</td>
<td>0.886 30 22 18</td>
<td></td>
<td>39.26/68 35.19/56 24.97/35</td>
</tr>
<tr>
<td>6</td>
<td>0.003 0 0 0</td>
<td>0.896 29 24 17</td>
<td></td>
<td>29.15/68 35.80/56 23.61/35</td>
</tr>
<tr>
<td>7</td>
<td>0.000 0 0 0</td>
<td>0.966 29 24 20</td>
<td></td>
<td>38.78/68 42.61/59 24.62/34</td>
</tr>
<tr>
<td>α</td>
<td>0.003 0 0 0</td>
<td>0.945 29 24 20</td>
<td></td>
<td>37.76/68 34.52/59 23.16/32</td>
</tr>
</tbody>
</table>

performance per prediction:
0.56/68 0.59/59 0.73/32
to be continued on next page
Table 2: 7 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. Column “MSE” shows mean squared error, “F1” (“F2”, “F3”) indicates number of errors greater than 0.2 (0.8, 1.4). “F2” indicates number of incorrect predictions (68 at most) using confidence measure 0.6. “F3” indicates number of incorrect confident predictions using confidence measure 0.6. The difference between “F1” and 68 is the corresponding number of correct confident predictions. Column “w” shows the number of pruned weights, column “u” shows the number of pruned units, the final 3 rows (“0.6”, “0.6”, “0.5”) list pairs of performance (see text) and number of confident predictions. \( \pi \) denotes the mean of 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).

With all confidence measures, our method outperforms all other methods.

<table>
<thead>
<tr>
<th>nr</th>
<th>train MSE</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>test MSE</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>removed w</th>
<th>u</th>
<th>performance-in-nr-pred. confidence: 0.6/0.6/0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.004</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.070</td>
<td>31</td>
<td>28</td>
<td>23</td>
<td>42.02-68</td>
<td>35.07-60</td>
<td>17.35-27</td>
</tr>
<tr>
<td>2</td>
<td>0.004</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.072</td>
<td>31</td>
<td>28</td>
<td>23</td>
<td>42.02-68</td>
<td>35.07-60</td>
<td>17.35-27</td>
</tr>
<tr>
<td>3</td>
<td>0.004</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.069</td>
<td>31</td>
<td>28</td>
<td>23</td>
<td>42.02-68</td>
<td>35.07-60</td>
<td>17.35-27</td>
</tr>
<tr>
<td>4</td>
<td>0.004</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.071</td>
<td>31</td>
<td>28</td>
<td>23</td>
<td>42.02-68</td>
<td>35.07-60</td>
<td>17.35-27</td>
</tr>
<tr>
<td>5</td>
<td>0.004</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.069</td>
<td>31</td>
<td>28</td>
<td>23</td>
<td>42.02-68</td>
<td>35.07-60</td>
<td>17.35-27</td>
</tr>
<tr>
<td>6</td>
<td>0.004</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.065</td>
<td>31</td>
<td>28</td>
<td>24</td>
<td>42.02-68</td>
<td>34.47-61</td>
<td>17.35-27</td>
</tr>
<tr>
<td>7</td>
<td>0.131</td>
<td>42</td>
<td>53</td>
<td>26</td>
<td>1.043</td>
<td>35</td>
<td>30</td>
<td>21</td>
<td>42.02-68</td>
<td>34.47-61</td>
<td>17.35-27</td>
</tr>
<tr>
<td>( \pi )</td>
<td>0.043</td>
<td>42</td>
<td>53</td>
<td>26</td>
<td>1.066</td>
<td>32</td>
<td>28</td>
<td>23</td>
<td>42.02-68</td>
<td>34.96-60</td>
<td>17.35-27</td>
</tr>
</tbody>
</table>

**Performance per prediction:**
- OBS: 0.62-68, 0.56-60, 0.64-62
- WD: 0.61-68, 0.58-61, 0.49-54
- FMS: 0.64-68, 0.62-64, 0.59-58

*Table continued from previous page*
Table 3: Upper performance bounds for arbitrary early stopping (ES) methods. They are obtained by cheating by looking at test set performance during learning. All nets start out with 8 hidden units. Column "MSE" shows mean squared error. "F1" ("F2"), "F3") indicates number of errors greater than 0.2 (0.8, 1.4). "F2" indicates number of incorrect predictions (68 at most) using confidence measure 0.6. "F3" indicates number of incorrect confident predictions using confidence measure 0.6. The difference between "F1" and 68 is the corresponding number of correct confident predictions. Column "b" shows the minimal number of millions of training examples needed to achieve small test set error, column "e" shows the corresponding maximal number (where test set error starts growing again). The final 3 rows ("0.0", "0.6", "0.9") list pairs of performance (see text) and number of confident predictions. \( \bar{x} \) denotes the mean of 7 trials.

<table>
<thead>
<tr>
<th>nr.</th>
<th>train</th>
<th>test</th>
<th>interval ( I )</th>
<th>performance-( \text{nr.-pred.} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSE</td>
<td>F1</td>
<td>F2</td>
<td>F3</td>
</tr>
<tr>
<td>ES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.004</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.006</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.007</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.004</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0.006</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0.005</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0.007</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.007</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

performance per prediction: 
\( 0.67-68 \) 
\( 0.72-62 \) 
\( 0.76-28 \)

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: \( E_{\text{tol}} = 0.13; \Delta \lambda = 0.001 \).
- WD: like with FMS, but \( w_0 = 0.2 \).
- OBS: \( E_{\text{tol}} = 0.015 \) (the same result was obtained with higher \( E_{\text{tol}} \) values, e.g. 0.13).

See section 4.6 for parameters common to all experiments.

4.4 EXPERIMENT 4 – stock market prediction (2).

We predict the DAX again, using the basic set-up of the experiment in section 4.3. 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).
- Network output exceeding 0.0 (0.6, 0.8) is interpreted as a prediction of positive tendency. Network output below 0.0 (-0.6, -0.8) is interpreted as a prediction of negative tendency. Other outputs don’t count.
- Performance of WD and FMS is also tested on networks “spoiled” by conventional backprop ("WDR" and "FMSR" – the “R” stands for Retraining).

Results are shown in table 4. Average performance of our method almost always exceeds the ones of weight decay, OBS, and conventional backprop. The only exception is weight decay with confidence measure 0.6 (but with the other comparisons, flat minimum search outperforms weight decay). Table 4 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.

Like table 3, table 5 shows upper performance bounds for early stopping methods. Again, they are obtained by cheating: test set performance is monitored during learning the training set. Columns “b” and “e” are analogous to those in table 3. Mean performance on interval I approximately equals the performance of WD and FMS based on confidence measures 0.0/0.8, and exceeds the performance of all investigated methods based on confidence measure 0.6. Recall that the probability of early stopping within I, but without cheating, is small.

<table>
<thead>
<tr>
<th>nr.</th>
<th>train MSE</th>
<th>test MSE</th>
<th>removed</th>
<th>performance–nr-pred. confidence: 0.0/0.6/0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F1 F2 F3</td>
<td>F1 F2 F3</td>
<td>w u</td>
<td></td>
</tr>
<tr>
<td>BP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.170 154</td>
<td>0.613 81</td>
<td>38.51</td>
<td>21.79-43 11.46-13</td>
</tr>
<tr>
<td>2</td>
<td>0.183 142</td>
<td>0.430 78</td>
<td>55.16</td>
<td>19.29-28 10.42-12</td>
</tr>
<tr>
<td>3</td>
<td>0.186 150</td>
<td>0.580 73</td>
<td>20.69</td>
<td>16.60-22 15.51-13</td>
</tr>
<tr>
<td>4</td>
<td>0.176 135</td>
<td>0.459 83</td>
<td>57.33</td>
<td>14.40-21 6.19-8</td>
</tr>
<tr>
<td>5</td>
<td>0.180 148</td>
<td>0.600 80</td>
<td>24.43</td>
<td>20.77-34 8.50-13</td>
</tr>
<tr>
<td>6</td>
<td>0.184 156</td>
<td>0.558 84</td>
<td>39.05</td>
<td>27.88-30 2.71-7</td>
</tr>
<tr>
<td>7</td>
<td>0.187 152</td>
<td>0.529 81</td>
<td>49.16</td>
<td>25.19-25 2.19-4</td>
</tr>
<tr>
<td>8</td>
<td>0.173 140</td>
<td>0.552 74</td>
<td>41.27</td>
<td>13.18-24 8.97-10</td>
</tr>
<tr>
<td>9</td>
<td>0.189 150</td>
<td>0.528 78</td>
<td>42.50</td>
<td>16.04-33 18.05-18</td>
</tr>
<tr>
<td>10</td>
<td>0.186 143</td>
<td>0.563 82</td>
<td>48.06</td>
<td>11.42-27 9.30-11</td>
</tr>
<tr>
<td>OBS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.217 158</td>
<td>0.515 80</td>
<td>38.09</td>
<td>19.89-38 9.91-16</td>
</tr>
<tr>
<td>2</td>
<td>0.213 164</td>
<td>0.491 79</td>
<td>40.88</td>
<td>22.72-25 10.72-11</td>
</tr>
<tr>
<td>3</td>
<td>0.245 170</td>
<td>0.463 81</td>
<td>50.78</td>
<td>13.93-18 16.68-8</td>
</tr>
<tr>
<td>4</td>
<td>0.215 162</td>
<td>0.525 76</td>
<td>32.20</td>
<td>21.61-27 13.82-14</td>
</tr>
<tr>
<td>5</td>
<td>0.212 165</td>
<td>0.498 83</td>
<td>34.50</td>
<td>17.18-25 10.84-7</td>
</tr>
<tr>
<td>6</td>
<td>0.235 162</td>
<td>0.503 79</td>
<td>40.55</td>
<td>14.31-25 8.23-8</td>
</tr>
<tr>
<td>7</td>
<td>0.217 154</td>
<td>0.525 84</td>
<td>43.60</td>
<td>10.99-26 4.55-8</td>
</tr>
<tr>
<td>8</td>
<td>0.213 162</td>
<td>0.497 84</td>
<td>36.58</td>
<td>17.83-24 4.57-6</td>
</tr>
<tr>
<td>9</td>
<td>0.213 160</td>
<td>0.498 81</td>
<td>40.36</td>
<td>21.52-28 8.52-7</td>
</tr>
<tr>
<td>10</td>
<td>0.212 167</td>
<td>0.500 83</td>
<td>45.72</td>
<td>20.39-26 8.23-10</td>
</tr>
<tr>
<td>WDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.169 155</td>
<td>0.626 80</td>
<td>34.52</td>
<td>21.90-44 16.44-15</td>
</tr>
<tr>
<td>2</td>
<td>0.181 145</td>
<td>0.446 77</td>
<td>55.10</td>
<td>19.19-28 10.56-12</td>
</tr>
<tr>
<td>3</td>
<td>0.185 150</td>
<td>0.584 73</td>
<td>24.58</td>
<td>15.45-23 14.33-12</td>
</tr>
<tr>
<td>4</td>
<td>0.175 137</td>
<td>0.451 82</td>
<td>62.54</td>
<td>17.14-23 6.33-9</td>
</tr>
<tr>
<td>5</td>
<td>0.179 146</td>
<td>0.612 80</td>
<td>23.70</td>
<td>20.33-34 9.48-11</td>
</tr>
<tr>
<td>6</td>
<td>0.183 156</td>
<td>0.544 82</td>
<td>36.16</td>
<td>28.25-32 5.95-7</td>
</tr>
<tr>
<td>7</td>
<td>0.186 152</td>
<td>0.524 83</td>
<td>50.37</td>
<td>25.42-25 2.19-4</td>
</tr>
<tr>
<td>8</td>
<td>0.171 138</td>
<td>0.560 76</td>
<td>39.28</td>
<td>13.64-23 8.97-10</td>
</tr>
</tbody>
</table>

(to be continued on next page)
Table 4: 10 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. Column “MSE” shows mean squared error. “F1” (“F2”, “F3”) indicates number of errors greater than 0.2 (0.8, 1.4). Column “w” shows the number of pruned weights, column “u” shows the number of pruned units. The final 3 rows (“0.0”, “0.6”, “0.8”) list pairs of performance (see text) and number of confident predictions. * denotes the mean of 10 trials. In one case (confidence measure 0.6), weight decay does slightly better than FMS. But in all other cases, flat minimum search outperforms all other methods, including WD.
Table 5: Upper performance bounds for arbitrary early stopping (ES) methods. They are obtained by cheating by looking at test set performance during learning. All nets start out with 8 hidden units. Column "MSE" shows mean squared error; "F1" ("F2", "F3") indicates number of errors greater than 0.2 (0.8, 1.4). Column "b" shows the minimal number of millions of training examples needed to achieve small test set error; column "e" shows the corresponding maximal number (where test set error starts growing again). The final 3 rows ("0.0", "0.6", "0.8") list pairs of performance (see text) and number of confident predictions.  denotes the mean of 10 trials.

### Parameters:
- Learning rate: 0.01.
- Architecture: (5-8-1).
- Number of training examples: 20,000,000.

Method specific parameters:
- **FMS**: $E_{\text{tol}} = 0.235$; $\Delta \lambda = 0.0001$; if $E_{\text{average}} < E_{\text{tol}}$ then $\Delta \lambda$ is set to 0.001.
- **WD**: like with FMS, but $w_0 = 0.2$.
- **FMSR**: like with FMS, but $E_{\text{tol}} = 0.15$; number of retraining examples: 5,000,000.
- **WDR**: like with FMSR, but $w_0 = 0.2$.
- **OBS**: $E_{\text{tol}} = 0.235$.

See section 4.6 for parameters common to all experiments.

### 4.5 EXPERIMENT 5 – stock market prediction (3).

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.

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).

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 $[-1, 1]$. 

<table>
<thead>
<tr>
<th>nr.</th>
<th>train</th>
<th>test</th>
<th>interval</th>
<th>performance-nr.-pred.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSE</td>
<td>F1</td>
<td>F2</td>
<td>F3</td>
</tr>
<tr>
<td>ES</td>
<td>MSE</td>
<td>F1</td>
<td>F2</td>
<td>F3</td>
</tr>
<tr>
<td>1</td>
<td>0.209</td>
<td>159</td>
<td>54</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>0.206</td>
<td>154</td>
<td>47</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>0.198</td>
<td>149</td>
<td>46</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>0.227</td>
<td>160</td>
<td>55</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>0.234</td>
<td>163</td>
<td>54</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>0.226</td>
<td>157</td>
<td>55</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>0.233</td>
<td>160</td>
<td>58</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>0.236</td>
<td>158</td>
<td>58</td>
<td>21</td>
</tr>
<tr>
<td>9</td>
<td>0.232</td>
<td>162</td>
<td>58</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>0.221</td>
<td>158</td>
<td>54</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>0.259</td>
<td>159</td>
<td>54</td>
<td>17</td>
</tr>
<tr>
<td>12</td>
<td>0.206</td>
<td>154</td>
<td>47</td>
<td>21</td>
</tr>
<tr>
<td>13</td>
<td>0.198</td>
<td>149</td>
<td>46</td>
<td>19</td>
</tr>
<tr>
<td>14</td>
<td>0.227</td>
<td>160</td>
<td>55</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>0.234</td>
<td>163</td>
<td>54</td>
<td>20</td>
</tr>
<tr>
<td>16</td>
<td>0.226</td>
<td>157</td>
<td>55</td>
<td>19</td>
</tr>
<tr>
<td>17</td>
<td>0.233</td>
<td>160</td>
<td>58</td>
<td>19</td>
</tr>
<tr>
<td>18</td>
<td>0.236</td>
<td>158</td>
<td>58</td>
<td>21</td>
</tr>
<tr>
<td>19</td>
<td>0.232</td>
<td>162</td>
<td>58</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>0.221</td>
<td>158</td>
<td>54</td>
<td>19</td>
</tr>
</tbody>
</table>
The following methods are applied to the training set: (1) Conventional backprop (BP), (2) weight decay (WD) according to [47], (3) flat minimum search (FMS). The resulting nets are evaluated on a test set consisting of 100 data points (August 1991 to July 1993).

Like in section 4.3, three different confidence measures are used: network output exceeding 0.0 (0.2, 0.4) is interpreted as a prediction of positive tendency. Network output below 0.0 (-0.2, -0.4) is interpreted as a prediction of negative tendency. Other outputs don’t count. Performance is measured like in section 4.3.

Results. Table 6 shows the results. Again, our method outperforms the other methods.

<table>
<thead>
<tr>
<th>train MSE</th>
<th>test MSE</th>
<th>rem. performance - nr. predicted confidence: 0.0/0.2/0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>w u</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BP</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.15</td>
<td>21 121 19 13</td>
</tr>
<tr>
<td>2</td>
<td>0.14</td>
<td>23 136 13 12</td>
</tr>
<tr>
<td>3</td>
<td>0.14</td>
<td>27 132 18 15</td>
</tr>
<tr>
<td>4</td>
<td>0.10</td>
<td>24 102 13 13</td>
</tr>
<tr>
<td>5</td>
<td>0.13</td>
<td>22 119 16 12</td>
</tr>
<tr>
<td>6</td>
<td>0.13</td>
<td>34 134 17 21</td>
</tr>
<tr>
<td>7</td>
<td>0.12</td>
<td>20 131 14 12</td>
</tr>
<tr>
<td>8</td>
<td>0.13</td>
<td>24 116 19 16</td>
</tr>
<tr>
<td>9</td>
<td>0.13</td>
<td>16 116 15 10</td>
</tr>
<tr>
<td>10</td>
<td>0.12</td>
<td>23 108 12 16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WDI</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.51</td>
<td>148 275 125 85</td>
</tr>
<tr>
<td>2</td>
<td>0.51</td>
<td>149 275 124 86</td>
</tr>
<tr>
<td>3</td>
<td>0.51</td>
<td>148 276 124 85</td>
</tr>
<tr>
<td>4</td>
<td>0.51</td>
<td>145 276 126 84</td>
</tr>
<tr>
<td>5</td>
<td>0.51</td>
<td>147 275 125 85</td>
</tr>
<tr>
<td>6</td>
<td>0.51</td>
<td>147 275 124 85</td>
</tr>
<tr>
<td>7</td>
<td>0.51</td>
<td>146 276 126 84</td>
</tr>
<tr>
<td>8</td>
<td>0.51</td>
<td>145 275 127 84</td>
</tr>
<tr>
<td>9</td>
<td>0.51</td>
<td>146 276 125 85</td>
</tr>
<tr>
<td>10</td>
<td>0.51</td>
<td>150 275 124 87</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WDI</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.34</td>
<td>84 215 60 46</td>
</tr>
<tr>
<td>2</td>
<td>0.34</td>
<td>106 230 61 63</td>
</tr>
<tr>
<td>3</td>
<td>0.34</td>
<td>75 239 57 43</td>
</tr>
<tr>
<td>4</td>
<td>0.34</td>
<td>89 231 61 36</td>
</tr>
<tr>
<td>5</td>
<td>0.34</td>
<td>86 232 50 47</td>
</tr>
<tr>
<td>6</td>
<td>0.34</td>
<td>89 244 66 50</td>
</tr>
<tr>
<td>7</td>
<td>0.34</td>
<td>84 230 60 46</td>
</tr>
<tr>
<td>8</td>
<td>0.34</td>
<td>86 242 62 59</td>
</tr>
<tr>
<td>9</td>
<td>0.34</td>
<td>90 234 51 47</td>
</tr>
<tr>
<td>10</td>
<td>0.34</td>
<td>93 234 68 53</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FMS</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.38</td>
<td>111 231 87 60</td>
</tr>
<tr>
<td>2</td>
<td>0.45</td>
<td>127 258 96 73</td>
</tr>
<tr>
<td>3</td>
<td>0.47</td>
<td>126 276 109 77</td>
</tr>
</tbody>
</table>

Again, our method outperforms the other methods. Performance per prediction: 0.690-0.0 0.154-55 0.153-25

To be continued on next page.
Table 6: 10 comparisons of conventional backprop (BP), optimal brain surgeon (OBS), weight decay (WD), flat minimum search (FMS). All nets start out with 9 hidden units. Column “MSE” shows mean squared error, “F1” indicates number of tendency errors, “F2” (“F3”) indicates number of errors greater than 0.2 (0.8), “F4” indicates number of tendency errors for changes from increasing DAX tendency to decreasing DAX tendency (or vice versa). Column “w” shows the number of pruned weights, column “u” shows the number of pruned units, the final 3 rows (“/0/0/” “/0/2/” “/0/4/”) list pairs of performance (see text) and number of confident predictions. \(\bar{x}\) denotes the mean of 10 trials. With all confidence measures, our method outperforms BP and WD.

**Parameters:**
- Learning rate: 0.01.
- Architecture: (12-9-1).
- Training time: 10,000,000 examples.
- Method specific parameters:
  - FMS1: \(E_{tol} \approx 0.34; \Delta \lambda = 0.003\).
  - FMS2: like with FMS1, but \(\Delta \lambda = 0.0005\). If \(E_{average} < E_{tol}\) then \(\Delta \lambda\) is set to 0.001.
  - WD1: like with FMS1, but \(w_g = 0.2\).
  - WD2: like with FMS2, but \(w_g = 0.2\).

See section 4.6 for parameters common to all experiments.
4.6 DETAILS / PARAMETERS

With exception of the experiment in section 4.2, 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 $w_0 = 0.2$. $E_{\text{average}}$ is the average error on the training set, approximated using exponential decay: $E_{\text{average}} = \gamma E_{\text{average}} + (1-\gamma)E_q(w, D_0)$, where $\gamma = 0.85$. All nets with the same number (see tables 1 - 6) start with the same weight initialization.

**FMS details.** To control $B(w, D_0)$'s influence during learning, its gradient is normalized and multiplied by the length of $E_q(w, D_0)$'s gradient (same for weight decay, see below). $\lambda$ is computed like in [47] 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 $\delta w_{ij}$ (see equation (11) in appendix A.3) exceeds the length of the weight range. However, the unknown scaling factor $\epsilon$ (see equations (9) and (11) in appendix A.3) is required to compute $\delta w_{ij}$. Therefore, we judge a weight $w_{ij}$ as being pruned if, with arbitrary $\epsilon$, $\delta w_{ij}$ is much bigger than the corresponding $\delta$'s of the other weights (typically, there are clearly separable classes of weights with high and low $\delta$'s). 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 $n_{\text{init}}$ time steps (typically, $n_{\text{init}} = 500,000$), 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 $0 > w_{ij} > -0.01$ are randomly initialized in $[-0.01, -0.005]$, and $\lambda$ is set to 0.

**Weight decay details.** We used the weight decay term in [47]: $D(w, w_0) = \sum_{i,j} \frac{w_{ij}^2}{H_{ij}^{\frac{1}{2}}}$.

Like with FMS, $D(w, w_0)$'s gradient was normalized and multiplied by the length of $E_q(w, D_0)$'s gradient. $\lambda$ 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 $\delta w$ 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 [12]):

- To detect the weight that deserves deletion, we use both $L_q = \frac{w_i^2}{w_{ij}^2}$ (the original value used by Hassibi et al.) and $T_q := -\frac{\partial^2 F}{\partial w_{ij}^2} w_{ij} + \frac{\partial F}{\partial w_{ij}} w_i^2$. 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 [20], to prevent numerical errors due to small eigenvalues of $H$, we do: if $L_q < 0.0001$ or $T_q < 0.0001$ 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 elements of a matrix.

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

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

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

Most previous algorithms for finding low complexity networks with high generalization capability are based on more prior assumptions than our approach (see appendix A.2). They can be broadly classified into two categories (see [40], however, for an exception):

(1) **Assumptions about the prior weight distribution.** Hinton and van Camp [14] and Williams [49] assume that pushing the posterior distribution (after learning) close to the prior leads to “good” generalization (see more details below). Weight decay (e.g., [11, 18]) can be derived e.g. from Gaussian or Laplace priors. Nowlan and Hinton [34] assume that networks with many similar weights generated by Gaussian mixtures are “better” a priori. MacKay’s priors [23] are implicit in additional penalty terms, which embody the assumptions made.

(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 [30, 43, 7, 13], e.g. “generalized cross validation” [6, 9], “final prediction error” [1], “generalized prediction error” [29, 28]. See also Holden [16], Wang et al. [46], Amari and Murata [2], and Vapnik’s “structural risk minimization” [10, 44].

**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” [8, 3, 27], input pruning [28, 37], unit pruning [48, 31, 24], weight pruning, e.g. “optimal brain damage” (OBD [20], “optimal brain surgeon” (OBS [12]).

**Hinton and van Camp** [14]. They minimize the sum of two terms: the first is conventional error plus variance, the other is the distance $\int p(\alpha | D_0) \log \frac{p(\alpha | D_0)}{p(\alpha)} d\alpha$ between posterior $p(\alpha | D_0)$ and prior $p(\alpha)$. The problem is to choose a “good” prior. In contrast to their approach, our approach does not require a “good” prior given in advance. 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 $\Delta w_{ij}$. Our approach, however, *does* justify linear approximation, as seen in appendix A.4.

**Wolpert** [50]. 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 a 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 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** [32]. 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).
APPENDIX – THEORETICAL JUSTIFICATION

Contents:
A.1 The probability distributions
A.2 Overfitting error
  A.2.1 “Closeness” of \( D_0 \) to \( D \)
  A.2.2 “Flatness” of \( D \)
  A.2.3 Approximation and minimum of the overfitting error
  A.2.4 Why does search for big \( D_0 \)-boxes work?
A.3 How to flatten the network output
A.4 Why does the Hessian decrease?
A.5 Efficient implementation of the algorithm
  A.5.1 Explicit derivative of equation (1)
  A.5.2 Fast multiplication by the Hessian

A.1. THE PROBABILITY DISTRIBUTIONS

Outline. To measure overfitting error (in section A.2), we need probability distributions on data and weights. The targets are partly due to noise (see section 2). The probability of matching the original, noise-free target component of an element in \( \hat{D} \) (see section 2) depends on the weights and the input. Using Bayes’ rule, we obtain an a posteriori distribution of the weights.

**Formal details.** In what follows, \( \alpha \) denotes a weight vector, and \( \hat{D} = \{(x_p, d_p) \mid p \geq 1\} \subseteq D \) (in section A.2 we will consider the special cases \( \hat{D} = D \) and \( \hat{D} = D_0 \)). In analogy to section 2, we define \( E_q(\alpha, \hat{D}) := \frac{1}{|\hat{D}|} \sum_{(x_p, d_p) \in \hat{D}} \| d_p - \alpha \cdot x_p \|^2 \) for \( \hat{D} \) with finite cardinality. For \( \hat{D} \) with infinite cardinality, we define \( E_q(\alpha, \hat{D}) := \lim_{n \to \infty} E_q(\alpha, D_n) \), if this limes is finite and equal for all \( \{D_n \mid n \geq 1\} \) with \( \lim_{n \to \infty} D_n = \hat{D} \), where \( |D_n| = n \) and \( D_n \subset \hat{D} \). Otherwise \( E_q(v, \hat{D}) \) is undefined.

Let \( p(\hat{D} \mid \alpha) \) denote the probability distribution according to which the network makes predictions\(^2\) of \( \hat{D} \)’s output values in the input range of \( \hat{D} \), given weights \( \alpha \). We assume Gaussian noise\(^3\) with deviation \( \sigma \) on the noise-free targets. With \( \beta := \frac{\sigma}{\|x\|} \), we obtain (see [23, 52])

\[
p(\hat{D} \mid \alpha) = \frac{\exp(-\beta E_q(\alpha, \hat{D}))}{Z(\beta)}.
\]

Here \( Z(\beta) \) is a normalizing constant, \( Z(\beta) = \int_\mathbb{R} \exp(-\beta E_q(\alpha, \hat{D}))d\hat{D} \), where \( \mathbb{R} \) is the \( K \)-dimensional vector space of possible data \( \hat{D} \in \mathbb{R} \). \( Z(\beta) \) does not depend on \( \alpha \), due to translation invariance of the error in \( \mathbb{R} \). \( Z(\beta) = \left( \frac{2\pi \beta}{K} \right)^{\frac{K}{2}} \) holds.\(^4\)

Using e.g. gradient descent algorithms, the prior does not coincide with the distribution used for weight initialization. The “true” prior \( p_A(\alpha) \) depends on the nature of the learning algorithm A. \( p_A(\alpha) \) gives the probability of \( \alpha \) after training with \( A \).\(^5\) From practical experience we infer that weight initialization does not heavily influence the posterior. From Bayes’ rule we obtain the posterior weight distribution, given the observations \( \hat{D} \):

\(^2\) Following [23, 51], \( p(\hat{D} \mid \alpha) \) is just a shorthand expression for \( p(\{d_p \mid (x_p, \alpha)\} \mid \alpha) \).

\(^3\) Alternatively, instead of assuming Gaussian noise, equation (3) can be obtained by following Levin, Tishby and Solla [22]. They derive it as the only solution of a functional equation, where \( \beta \) is a constant expressing the sensitivity of \( p(\hat{D} \mid \alpha) \) on the error.

\(^4\) It may be useful to define \( E(\beta) := \int E_q(\alpha, \hat{D}) p(\hat{D} \mid \alpha) d\hat{D} = \frac{1}{|\hat{D}|} \int E_q(\alpha, \hat{D}) \exp(-\beta E_q(\alpha, \hat{D})) d\hat{D} = \left( \frac{2\pi \beta}{K} \right)^{\frac{K}{2}}. \)

\( E(\beta) \) is a function of the derivative of \( Z(\beta) : \log E(\beta) = K \log \left( \frac{2\pi \beta}{K} \right)^{\frac{K}{2}} = -K \log(2\beta). \) \( E(\beta) \) is the mean squared error over all possible data when using a fixed \( \beta \), thus being a measure of the “acceptable error level”. See Levin et al. [22]. Why should \( E(\beta) \) be of interest? With an estimate of \( \beta \) we are able to estimate \( E_{\text{reg}} \) as well.

\(^5\) For instance, MacKay [23] obtains \( p_A(\alpha) \) equal to a multidimensional Gaussian distribution with mean 0, by adding \( \lambda \| \alpha \|^2 \) to his error function. Similar statements can be made about Hinton and van Camp’s approach [14].
\[ p_A(\alpha | \tilde{D}) = \frac{p(\tilde{D} | \alpha)p_A(\alpha)}{p_A(\tilde{D})}. \]  

\( p_A(\tilde{D}) := \int p(\tilde{D} | \alpha)p_A(\alpha) d\alpha \)

tells us about learning algorithm A’s usefulness for predicting \( \tilde{D} \)’s output values in \( \tilde{D} \)’s input range. \( p_A(\alpha) = \int p_A(\alpha | \tilde{D})p_A(\tilde{D})d\tilde{D} \) holds. With problem class \( P \) and \( \tilde{D} \in P \), \( p_A(\tilde{D}) = p_P(\tilde{D}) \) defines properties of an “optimal” learning algorithm A with respect to \( P \), where \( p_P(D) \) is the probability of choosing \( D \) from \( P \).

With uniform \( p_P(.) \), no algorithm can outperform error reducing algorithms (see [51]). Typical problem distributions in the real world are not uniform, however – the real world appears to favor solutions with low algorithmic complexity. See e.g. [40]. See [41] for a universal “self-improving” system which continually attempts to incrementally modify its prior based on experience with previous problems.

### A.2. OVERFITTING ERROR

**Outline.** In analogy to [46] and [10], we decompose the generalization error into an “overfitting” error and an “underfitting” error. There is no significant underfitting error (corresponding to Vapnik’s empirical risk) if \( E_\alpha(w, D_0) \leq E_{\text{tol}} \). Some thought is required, however, to define the “overfitting” error. We do this in a novel way.

**Intuition.** Since we do not know the relation \( D \), we cannot know \( p(\alpha | D) \), the “optimal” posterior weight distribution we would obtain by training the net on \( D \) (”sure thing hypothesis”). But, for theoretical purposes, suppose we did know \( p(\alpha | D) \). Then we could use \( p(\alpha | D) \) to initialize weights before learning the training set \( D_0 \). Using the Kullback-Leibler distance [19], we measure the information (due to noise) conveyed by \( D_0 \), but not by \( D \) (see figure 4). In conjunction with the initialization above, this provides the conceptual setting for defining an overfitting error measure. But, the initialization does not really matter, because it does not heavily influence the posterior (see section A.1).

![Figure 4: Positive contributions to the overfitting error](image1.png)

![Figure 5: Positive contributions to the underfitting error](image2.png)
Formal details. We assume a conventional gradient based learning algorithm $A$ adjusting the weights to locally minimize the error. Then $p_A(a)$ is uniform and $p(\alpha \mid D) := p_A(a \mid D) = p(D \mid a)$. See equation (4). The overfitting error is the Kullback-Leibler distance of the posteriors:

$$E_o(D, D_b) = \int p(\alpha \mid D_b) \log \frac{p(\alpha \mid D_b)}{p(\alpha \mid D)} d\alpha = \int p(\alpha \mid D_b) \log p(D_b \mid \alpha) d\alpha - \int p(\alpha \mid D_b) \log p(D \mid \alpha) d\alpha = \beta \left( \int p(\alpha \mid D_b) E_q(\alpha, D) d\alpha - \bar{E}_q(D_b) \right),$$

where $\bar{E}_q(D_b) := \int p(\alpha \mid D_b) E_q(\alpha, D_b) d\alpha$ is the mean error after learning $D_b$, and $E_o(D, D_b)$ is the expectation of $\log \frac{p(\alpha \mid D_b)}{p(\alpha \mid D)} = -\log p(\alpha \mid D) - (-\log p(\alpha \mid D_b))$ (the expected difference of the minimal description [42] of $\alpha$ with respect to $D$ and $D_b$, after learning $D_b$).

Now we measure the expected overfitting error relative to $M_w$ (see section 2) by computing the expectation of $\log \frac{p(\alpha \mid D)}{p(\alpha \mid D_b)}$ in the range $M_w$:

$$E_{ro}(w) = E_{ro}(D, D_b, M_w) = \beta \left( \int_{M_w} p_{M_w}(\alpha \mid D_b) E_q(\alpha, D) d\alpha - \bar{E}_q(D_b, M_w) \right).$$

Here $p_{M_w}(\alpha \mid D_b) := \frac{p(\alpha \mid D_b)}{\int_{M_w} p(\alpha \mid D_b) d\alpha}$ is the posterior of $D_b$ scaled to obtain a distribution within $M_w$, and $\bar{E}_q(D_b, M_w) := \int_{M_w} p_{M_w}(\alpha \mid D_b) E_q(\alpha, D_b) d\alpha$ is the mean error in $M_w$ with respect to $D_b$.

Clearly, we would like to pick $w$ such that $E_{ro}(w)$ is minimized. How to do that? Actually, we will minimize an approximation of $E_{ro}(w)$ (sections 2.3, A.2.4). Towards this purpose, we need to make two additional prior assumptions, which are actually implicit in most previous approaches (which make additional stronger assumptions, see section 5): (1) “Closeness assumption”: every minimum of $E_q(\cdot, D_b)$ is “close” to a maximum of $p(\alpha \mid D)$ (see formal definition below). Intuitively, “closeness” ensures that $D_b$ can indeed tell us something about $D$, such that training on $D_b$ may indeed reduce the error on $D$. (2) “Flatness assumption”: the peaks of $p(\alpha \mid D)$’s maxima are not sharp. This MDL-like assumption holds if not all weights have to be known exactly to model $D$. It ensures that there are regions with low error on $D$. Let us have a closer look at both assumptions.

A.2.1. “Closeness” of $D_b$ to $D$

Intuition. In analogy to section 2, equation (5), we measure the underfitting error – the information conveyed by $D$, but not by $D_b$ (note the exchange of $D$ and $D_b$ – see figure 5). This information is determined by the $D_b$-error at the most likelihood weight of $D$. The concept of underfitting error allows for defining “closeness”. Closeness implies that the most likelihood weight of $D$ and the most likelihood weight of $D_b$ are “close” together in weight space.

Formal details. The underfitting error is

$$E_u(D_b, D) = \int p(\alpha \mid D) \log \frac{p(\alpha \mid D)}{p(\alpha \mid D_b)} d\alpha = \int p(\alpha \mid D) \log p(D \mid \alpha) d\alpha - \int p(\alpha \mid D) \log p(D_b \mid \alpha) d\alpha = \beta \left( \int p(\alpha \mid D) E_q(\alpha, D_b) d\alpha - \bar{E}_q(D) \right).$$

Here the errors are defined in analogy to the corresponding definitions for the overfitting error, see equation (5). Let $\tilde{w}$ be a most likelihood weight with respect to $D$. $p(\alpha \mid D)$ has its peak at (or near) $\tilde{w}$. Thus, the underfitting error is dominated by $E_q(\tilde{w}, D_b)$. Let $w$ be a most likelihood weight with respect to $D$, and let $\tilde{U}_{E_{ro}}(\tilde{w})$ be a connected region around $\tilde{w}$, such that $\tilde{\alpha} \in \tilde{U}_{E_{ro}}(\tilde{w})$ implies $E_q(\alpha, D_b) < E_{ro}$.

Definition: $\tilde{w}$ is near $\tilde{w}$ with respect to $E_{ro}$, iff $\tilde{w} \in \tilde{U}_{E_{ro}}(\tilde{w})$. $D_b$ is close to $D$, iff for all most likelihood weights $\tilde{w}$ of $D_b$, there exists a most likelihood weight $\tilde{w}$ of $D$ such that $\tilde{w}$ is near $\tilde{w}$. 

17
A.2.2. “FLATNESS” OF $D$

**Intuition.** $D$ has a “flat” error surface or a “flat” posterior if (1): at least one weight is not needed to generate $D$, or (2): there exists a big box with respect to $D$, such that all weight vectors within the box yield about the same error as the most likelihood weight vector of $D$ (which yields minimal error). Hence, there exists a “good” weight vector which can be described with low precision (and few bits).

**Formal details.** Let $U_{E_q(\bar{w}, D) + \epsilon}(\hat{w})$ be a connected region around $\hat{w}$, where $\alpha \in U_{E_q(\bar{w}, D) + \epsilon}(\hat{w})$ implies $E_q(\alpha, D) < E_q(\hat{w}, D) + \rho$, where $\rho$ is a positive constant near zero.

**Definition.** $D$ is flat, if (1) there exists a $\hat{w} \in U_{E_q(\bar{w}, D) + \epsilon}(\hat{w})$, such that there exists at least one $w_{ij}$, such that for all $c$ with $|c| < \epsilon$ below maximal weight value the following holds: $\hat{w} + c \epsilon_{ij} \in U_{E_q(\bar{w}, D) + \epsilon}(\hat{w})$, where $\epsilon_{ij}$ is the unit vector in the direction corresponding to $i, j$.

Or, $D$ is flat, if (2) $V(\Delta w) \gg 0$ (i.e. $\hat{w} = \bar{w}$) or, otherwise, there exists $\bar{w} \in U_{E_q(\bar{w}, D) + \epsilon}(\hat{w})$ such that $V(\Delta w) \gg V(\Delta \bar{w})$. Here $V(\Delta w) := \max \{2^B \prod_{ij} \tau_{ij} \mid \forall \kappa_{uv} : 0 \leq \kappa_{uv} \leq \tau_{uv} \leq \text{maximal weight value} : w + \sum_{uv} \pm \kappa_{uv} c_{uv} \in U_{E_q(\bar{w}, D) + \epsilon}(\hat{w}) \}$. $V(\Delta w)$ is the volume of the box $G_w$. Note the similarity to the definition of $M_w$ and $V(\Delta w)$ in section 2: within $U_{E_q(\bar{w}, D)}(w)$. $M_w$ is the biggest box with center $w$, given $D_w$. Within $U_{E_q(\bar{w}, D) + \epsilon}(\hat{w})$, $G_w$ is the biggest box with center $w$, given $D$. Note: (1) implies (2). Therefore, in what follows we focus on (2).

A.2.3. APPROXIMATION AND MINIMUM OF $E_{\tau_{\rho}}$

$M_w$ is defined to have maximal box volume within $U_{E_{\tau_{\rho}}}(w)$. Within the box, the error $E_q(\alpha, D)$ is close to $E_{\tau_{\rho}}$. Having found a very flat box (see flatness condition 2 in A.3), we may assume that $p_{M_w}(\alpha \mid D_h) \approx \frac{1}{V(\Delta w)}$, i.e. $p_{M_w}$ is uniform. Now we can approximate $E_{\tau_{\rho}}$:

$$E_{\tau_{\rho}}(w) \approx \beta \left( \frac{\int_{M_w} E_q(\alpha, D) \, d\alpha}{V(\Delta w)} - E_{\tau_{\rho}} \right).$$

$\beta$, the variance of the target noise, and the tolerable error are fixed parameters of the relative overfitting error. This error is determined by $\int_{M_w} E_q(\alpha, D) \, d\alpha \approx \text{mean}(E_q(\alpha, D), M_w)$, which is approximately the mean value of $E_q(\alpha, D)$ within a flat box. Due to flatness, there exists at least one $\bar{w}$ such that $\text{mean}(E_q(\alpha, D), G_{\bar{w}}) \approx E_q(\bar{w}, D)$. Due to $\bar{w}$ being the most likelihood weight of $D$, $\text{mean}(E_q(\alpha, D), G_{\bar{w}}) \geq E_q(\hat{w}, D)$ for every weight $w$.

We would like to find some $\hat{w}$ such that the overfitting error is (nearly) minimal both at $\hat{w}$ and on $G_{\bar{w}}$. Near $\hat{w}$, the vector $\bar{w}$ contains the biggest box $G_{\bar{w}}$, given $D$. Since only $D_h$ is available, direct search for $G_{\bar{w}}$ is not possible -- all we can do is to look for a big $M_w$. The problem is: not every $w$ defining a big $M_w$ defines a big $G_w$ as well. To see this, suppose $D$ contains a subset containing no elements of $D_h$. The next section shows how to deal with this problem.

A.2.4. WHY DOES SEARCH FOR BIG $D_h$-BOXES WORK?

**Outline.** If $E_{\tau_{\rho}}$ is small enough (this implies “enough” closeness), we do have a chance to find a $\bar{w}$ defining a big $G_{\bar{w}}$. Since flatness implies the existence of a big $D_h$-box, this subsection justifies our search for big $D_h$-boxes by showing that a big $D_h$-box implies a big $D_h$-box. Is it possible to end up with a “wrong” big $D_h$-box (one that is not implied by a big $D_h$-box)? Not if there is enough “closeness”, as will be seen below.

**Formal details.** Let’s suppose there exists a $\bar{w} \in U_{E_{\tau_{\rho}}}(w)$, such that $V(\Delta \bar{w}) \gg V(\Delta \bar{w})$ for all $\bar{w}$ (see A.2.2. on flatness), but $\text{mean}(E_q(\alpha, D), M_w) \gg E_q(\hat{w}, D)$. Then $\hat{w}$ cannot define a big $G_{\bar{w}}$ (we define a “wrong” $D_h$-box). Now assume that the inner product of $\bar{w}, V(\Delta \bar{w}) \parallel \bar{w}$ and the vector pointing to the center of the biggest $D_h$-box close to $w$ is always positive. Then we cannot end up with a $\hat{w}$ as above. Due to flatness, network outputs are about equal on $G_{\bar{w}}$. From this we infer (*): $\text{mean}(E_q(\alpha, D_h), M_w) \approx E_q(\hat{w}, D_h) \approx E_q(\hat{w}, D_h)$. Therefore, in what follows we focus on (2).

The validity of our “enough closeness assumption” is equivalent to the validity of the assumption that $E_{\tau_{\rho}}$ goes towards $E_q(\hat{w}, D_h)$. The latter implies $E_q(\hat{w}, D_h) \rightarrow E_q(\hat{w}, D_h)$ (due to
closeness relative to $E_{tol}$. Now let us assume $E_q(\bar{w}, D_h) > E_q(\bar{w}, D_h)$ (this holds if $|D_h| > W$). This implies that $V(\Delta \bar{w})$ decreases if $E_{tol}$ decreases. From (*) (see last paragraph) we deduce that $V(\Delta \bar{w})$ does not decrease. Hence, $V(\Delta \bar{w}) < V(\Delta \bar{w})$ for at least one $\bar{w}$. Alas, we can find a $\bar{w}$ by a gradient based method for searching big box volumes, provided there is enough closeness ($E_{tol}$ is small enough). Now we see: maximizing $V(\Delta \bar{w})$ (thus minimizing $E_{rea}$) is equivalent to a search for acceptable minima with “flat” network output (the error depends on the output only). The next subsection shows how to find networks with flat output.

A.3. HOW TO FLATTEN THE NETWORK OUTPUT

Outline. To find nets with flat outputs, two conditions will be defined to specify $B(w, D_h)$ (see section 3). The first condition ensures flatness. The second condition enforces “equal flatness” in all weight space directions, and can be justified using an MDL-based argument. In both cases, linear approximations will be made (to be justified in A.4).

Formal details. We are looking for weights (causing tolerable error) that can be perturbed without causing significant output changes. Perturbing the weights $w$ by $\delta w$ (with components $\delta w_{ij}$), we obtain $ED(w, \delta w) := \sum_k (\delta^2 w_{ij} + \delta^3 w_{ij})^2$, where $\delta^2 w_{ij}$ expresses $w_{ij}$’s dependence on $w$ (in what follows, however, $w_{ij}$ is often modified for convenience). Linear approximation (justified in A.4) gives us “Flatness Condition 1”:

$$ED(w, \delta w) \approx ED(\delta w) := \sum_k \left( \sum_{ij} \frac{\partial^2 w}{\partial w_{ij}} \delta w_{ij} \right)^2 \leq ED_{max}(\delta w) := \sum_k \left( \sum_{ij} \left( \frac{\partial^3 w}{\partial w_{ij}} \right)^2 \delta w_{ij} \right) \leq \epsilon, \quad (9)$$

where $\epsilon > 0$ defines tolerable output changes within a box and is small enough to allow for linear approximation (it does not appear in $B(w, D_h)$’s gradient, see section 3). $ED$ is the linear approximation of $ED$, and $ED_{max}$ is $\max \{ ED(w, \delta w) \}$. Flatness condition 1 is a “robustness condition” (or “fault tolerance condition”, or “perturbation tolerance condition” – see e.g. [25, 32, 33, 24, 4, 17, 5]).

The validity of approximation (8) depends on how flat box $M_w$ is. Many $M_w$ satisfy flatness condition 1. To select a particular, very flat $M_w$, the following “Flatness Condition 2” uses up degrees of freedom left by equation (9):

$$\forall i, j, u, v: (\delta w_{ij})^2 \sum_k \left( \frac{\partial^2 w}{\partial w_{ij}} \right)^2 \leq (\delta w_{uv})^2 \sum_k \left( \frac{\partial^3 w}{\partial w_{ij}} \right)^2. \quad (10)$$

Flatness condition 2 enforces equal “directed errors” $ED_{ij}(w, \delta w_{ij}) = \sum_k \left( \delta^2 w_{ij} + \delta^3 w_{ij} \right) = \sum_k \left( \frac{\partial^3 w}{\partial w_{ij}} \right)^2 \delta w_{ij}$, where $\delta^2 w_{ij}$ has the obvious meaning. Linear approximation is justified by the choice of $\epsilon$ in equation (9).

How to derive the algorithm from flatness conditions 1 and 2. We first solve equation (10) for $|\delta w_{ij}| = |\delta w_{uv}| \frac{\sum_k \left( \frac{\partial^2 w}{\partial w_{ij}} \right)}{\sum_k \left( \frac{\partial^3 w}{\partial w_{ij}} \right)}$ (fixing $u, v$ for all $i, j$). Then we insert $|\delta w_{ij}|$ into equation (9) (replacing the second “$\leq$” in (9) by “$=$”). This gives us an equation for the $|\delta w_{ij}|$ (which depend on $w$, but this is notationally suppressed):

$$|\delta w_{ij}| = \frac{\sqrt{\sum_k \left( \frac{\partial^2 w}{\partial w_{ij}} \right)^2}}{\sqrt{\sum_k \left( \frac{\partial^3 w}{\partial w_{ij}} \right)^2}} \frac{\sum_k \left( \frac{\partial^2 w}{\partial w_{ij}} \right)}{\sum_k \left( \frac{\partial^3 w}{\partial w_{ij}} \right)^2}. \quad (11)$$

The $|\delta w_{ij}|$ approximate the $\Delta w_{ij}$ from section 2. The box $M_w$ is approximated by $AM_w$, the box with center $w$ and edge length $2|\delta w_{ij}|$. $M_w$’s volume $V(\Delta w)$ is approximated by $AM_w$’s box volume $V(\Delta w) := 2^W \prod_{ij} |\delta w_{ij}|$. Thus, $\tilde{H}(w, D_h)$ (see section 3) can be approximated by
Flatness condition 2 is essential: flatness condition 1 by itself corresponds to nothing more but first-order derivative reduction (ordinary sensitivity reduction).

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

MDL-based justification of flatness condition 2. Let us assume a sender wants to send a description of some box center \( w \) to a receiver who knows the inputs but not the targets. The MDL principle suggests that the sender wants to minimize \( w \)'s expected description length. Let \( ED_{\text{mean}}(w, D_h) \) denote the mean value of \( ED \) on the box. Expected description length is approximated by \( ED_{\text{mean}}(w, D_h) + B(w, D_h) \). One way of seeing this is to apply Hinton and van Camp's "bits back" argument [14] to a uniform prior (\( ED_{\text{mean}} \) corresponds to Hinton and van Camp's 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 (\( \Delta w_{ij} \) is given):

1. Define a variable interval \( I_{ij} \subset R \).
2. Make \( I_{ij} \) equal to the interval constraining possible weight values.
3. Divide \( l_{ij} \) into 2 equally-sized disjunct intervals \( l_1 \) and \( l_2 \).
   - If \( w_{ij} \in l_1 \) then \( I_{ij} \leftarrow l_1 \); write '1'.
   - If \( w_{ij} \notin l_1 \) then \( I_{ij} \leftarrow l_2 \); 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 \( B(w, D_h) + \epsilon \), where the constant \( \epsilon \) is independent of the box \( M_w \). From \( ED(w, w_1 - w) \) (\( w_1 \) is the center of the "bit-box") and the bitstring describing the "bit-box", the receiver can compute \( w \) as follows: he chooses an initialization weight vector within the "bit-box" and uses gradient descent to decrease \( B(w_1, D_h) \) until \( ED(w_1, w_1 - w) = ED(w, w_1 - w) \), where \( w_1 \) in the bit-box denotes the receiver's current approximation of \( w \) (\( w_1 \) is constantly updated by the receiver). This is like "FMS without targets" -- recall that the receiver knows the inputs \( x_p \) of \( D_h \). Since \( w \) corresponds to the weight vector with the highest degree of local flatness within the "bit-box", the receiver will find the correct \( w \).

\( ED(w, w_1 - w) \) is described by a Gaussian distribution with mean zero. Hence, the description length of \( ED(w, w_1 - w) \) is \( ED(w, w_1 - w) = \epsilon w_1 \), the center of the "bit-box", cannot be known before training. However, we do know the expected description length of the box center, which is \( ED_{\text{mean}} + B(w, D_h) + k \) (\( k \) is a constant independent of \( w \)). Let us approximate \( ED_{\text{mean}} = \frac{1}{V(\tilde{w})} \int_{\tilde{w}}^{w} \frac{1}{\tilde{w}} \sum_{\hat{w}} \left( \frac{\delta \tilde{w}_{ij}}{\delta \hat{w}_{ij}} \right)^2 \prod_{\hat{w}, \hat{w} \neq \hat{w}_j} \delta \hat{w}_{ij} \right) \right) \right) \frac{1}{\hat{w}} \sum_{j} \frac{\delta \hat{w}_{ij}}{\delta \hat{w}_{ij}} \right) \frac{1}{\hat{w}} \sum_{j} \frac{\delta \hat{w}_{ij}}{\delta \hat{w}_{ij}} \right) \}

Among those \( w \) that lead to equal \( B(w, D_h) \) (the negative logarithm of the box volume plus \( W \log 2 \)), we want to find those with minimal description length. Using Lagrange multiplicators, it can be shown that \( ED_{\text{mean}} \) is minimal under the condition \( B(w, D_h) = \text{constant iff flatness condition 2 holds} \). To conclude: with given box volume, we need flatness condition 2 to minimize the expected description length of the box center.

Comments. Flatness condition 2 influences the algorithm as follows: (1) The algorithm prefers to increase the \( \Delta w_{ij} \)'s of weights which currently are not important to generate the target output. (2) The algorithm enforces equal sensitivity of all output units with respect to the weights. Hence, if certain hidden units help to compute targets only for a subset of the output units, then the sensitivity of the remaining output units with respect to those hidden units will be reduced; the algorithm tends to group hidden units according to their relevance for groups of output units.
A.4. 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 A.3 and approximation (8) of $E_{F_o}$ in A.3.

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. For simplicity, in what follows we use the same activation function $f$ for all units. We consider a strictly layered feedforward network with $K$ output units and $g$ layers. We obtain

$$\frac{\partial y}{\partial w_{ij}} = f'(s_i) \begin{cases} y_i^j & \text{for } i = l \\ \sum_m w_{jm} \frac{\partial y}{\partial w_{ij}} & \text{for } i \neq l \end{cases},$$

(12)

where $y^i$ denotes the activation of the $i$-th unit, and $s_i = \sum_m w_{im} y^m$.

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 (12) we obtain:

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

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

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

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

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

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

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

$$\sum_{i,j, \text{ where unit } i \text{ in a layer } < (g-2)} \log \sum_k \left( f'(s_i) \sum_l f'(s_{kl}) w_{kl} \sum_{l_2} \frac{\partial y_j^{i_2}}{\partial w_{ij}} \right)^2$$

(13)

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 \( \sum_{j} (f'(s_j) \sum_{i} f'(s_i) w_{ij} \sum_{k} \ldots \sum_{k} f'(s_j) w_{ij} w_{ij})^2 \) 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 \( \sum_{j} (f'(s_j) w_{ij})^2 \)). These
terms also decrease in proportion to unit \( j \)'s fan-in. Analogously, equation (13) 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 \( w_{ij} \) 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 4.2.

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

\[
\frac{\partial^2 \sigma^k}{\partial w_{ij} \partial w_{uv}} = \frac{f''(s_j)}{(f'(s_j))^2} \frac{\partial \sigma^k}{\partial w_{ij}} \frac{\partial \sigma^k}{\partial w_{uv}} + f'(s_j) \left( \sum_{l} w_{li} \frac{\partial^2 \sigma^k}{\partial w_{ij} \partial w_{uv}} + \delta_{il} \frac{\partial y^l}{\partial w_{uv}} + \delta_{il} \frac{\partial y^l}{\partial w_{ij}} \right),
\]

where \( \delta \) is the Kronecker-Delta. Searching for big boxes, we run into regions of acceptable minima
with \( \sigma^k \)'s close to target (section 2). Thus, by scaling the targets, \( f'(s_j) \) can be bounded.
Therefore, the first term in equation (14) decreases during learning.

According to the analysis above, the first order derivatives in the second term of (14) are
pushed towards zero. So are the \( w_{ij} \) of the sum in the second term of (14).

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

\[
\left| \frac{\partial^2 y^l}{\partial w_{ij} \partial w_{uv}} \right| = \left| \delta_{il} \delta_{ij} f''(s_l) y^l \right| < C_1,
\]

where \( y^l \), \( y^u \) 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

\[
\left| \frac{\partial^2 y^l}{\partial w_{ij} \partial w_{uv}} \right| = \left| f''(s_l) (w_{li} y^l + \delta_{il} y^l) (w_{ju} y^l + \delta_{ij} y^l) + f'(s_l) (w_{li} \delta_{ij} f''(s_l) y^l + \delta_{ij} \delta_{ij} f'(s_l) y^l + \delta_{ij} \delta_{ij} f'(s_l) y^l) \right| < C_2,
\]

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 \( H_{ij,uv} \) decrease where \( |\delta_{ij} w_{ij}| \) or
\( |\delta_{ij} w_{uv}| \) increase.
A.5. EFFICIENT IMPLEMENTATION OF THE ALGORITHM

Outline. We first explicitly compute the derivatives of (1). Then we show how to use Pearlmutter and Moller’s algorithm [26, 35] to speed up the computation of second order terms (A.5.2).

A.5.1 EXPLICIT DERIVATIVE OF EQUATION (1)

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

$$\frac{\partial B(u, D_0)}{\partial w_{uv}} = \sum \delta_{i,j} \frac{1}{\sqrt{\sum_m (\frac{\partial w_{uv}}{\partial \theta_{ij}})}} \sum (\text{sign}(\frac{\partial w_{uv}}{\partial \theta_{ij}})) \frac{\partial w_{uv}}{\partial \theta_{ij}} \sum_m \frac{\partial w_{uv}}{\partial \theta_{ij}} \sum_m \frac{\partial w_{uv}}{\partial \theta_{ij}} \sum_m \frac{\partial w_{uv}}{\partial \theta_{ij}}$$

$$= \frac{1}{\sqrt{\sum_m (\frac{\partial w_{uv}}{\partial \theta_{ij}})}} \sum (\text{sign}(\frac{\partial w_{uv}}{\partial \theta_{ij}})) \frac{\partial w_{uv}}{\partial \theta_{ij}} \sum_m \frac{\partial w_{uv}}{\partial \theta_{ij}} \sum_m \frac{\partial w_{uv}}{\partial \theta_{ij}} \sum_m \frac{\partial w_{uv}}{\partial \theta_{ij}}$$

To compute (2), we need

$$\frac{\partial B(u, D_0)}{\delta_{i,j}} = \sum \frac{\partial w_{uv}}{\partial \theta_{ij}}$$

$$= \frac{1}{\sqrt{\sum_m (\frac{\partial w_{uv}}{\partial \theta_{ij}})}} \sum (\text{sign}(\frac{\partial w_{uv}}{\partial \theta_{ij}})) \frac{\partial w_{uv}}{\partial \theta_{ij}} \sum_m \frac{\partial w_{uv}}{\partial \theta_{ij}} \sum_m \frac{\partial w_{uv}}{\partial \theta_{ij}} \sum_m \frac{\partial w_{uv}}{\partial \theta_{ij}}$$

where $\delta$ is the Kronecker-Delta. Using the nabla operator and (18), we can compress (17):

$$\nabla_{uv} B(w, D_0) = \sum \mathbb{H}^i (\nabla_{\theta_{ij}} B(w, D_0))$$

where $\mathbb{H}$ is the Hessian of the output $\phi$. Since the sums over $i, j$ in (18) need to be computed only once (the results are reusable for all $i, j$), $\nabla_{\theta_{ij}} B(w, D_0)$ can be computed in $O(W)$ time.

The product of the Hessian and a vector can be computed in $O(W)$ time (see next section). With constant number of output units, the computational complexity of our algorithm is $O(W)$.

A.5.2. FAST MULTIPLICATION BY THE HESSEAN

Pearlmutter and Moller compute the product of a vector and the Hessian of the error in $O(W)$ time [26, 35]. Using Pearlmutter’s notation, we do the same with the Hessian of the output. An operator $R$ is defined as follows:

$$R_y \{g(x)\} \equiv \frac{\partial}{\partial t} g(x + ty) \big|_{t=0}$$

The Hessian of the $k$th output $\phi^k$ of a feedforward net is computed in 3 successive passes:

1. First backward pass ($\phi = \phi^k$):

$$\frac{\partial y^i}{\partial y^j} = \begin{cases} 1 & \text{for } i = l \\ \sum_j w_{ij} \frac{\partial y^j}{\partial y^k} & \text{for } i \neq l \end{cases}$$

23
\[
\frac{\partial y^j}{\partial s_i} = f'(s_i) \frac{\partial y^j}{\partial y^i}.
\]
\[
\frac{\partial y^j}{\partial w_{ij}} = y^j \frac{\partial y^j}{\partial s_j}.
\]

2. First forward pass:
\[
R\{s_i\} = \sum_j (w_{ij} R\{y^j\} + \frac{\partial B(w, D_0)}{\partial (\frac{\partial y^j}{\partial w_{ij}})} y^j),
\]
\[
R\{y^j\} = \begin{cases} 0 & \text{for } y^j \text{ input} \\ R\{s_i\} f'_i(s_i) & \text{otherwise} \end{cases}.
\]

3. Second backward pass \((y' = \delta)\):
\[
R\{\frac{\partial y^j}{\partial y^i}\} = \begin{cases} \sum_j \left( w_{ij} R\{\frac{\partial y^j}{\partial y^i}\} + \frac{\partial B(w, D_0)}{\partial (\frac{\partial y^j}{\partial w_{ij}})} y^j \right) & \text{for } y^j \text{ in layers not below } y' \\ 0 & \text{for } y^j \text{ in layers below } y' \end{cases},
\]
\[
R\{\frac{\partial y^j}{\partial s_i}\} = f'_i(s_i) R\{\frac{\partial y^j}{\partial y^i}\} + R\{s_i\} f'_i(s_i) \frac{\partial y^j}{\partial y^i},
\]
\[
R\{\frac{\partial y^j}{\partial w_{ij}}\} = y^j R\{\frac{\partial y^j}{\partial s_j}\} + R\{y^j\} \frac{\partial y^j}{\partial s_j}.
\]

The elements of the vector \(H^j(\nabla_{w_{ij}} B(w, D_0))\) are \(R\{\frac{\partial \sigma}{\partial w_{ij}}\}\), see equation (19). Using the technique in [35], recurrent networks can be dealt with as well.

References