*LETTER* Communicated by Shun-ichi Amari

# On "Natural" Learning and Pruning in Multilayered Perceptrons

**Tom Heskes**
*Foundation for Neural Networks, University of Nijmegen, 6525 EZ Nijmegen, The Netherlands*

**Several studies have shown that natural gradient descent for on-line learning is much more efficient than standard gradient descent. In this article, we derive natural gradients in a slightly different manner and discuss implications for batch-mode learning and pruning, linking them to existing algorithms such as Levenberg-Marquardt optimization and optimal brain surgeon.**

**The Fisher matrix plays an important role in all these algorithms. The second half of the article discusses a layered approximation of the Fisher matrix specific to multilayered perceptrons. Using this approximation rather than the exact Fisher matrix, we arrive at much faster "natural" learning algorithms and more robust pruning procedures.**

## 1 Introduction

Natural gradient has recently received quite a lot attention (Amari, 1998; Rattray, Saad, & Amari, 1998; Yang & Amari, 1998). The basic idea is to replace the standard gradient by the natural gradient, defined as the steepest descent direction in Riemannian space (Amari, 1998). It has been shown (Rattray et al., 1998; Yang & Amari, 1998) that the use of natural gradients in on-line backpropagation learning for multilayered perceptrons (MLPs) leads to a tremendous speed-up in convergence. One of the important observations is that natural gradients greatly help to break the symmetry between hidden units (Rattray et al., 1998), which, using standard gradients, is the cause of long plateaus (long time spans in which the performance of the MLP barely improves—also referred to as temporary minima; see, among others, Ampazis, Perantonis, & Taylor, 1999; Saad & Solla, 1995; Wiegerinck & Heskes, 1996).

The goal of this article is twofold:

- *A different formulation of natural gradients* (section 2). We will derive "natural" gradients in a slightly different way. In this formulation, the appropriate metric follows naturally from the error function one tries to minimize (section 2.1). We discuss the overlap and differences with Amari's approach (section 2.2). The natural equivalents of batch-mode

 

learning and pruning can be defined and linked to existing algorithms (sections 2.3 and 2.4).

- *An efficient approximation of the Fisher matrix for MLPs* (sections 3 and 4). The Fisher matrix plays a key role in the theory about natural gradients. In practical applications of "natural" rather than "standard" learning or pruning procedures, one constantly has to compute and invert this matrix of size $n_{\text{weights}} \times n_{\text{weights}}$ with $n_{\text{weights}}$ the total number of network weights. The proposed approximation loosens this computational burden by orders of magnitudes (section 4.1). For on-line learning, we show that the approximated Fisher matrix is sufficiently powerful to help break the symmetry between hidden units and escape from the corresponding plateau in the same order of iterations as with the exact Fisher matrix (section 4.2). In the context of batch-mode learning, we arrive at an alternative to Levenberg-Marquardt optimization, which can be much more efficient, especially for large networks (section 4.3). Pruning algorithms based on the approximated Fisher matrix already exist (although not interpreted in this way), and although in principle they are less powerful, they are claimed to be more robust than pruning algorithms using the exact Fisher matrix (Laar & Heskes, 1999) (section 4.4).

## 2 Natural Gradient in a Slightly Different Formulation

**2.1 On-line Learning.** We are concerned here with supervised learning. Given an input $x$, the output of a model with parameters $W$ reads $y(W, x)$. Both $x$ and $y$ can be vectors. We will use $T$ to denote the transpose. In supervised learning, this output is compared with a given target vector $t$, and the error between the two is defined as $d(y(W, x), t)$. Well-known error functions are sum-squared error, $d(y, t) = (y-t)^2/2$ and cross-entropy $d(y, t) = t \log[t/y] + (1 - t) \log[(1 - t)/(1 - y)]$, but other error functions might be chosen as well, as long as they are at least twice differentiable with respect to $y$.

In on-line learning, a weight update is based on one combination of inputs $x$ and targets $t$. The standard stochastic gradient procedure is

$$\Delta W \equiv \hat{W} - W = -\eta \frac{\partial d(y(W, x), t)}{\partial W}, \tag{2.1}$$

with $\eta$ a (usually small) learning rate, $\hat{W}$ the new, and $W$ the current set of parameters. Amari's work on natural gradient learning (see, e.g., Amari, 1998) has made very clear that this particular choice for the gradient is rather arbitrary and should be replaced by natural gradients, defined as the steepest descent direction in Riemannian space. Here we use a slightly different formulation to arrive at more or less the same results. The advantage of our formulation is that we can achieve similar "natural" gradients for any kind

of (twice differentiable) error function $d(y, t)$, without the need to introduce (manifolds of) probability distributions or Riemannian spaces.

First note that the standard gradient descent procedure (see equation 2.1) is, in lowest order of the learning parameter $\eta$, equivalent to[1]

$$\hat{W} = \underset{W'}{\operatorname{argmin}} \left[ \eta d(y(W', x), t) + \frac{1}{2} \|W' - W\|^2 \right], \tag{2.2}$$

with $\|W\|^2 \equiv \sum_i W_i^2$ the Euclidean distance. The first term expresses the desire to learn the new training example and the second term the need to be conservative, that is, to retain the information acquired in previous learning trials (Kivinen & Warmuth, 1997).

The obvious question is, Why use the Euclidean distance and not some other distance measure? Indeed the "natural" choice would be to use the same distance[2] function for comparing the outputs of the new model with the given targets as for comparing them with those of the old model: $d(y(W', x), y(W, x))$. To make this distance independent of the currently presented input $x$, we take the average over the distribution of all inputs; that is, we define

$$D(W', W) = \left\langle d(y(W', x), y(W, x)) \right\rangle_x . \tag{2.3}$$

This distance function is independent of the distribution of the targets $t$. On the other hand, it depends on the distribution of the inputs $x$, so the underlying assumption is that we know or at least can estimate this distribution. We will discuss a batch-mode procedure, for which this is obviously not a problem. However, in the on-line version, knowledge of the input distribution is indeed an important assumption, on which all studies on natural gradient procedures for supervised learning are based.

So replacing in equation 2.2 the Euclidean distance by the more "natural" $D(W', W)$ of equation 2.3, we obtain

$$\hat{W} = \underset{W'}{\operatorname{argmin}} \left[ \eta d(y(W', x), t) + D(W', W) \right], \tag{2.4}$$

---

[1] For an exact correspondence with the gradient 2.1, we should write

$$\Delta W = \eta \lim_{\epsilon \to 0} \frac{1}{\epsilon} \left\{ \underset{W'}{\operatorname{argmin}} \left[ \epsilon d(y(W', x), t) + \frac{1}{2} \|W' - W\|^2 \right] - W \right\}.$$

In fact, this kind of limit can be viewed as an alternative definition of a gradient to be compared with the schoolbook definition of a difference in the limit $\epsilon$ to zero, but this seems unnecessarily complicated for the current purposes.

[2] Since we do not require $d(y, t) = d(t, y)$ the term *distance* is not always appropriate and could be replaced with *divergence*. We stick with *distance* because it does provide the correct sense.

which, in lowest order of $\eta$, corresponds to the gradient procedure

$$\Delta W = -\eta F^{-1}(W) \frac{\partial d(y(W,x),t)}{\partial W}, \text{ with } F(W) \equiv \left. \frac{\partial^2 D(W', W)}{\partial W' \partial W'^T} \right|_{W'=W}. \quad (2.5)$$

We refer to $F(W)$ as the Fisher matrix, although it need not be a Fisher matrix according to its stricter definition as in equation 2.8 below.

In computing the Fisher matrix, it is quite convenient that all second-order derivatives of the outputs with respect to the parameters $W$ disappear; that is, we have

$$\left. \frac{\partial^2 d(y(W', x), y(W, x))}{\partial W' \partial W'^T} \right|_{W'=W} = \frac{\partial y(W,x)}{\partial W} h(y(W,x)) \frac{\partial y(W,x)}{\partial W^T}, \quad (2.6)$$

with

$$h(y) = \left. \frac{\partial^2 d(y', y)}{\partial y' \partial y'^T} \right|_{y'=y}. \quad (2.7)$$

In other words, we only have to compute the second-order derivative $h(y)$ of the error function with respect to the outputs, not the second-order derivative of the outputs with respect to the parameters $W$, which does appear in the computation of the Hessian matrix (see also section 2.3).

**2.2 Relation to Amari's Natural Gradients.** In this section we first show that our definition of a "natural" gradient does indeed correspond to Amari's expressions for error functions of the log likelihood or Kullback-Leibler type. Next we discuss when and how the two definitions of "natural" gradients differ.

Let us consider the special case where the outputs $y$ and targets $t$ can be interpreted as probabilities or parameters in a probability density, and the error function $d(y, t)$ is the Kullback-Leibler divergence between the probabilities implied by $y$ and $t$:

$$d(y, t) = \int dz\, p(z|t) \log \left[ \frac{p(z|t)}{p(z|y)} \right].$$

Both the sum-squared error ($y$ refers to the mean of a gaussian distribution with fixed variance, that is, $p(z|y) \sim \mathcal{N}(y, \sigma)$ with $\sigma$ fixed and known) and the cross-entropy ($y$ is the class probability, that is, $p(z = 1|y) = y$ and $p(z = 0|y) = 1 - y$; the integral can be replaced by a sum over $z = \{0, 1\}$), described in the beginning of section 2.1, can be written in this way up to irrelevant constants. For this kind of error function, we obtain, after some straightforward manipulations,

$$F(W) = \frac{\partial^2}{\partial W' \partial W'^T} \left\langle \int dz\, p(z|W, x) \log \left[ \frac{p(z|W, x)}{p(z|W', x)} \right] \right\rangle_x \Bigg|_{W'=W}$$

$$= \left\langle \int dz\, p(z|W,x) \frac{\partial \log p(z|W,x)}{\partial W} \frac{\partial \log p(z|W,x)}{\partial W^T} \right\rangle_x , \tag{2.8}$$

the Fisher information (Amari 1985, 1998). In other words, we have shown here that in cases where the error function to be minimized can be interpreted as a Kullback-Leibler divergence, Amari's natural gradient learning coincides with the formulation in equation 2.4.

To understand when and how the two definitions of "natural" differ, let us first rephrase Amari's arguments in our context. Given a manifold of probability distributions parameterized by $W$, the most general definition of information divergence under some invariance considerations and obvious constraints for divergences (nonnegativity, zero iff the two distributions are equivalent, triangle inequality) is the so-called $f$-divergence (Amari, 1985),

$$D(W', W) = \left\langle \int dz\, p(z|W,x) f\left( \frac{p(z|W',x)}{p(z|W,x)} \right) \right\rangle_x , \tag{2.9}$$

where $f(\cdot)$ is any at least twice differentiable convex function, strictly positive, except that $f(1) = 0$. The Kullback-Leibler divergence follows from $f(r) = r - \log r - 1$. It is easy to show that any choice of $f(\cdot)$ yields, up to an irrelevant multiplying constant, the same Fisher information (see equation 2.8). This makes the Fisher information matrix the preferred metric in the manifold of probability distributions—that is, the only metric that is invariant under reparameterizations.

In Amari's formulation, there is no direct link between the first and the second term on the right-hand side of equation 2.4. No matter what the first term is, the second term is an $f$-divergence of the form equation 2.9, and thus the gradient of the first term has to be transformed using the Fisher information metric, equation 2.8. In the formulation presented in section 2.1, there does exist a direct link between the two terms. Here the chosen distance function, since it depends on only the model outputs given the model parameters, by definition also ensures invariance under reparameterization. However, it is not unique in the sense that any distance function with appropriate invariance conditions gives rise to the same Fisher matrix (see equation 2.5). The remaining indifference stems from the fact that different choices for $d(y', y)$ in the definition of $D(W', W)$ can lead to different second-order derivatives $h(y)$ in equation 2.7, which yield different Fisher matrices in equation 2.5.

As long as the error function $d(y, t)$ can be interpreted as some kind of $f$-divergence between probability distributions implied by the model outputs $y$ and targets $t$, the two formulations yield exactly the same procedure. This is the case for most commonly used error functions such as sum-squared error and cross-entropy. The formulation of section 2.1 is more general in that it also yields some kind of "natural" gradient if there is no such interpretation. Amari's formulation, on the other hand, is more general in that it can be

applied to learning rules that cannot be derived from an error function, but for which it is possible to define the Riemannian structure (and impossible or more difficult to define an appropriate distance function). An example is the application of natural gradients to independent component analysis (see, e.g., Amari, 1998).

**2.3 Batch-Mode Learning.** The above description, like Amari's work on natural gradients, focuses on on-line learning. However, exactly the same line of reasoning can be followed for gradient-descent batch-mode learning. In batch-mode learning, a parameter update is obtained by averaging over all inputs $x$ and corresponding targets $t$; that is, in equations 2.1, 2.2, 2.4, and 2.5, we can simply replace $d(y(W, x), t)$ by $\langle d(y(W, x), t)\rangle_{x,t}$. In a batch-mode scenario, we can interpret equation 2.2 as a smooth approach to the minimum of $\langle d(y(W, x), t)\rangle_{x,t}$, trying to keep some history about the initial conditions by staying close to the previous set of parameters. This interpretation might explain some of the popularity of early stopping, where one starts with small parameters (a prior with no functional dependency between inputs and outputs) and terminates the procedure when the model starts overfitting. In this context, the distance measure used for comparing $W'$ and $W$ in equations 2.2 and 2.4 can be viewed as a kind of (dynamic) prior. Pushing this analogy even further, the Euclidean distance in equation 2.2 corresponds to a prior similar to standard weight decay, whereas the "natural" distance $D(W', W)$ of equation 2.3 implements a dynamic version of Jeffrey's prior (see any textbook on Bayesian statistics, e.g., Gelman, Carlin, Stern, & Rubin, 1995).

There is another link, maybe not so much in spirit, but more in the expressions that have to be computed, with the Levenberg-Marquardt method for least-squares fitting (see, e.g., Luenberger, 1984, and many other textbooks on optimization). The idea behind the Levenberg-Marquardt method is that the error $\langle d(y(W, x), t)\rangle_{x,t}$ is, at least close to its minimum, quite well approximated by a quadratic form. If this is indeed a good approximation, we can jump to the minimum in a single leap, through (compare with equation 2.5),

$$\Delta W = -H^{-1}(W)\frac{\partial \langle d(y(W, x), t)\rangle_{x,t}}{\partial W},$$

$$\text{with } H(W) \equiv \frac{\partial^2 \langle d(y(W, x), t)\rangle_{x,t}}{\partial W \partial W^T}. \tag{2.10}$$

Now, the Levenberg approximation of the Hessian matrix $H(W)$ for sum-squared error is nothing but our definition of the Fisher matrix:

$$H(W) \approx \left\langle \frac{\partial y(W, x)}{\partial W}\frac{\partial y(W, x)}{\partial W^T}\right\rangle_x = F(W), \tag{2.11}$$

where in the first step where the second-order derivative of $y(W, x)$ with respect to the parameters $W$ is neglected, and where the last step follows from equation 2.6 and $h(y) = 1$ for $d(y, t) = (y - t)^2/2$ in equation 2.7. In other words, except for some technicalities like adding a constant to the diagonal, Levenberg-Marquardt least-squares fitting is equivalent to a greedy version of batch-mode natural gradient learning for sum-squared error. In fact, this correspondence need not be restricted to sum-squared error: the Fisher matrix $F(W)$ for general error functions defined in equation 2.6 is indeed a valid approximation of the Hessian for these error functions, similar in spirit to the Levenberg approximation, equation 2.11, for the Hessian of sum-squared error.

**2.4 Natural Pruning.** Pruning seems quite different from learning. With pruning, one intends to remove from a trained model those parameters that hardly affect, or maybe even cause to deteriorate, the performance of the model. The reason to discuss pruning here is that most of the existing and popular pruning algorithms can be understood in terms of distance functions and metrics. In fact, as we will see, optimal brain surgeon (OBS) (Hassibi & Stork, 1993) can be interpreted as the natural equivalent of pruning based on weight magnitude.

Usually pruning is done in an iterative procedure. Starting with the full model, one successively removes those parameters that appear to be least relevant. The better the performance of the model with a particular parameter (or set of parameters) left out (i.e., the lower the error), the less relevant this parameter. To quantify the relevance of a particular subset of parameters $W_k$, one has to go through the following two steps:

1. Find the new model $\hat{W}(k)$ closest to the original $W$ according to the predefined distance function $D(W', W)$, with constraint $W'_k = 0$ (compare with equation 2.4):

$$\hat{W}(k) = \underset{W', W'_k = 0}{\operatorname{argmin}} D(W', W).$$

   Below we will elaborate on specific choices for $D(W', W)$.

2. Compute the error of this new model according to some predefined criterion $E_k = E(\hat{W}(k))$. Often used criteria are training, validation, or test error or an estimate of these (see e.g., Pedersen, Hansen, & Larsen 1996), but also the distance to the original model, as, for example, in OBS, $E(W') = D(W', W)$.

Apart from all kinds of more technical details, the principal difference between different pruning strategies is their choice for the distance function $D(W', W)$ and, to a lesser extent, their choice for the performance criterion $E(W')$. A simple, and not recommended, choice would be the Euclidean distance $D(W', W) = \|W' - W\|^2$ combined with $E(W') = D(W', W)$, yielding

$E_k = \|W_k\|^2$. The resulting algorithm is: prune the weights according to their magnitude. OBS (Hassibi & Stork, 1993) (see also its "generalized" version; Stahlberger & Riedmiller, 1997) can be interpreted as the "natural" equivalent of pruning based on weight magnitude. The distance function used in OBS, although derived from a different perspective, is the quadratic approximation of the "natural" distance, equation 2.3:

$$D(W', W) = \left\langle d(y(W', x), y(W, x)) \right\rangle_x \approx \frac{1}{2}(W' - W)^T F(W)(W' - W), \tag{2.12}$$

with $F(W)$ the Fisher matrix given in equation 2.5. So there is a close link between (natural) pruning and learning: it all boils down to the definition of an appropriate distance between two different models.

Until now, everything has been quite general; apart from yielding an output $y$ given an input $x$, we have not made any assumptions about our model with parameters $W$. Some of the algorithms mentioned in the previous sections have been derived and introduced specifically for neural networks. But although in practice there may be some advantages to a neural interpretation or implementation, in principle, they can be applied to any parameterized model.

## 3 The Fisher Matrix for Multilayered Perceptrons

The Fisher matrix plays a key role in the theory and algorithms discussed in the previous sections. In this section, we propose a simplification of the Fisher matrix, specific to MLPs. By replacing the exact Fisher matrix with this approximation, we hope to arrive at algorithms that make the most out of the layered structure of MLPs, by either increasing their speed or improving their robustness.

**3.1 Notation.** Let us consider an MLP with $L$ layers of weights. Our notation will be slightly different from the one used in the previous section. $y_l$ is an $n_l \times 1$ vector with the output activities of the $l$th layer, where layer 0 is the input layer and layer $L$ the output layer. In other words, the outputs $y$ in the previous section are now called $y_L$; that is, the error function depends on only the outputs $y_L$. $x_l$ stands for the input activity of the units in layer $l$. They are related to $y_{l-1}$ and $y_l$ through

$$x_l = W_l y_{l-1} \text{ and } y_l = f_l(x_l), \tag{3.1}$$

with $f_l(\cdot)$ some kind of transfer function (usually a sigmoid for hidden units, by definition the identity for the input units), and $W_l$ an $n_l \times (n_{l-1}+1)$ matrix with weights, where one column has been added to include biases (e.g., by defining $y_{l0} = 1$ for all layers $l < L$). The input $x$ of the previous section

corresponds to $x_0 = y_0$. (We will still use $\langle \ldots \rangle_x$ to denote an average over all inputs and $\langle \ldots \rangle_{x,t}$ for an average over both inputs and targets.) The set of parameters $W$ is now the combination of all weights $\{W_1, \ldots, W_L\}$. For notational convenience, we assume that there are no connections across layers of hidden units. What follows can be easily generalized to this more general case.

**3.2 The Exact Fisher Matrix.** Equation 3.1 fully specifies the forward pass through the network. Gradient and Fisher matrix follow from the usual backpropagation rules. With definitions

$$g \equiv \frac{\partial d(y_L, t)}{\partial y_L} \text{ and } \gamma_l \equiv \frac{\partial y_L}{\partial x_l^T},$$

it is easy to derive

$$\frac{\partial d(y_L(W, x), t)}{\partial W_l} = [\gamma_l^T g] y_{l-1}^T, \tag{3.2}$$

where the $n_L \times (n_{l-1} + 1)$ matrices $\gamma_l$ can be computed efficiently using the backprop rule,

$$\gamma_{l-1} = \gamma_l W_l f'(x_{l-1}).$$

The components of the Fisher matrix directly follow from equation 2.6 combined with the above definitions:

$$F_{ll'}(W) = \left\langle \gamma_l^T h \gamma_{l'} \, y_{l'-1} y_{l-1}^T \right\rangle_x, \tag{3.3}$$

where $h \equiv h(y_L)$ from equation 2.7.

**3.3 An Approximate Fisher Matrix for MLPs.** Although the expressions may seem straightforward, what we have to deal with is a full $n_{\text{weights}} \times n_{\text{weights}}$-dimensional matrix with $n_{\text{weights}}$ the total number of weights. We have to compute and, perhaps even worse, invert this matrix each iteration. For example, this is what makes Levenberg-Marquardt optimization unpopular for applications with large MLPs. Do we really need this complexity?

Considering the way we arrived here, maybe not. The Fisher matrix follows from our choice of a "natural" metric to compute the distance between the network before and after the update. As we will see, an approximation of the Fisher matrix leads to a different choice of this metric. Or, the other way around, maybe we can choose a different metric that leads to a much faster "natural" learning procedure. Here we will arrive at such a metric by simplifying the exact Fisher matrix.

The first simplification is to treat different layers independently; we make the Fisher matrix block diagonal by ignoring the cross-terms that occur from changes in weights across layers:

$$F_{ll'} \approx F_{ll}\delta_{ll'} = \left\langle \gamma_l{}^T h \gamma_l \, y_{l-1} y_{l-1}^T \right\rangle_x \delta_{ll'}.$$

There is a very good reason to do this: these effects—for example, that a change in a weight can be compensated for by a change in a weight in a different layer—are highly nonlinear, so the quadratic approximation implied by the Fisher matrix may not be accurate at all (see Laar & Heskes, 1999, and section 4.4).

The other simplification is to neglect correlations between the derivatives $\gamma_l$ and the activities $y_{l-1}$:

$$F_{ll'} \approx [\Gamma_l \otimes C_{l-1}]\delta_{ll'} \text{ with } \Gamma_l \equiv \left\langle \gamma_l{}^T h \gamma_l \right\rangle_x \text{ and } C_l \equiv \left\langle y_l y_l^T \right\rangle_x. \tag{3.4}$$

Again, this seems to be a reasonable approximation: the term $\gamma_l{}^T h \gamma_l$ measures the impact of changes in hidden unit activities on the error. That is, it depends on what happens upstream (picturing an information flow from input to output), whereas the activities $y_{l-1}$ are directly affected by what happens downstream.

Now we can go back and derive the metric that corresponds to this approximation of the Fisher matrix. First, we note that the quadratic approximation, equation 2.12, yields exactly the same natural gradient procedure as the original $D(W', W)$, since only the second-order derivative with respect to the first argument matters. By construction it is quadratic in $W'$ but not in $W$ (for nonconstant $F(W)$). The approximate distance function, which follows from substitution of the approximate Fisher matrix, equation 3.4, in equation 2.12, and some rewriting, has the same asymmetry:

$$D_{\text{approx}}(W', W) = \frac{1}{2} \sum_{l=1}^{L} \left\langle [x_l(W) - W_l' y_{l-1}(W)]^T \Gamma_l(W) [x_l(W) - W_l' y_{l-1}(W)]^T \right\rangle_x, \tag{3.5}$$

with $x_l(W)$ and $y_l(W)$ the incoming and outgoing activities of the units in layer $l$ for a network with parameters $W$. In other words, the approximate Fisher matrix of equation 3.4 corresponds to a sum-squared error between the activities in all layers of the networks, weighted by $\Gamma_l(W)$ which measures how, on average, changes in these activities affect the error on the outputs of the network.

## 4 Implications for Learning and Pruning

**4.1 Inversion of the Fisher Matrix.** The reduction in computational complexity for inverting the Fisher matrix is enormous. We have to invert

the matrices $C_l$ of size $(n_l + 1) \times (n_l + 1)$ for $l = 0$ to $L - 1$ and the matrices $\Gamma_l$ of size $n_l \times n_l$ for $l = 1$ to $L$. To give a rough order of magnitude, suppose that all layers are of about the same size $n_l = n$ and that inverting an $n \times n$ matrix requires $n^3$ executions. Then, neglecting biases, inverting the exact Fisher matrix requires $L^3n^6$ (total number of weights cubed) executions[3] whereas inverting the approximated Fisher matrix requires only $2Ln^3$ executions ($2L\times$ number of units per layer cubed). For a moderate MLP with one layer of hidden units ($L = 2$) and $n = 5$ units per layer, we are already talking about a factor 250 speed-up. Computing the forward and backward pass through the network to compute the batch-mode gradient each requires on the order of $Ln^2p$ executions (number of weights $\times$ the number of patterns $p$), which is for reasonable amounts of patterns ($p$ of the same order as $n$) about as time-consuming as inverting the approximate Fisher matrix. Computing the approximate Fisher matrix is only about twice as time-consuming as computing the batch-mode gradient. Note further that $C_0$, the covariance matrix of the network inputs, needs to be computed and inverted only once (in principle; see, however, section 4.3).

So, roughly speaking, to use the "natural" batch-mode gradient with the approximated Fisher matrix makes batch-mode learning only a factor 2 slower when all layers are of about the same size. Large variations in the size of the layers and a relatively small number of patterns increase the relative overhead, but using the approximate Fisher is always an order of magnitude faster than working with the exact Fisher matrix.

**4.2 Approximate On-line "Natural" Learning.** Combining equations 2.5, 3.2, and 3.4, we obtain for the weights $W_l$ in layer $l$ the on-line learning rule

$$\Delta W_l = -\eta F_{ll}^{-1}(W)\frac{\partial d(y_L(W, x), t)}{\partial W_l} = \left\{\Gamma_l^{-1}[\gamma_l^T g]\right\}\left\{y_{l-1}^T C_{l-1}^{-1}\right\}. \qquad (4.1)$$

The second term between braces is the output activity of the preceding layer of units $y_{l-1}$, but then normalized and decorrelated by the inverse of the corresponding covariance matrix $C_{l-1}$. This idea, to decorrelate the activities of the layers before computing the weight update, is not new (see, e.g., Almeida & Silva, 1992), but here it comes out naturally as an approximation to natural gradient learning. An interpretation of the first term between braces is slightly more involved. Recall that

$$[\gamma_l^T g] = \frac{\partial d(y_L(W, x_l), t)}{\partial x_l}$$

[3] In some special cases, where the input distribution is known, e.g., gaussian, the Fisher matrix can be calculated analytically and inverted even in order $n_{\text{input}}^2$ time, if $n_{\text{input}}$, the number of inputs, is much larger than the number of hidden and output units (Yang & Amari, 1998).

$$\text{and } \Gamma_l = \left\langle \left. \frac{\partial^2 d(y_L(W, x_l'), y_L(W, x_l))}{\partial x_l' \partial x_l'^T} \right|_{x_l'=x_l} \right\rangle_x ,$$

that is, $[\gamma_l^T g]$ measures the impact of a change in $x_l$ on the error $d(y_L(W, x_l), t)$, while $\Gamma_l$ is some kind of Fisher matrix, but now considering changes with respect to $x_l$ rather than with respect to a set of parameters. The general effect of incorporating curvature information in this way is to make larger steps in flat directions and smaller steps in steep directions. So here, by multiplying the gradient $[\gamma_l^T g]$ with the inverse of the curvature $\Gamma_l$, updates are more focused to create differences between the activities of units in the same layer. The tendency of hidden units to represent the same features, which often appears in the initial stages of learning, is the cause of the notorious plateaus, long time spans in which the performance of the MLP hardly changes. In previous studies it has been shown that incorporating curvature information helps a great deal to break the symmetry between the hidden units and thus considerably shortens the length of the plateau (see, e.g., Rattray et al., 1998; Yang & Amari, 1998). The obvious question is then whether we need the full Fisher matrix to achieve symmetry breaking or can do with the approximation as well.

*4.2.1 Learning the Tent Map.* To answer this question, we consider a simple simulation study: on-line learning of the tent map. In various studies it has been shown that standard on-line learning of the tent map is severely hampered by the existence of a long plateau (see, e.g., Hondou & Sawada, 1994; Wiegerinck & Heskes, 1996). In this work, the effect of correlations between subsequently presented patterns has been studied. Here we simply view it as a static problem, with a fixed training set of 100 patterns. The 100 inputs $x$ were homogeneously drawn from the interval $[-1, 1]$, and the corresponding targets $t$ obey $t = 1 - |x|$. Our MLP has two hidden units (see Figure 1). Initial weights were drawn at random from a gaussian with zero average and standard deviation 0.1. We performed three runs of on-line learning, each starting from the same set of initial weights and with the same sequence of pattern presentations:

**Standard on-line.** $\Delta W = -\eta \frac{\partial d(y(W,x),t)}{\partial W}$ with $\eta = 0.1$. With a higher learning parameter, fluctuations start to become unacceptably large. A smaller learning parameter further reduces the chance to escape from the plateau.

**Exact natural gradient.** $\Delta W = -\eta [F(W) + \mu]^{-1} \frac{\partial d(y(W,x),t)}{\partial W}$ with $F(W)$ the Fisher matrix numerically computed based on the 100 inputs $x$, $\eta = 0.01$ and $\mu = 0.01$ a small constant added to the diagonal for stabilization (similar to the additional constant in Levenberg-Marquardt).

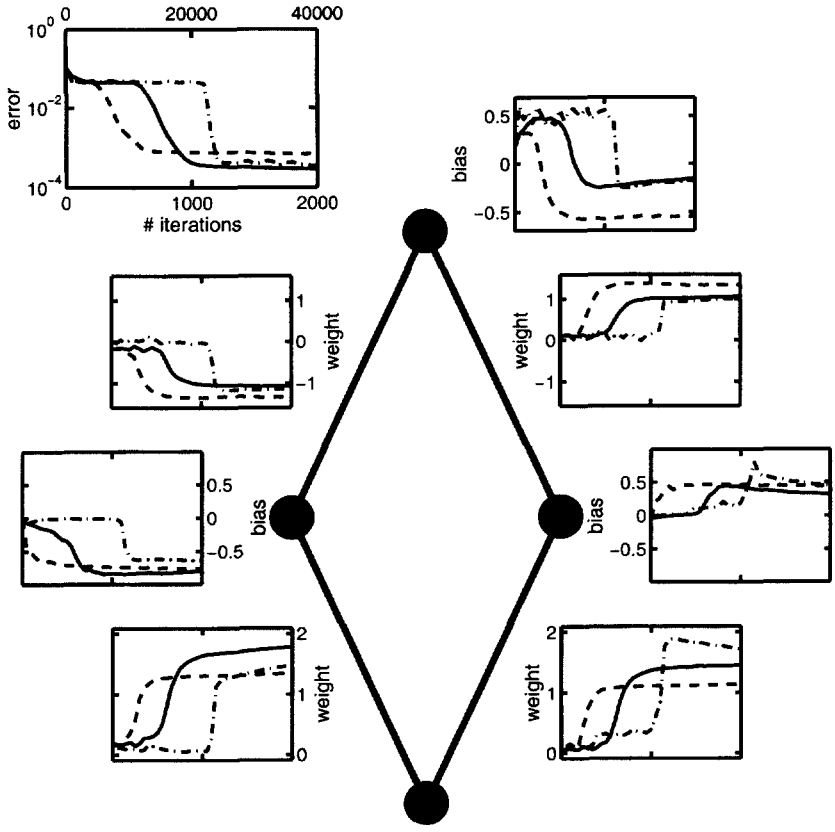


Figure 1: On-line learning of the tent map. The architecture of the network (middle), evolution of all weights and biases (on the sides), and the error (top) for three different learning procedure: standard on-line (dash-dotted; upper time axes), exact Fisher matrix (solid), and approximate Fisher matrix (dashed).

**Approximate natural gradient.** $\Delta W = -\eta[F_{\text{approx}}(W)+\mu]^{-1}\frac{\partial d(y(W,x),t)}{\partial W}$ with the same $\eta = 0.01$ and $\mu = 0.01$ as for natural gradient learning and where $F_{\text{approx}}(W)$ stands for the proposed approximation of the Fisher matrix.

In other words, except for the replacement of the exact by the approximate Fisher matrix, the conditions for the latter two runs were completely equivalent. Standard on-line learning is used for reference.

The results are displayed in Figure 1. The plot in the upper-left corner displays the error $\langle d(y(W,x),t)\rangle_{x,t}$ as a function of time (number of patterns presented) for the three runs. The upper time axis is for standard on-line

learning (dash-dotted lines); the lower time axis is for exact natural gradient (solid lines) and approximate natural gradient (dashed lines). The other plots show the evolution of all the weights and biases, where the time axes are the same as for the errors (20 standard on-line steps for 1 exact or approximate natural gradient step). Standard on-line learning has great difficulty escaping from the initial plateau. This plateau corresponds to a solution with all weights and biases close to zero except for the output bias, which is roughly 0.5, the average of the targets $t$; there is barely any functional dependence on the inputs $x$. The two runs using curvature information break the symmetry between the hidden units fairly rapidly and escape from this plateau. The run using the approximate Fisher matrix manages this even a little faster than the run using the exact Fisher matrix. After breaking this symmetry, both converge to a good solution (error $< 0.001$), the one for the exact Fisher matrix slightly better than the one for the approximate Fisher matrix. Both are still slowly improving.

A closer look shows that the run using the approximate Fisher matrix in fact "jumps" from the initial plateau to a second, much lower plateau. The first plateau—the one that is so hard to take for standard on-line learning—has to do with breaking the symmetry between the hidden units. Similar plateaus are the subject of many analytic studies on on-line learning in large networks (number of input units going to infinity with a fixed number of hidden units) (Rattray et al., 1998). The second plateau, however, is caused by the fact that small changes in the input-to-hidden weights and small (opposite) changes in the hidden-to-output weights almost cancel. This effect is neglected by the proposed approximation, yet properly taken into account by the full Fisher matrix, which does "look across layers of hidden units."

In short, the simulation shows that the approximate Fisher matrix is sufficiently powerful to break the symmetry between the units and reduces the length of the plateau by the same order of magnitude as the full Fisher matrix. A second plateau, which has to do with interdependencies between weights in different layers, cannot be tackled efficiently using the approximate Fisher matrix, but disappears when using the exact Fisher matrix.

*4.2.2 Escape Times for Symmetry Breaking.* The simulation described above is just a single instance of a single problem. To check whether the general picture is more or less reproducible, we have compared the three different learning strategies (standard on-line, exact natural gradient, and approximate natural gradient) on three different on-line learning problems and averaged over many different instances generated by randomizing over both initial conditions and the presentation order of the training patterns. We focus on the number of iterations it takes to escape from the first plateau that requires breaking of the symmetry between hidden units.

We considered the following problems. Unless specified otherwise, networks were initialized with weights drawn independently from a gaussian distribution with mean zero and standard deviation 0.01, had two hidden

units with hyperbolic tangent transfer functions, and were trained to minimize sum-squared error. For standard on-line, we took learning parameter $\eta = 0.1$, for both exact and approximate natural gradient learning $\eta = 0.01$ with a constant $\mu = 0.01$ added to the diagonal to prevent ill-conditioning. In all simulations, these appeared to be close to optimal settings for the learning parameters. With much smaller learning parameters breaking the symmetry takes (even) more time, much larger learning parameters lead to numerical instabilities.

**XOR.** Two binary inputs, one output, $t = 0$ for $x = \{1, 1\}$ and $x = \{-1, -1\}$, $t = 1$ for $x = \{1, -1\}$ and $x = \{-1, 1\}$. Each iteration, one of the four training patterns is drawn at random.

**Tent map.** As described above, with $x$ at each iteration drawn homogeneously from the interval $[-1, 1]$ and $t = 1 - |x|$. Exact and approximate Fisher matrix were obtained through numeric integration.

**Soft-committee.** A kind of standard problem used for analyzing neural network learning in the statistical mechanics framework (see, e.g., Rattray et al., 1998). Here we considered $N = 10$ inputs, at each iteration independently drawn from a gaussian with zero mean and unit standard deviation. The target comes from a "teacher," which is another neural network with exactly the same architecture as the "student." Both student and teacher have no thresholds, one output, and all hidden-to-output weights fixed to one. To obtain plateaus of about the same length as for the other problems, we had to initialize to extremely small weights, drawn from a gaussian distribution with standard deviation $10^{-15}$ rather than 0.01 (apparently symmetry breaking in soft-committee machines is relatively much easier than in learning the XOR problem or the tent map). Exact and approximate Fisher matrix were obtained through numeric integration (which is why we considered a relatively small network). Besides randomizing the initial conditions of the "student" and the inputs $x$ drawn at each iteration, we also randomized over "teachers": at the beginning of each run, the input-to-hidden weights of the teacher network were independently drawn from a gaussian distribution with standard deviation $1/\sqrt{N}$.

On each of the problems, we performed 1000 independent runs for each of the on-line learning procedures. For each run, we carried out the number of iterations needed to escape from the plateau—the number of iterations before the training error reached a level somewhere in the middle between the error on the plateau (before symmetry breaking) and close to optimal (after symmetry breaking). The results are displayed in Figure 2. With standard on-line learning (dash-dotted lines), only 11 out of 1000 networks managed to escape from the plateau within $10^4$ iterations for the XOR-problem, and none for the tent map. The results for natural learning

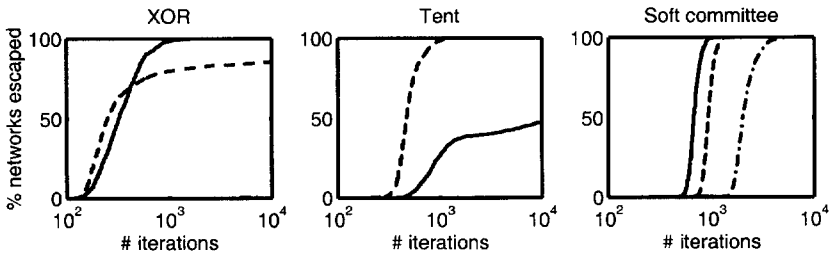


Figure 2: Number of iterations needed to escape from the plateau for the XOR problem, learning the tent map, and imitating a soft committee machine for three different on-line learning procedures: standard on-line (dash-dotted), exact Fisher matrix (solid), and approximate Fisher matrix (dashed).

using the approximate and exact Fisher matrix are comparable: where the approximate version seems to have somewhat more difficulty with breaking the symmetry between the hidden units for the XOR problem, the exact Fisher matrix has more problems with the tent map. Learning the soft committee machine appeared to be relatively easy for all procedures, and the differences between them are much less striking. As argued in Heskes and Coolen (1997), the plateaus for learning the soft committee machine and related problems studied in the statistical mechanics literature are an order of magnitude easier to tackle than the plateaus that occur when learning the tent map or the XOR function. In the former case, the plateau corresponds to a saddle point of the error: the gradient on the plateau is negligible, but the Hessian has a finite negative eigenvalue that drives the escape. In the latter two cases, the error surface really is flat: the smallest eigenvalue of the Hessian matrix is negligible as well (zero curvature), which makes it much harder to escape.

**4.3 Using the Approximate Fisher Matrix in Levenberg-Marquardt.** As explained in section 2.3, there is a close connection between batch-mode natural learning and Levenberg-Marquardt optimization. With Levenberg-Marquardt, weight changes obey (compare with equation 2.10),

$$\Delta W = -[F(W) + \mu]^{-1} \frac{\partial \langle d(y_L(W, x), t) \rangle_{x,t}}{\partial W}, \tag{4.2}$$

where $\mu$ is a constant added to the diagonal of $F(W)$ to ensure that the error after the learning step is lower than the error before the learning step (extremely large $\mu$ corresponds to a standard batch-mode gradient-descent weight update with learning parameter $\mu^{-1}$).

Simply replacing the exact $F(W)$ by its approximation, equation 3.4, would yield (compare with equation 4.1):

$$\Delta W_l = -[F_{ll}(W)+\mu]^{-1}\frac{\partial\langle d(y_L(W,x),t)\rangle_{x,t}}{\partial W_l} \text{ with } F_{ll}(W) = [\Gamma_l \otimes C_{l-1}].$$

However, adding the constant to the diagonal destroys the decomposition $F_{ll}^{-1} = \Gamma_l^{-1} \otimes C_{l-1}^{-1}$, which considerably speeds up the computation of the inversions. The alternative, with similar properties but keeping the decomposition intact, would be to add the constant not to the diagonal of $F_{ll}(W)$ but to both $\Gamma_l$ and $C_{l-1}$:

$$\Delta W_l = -[(\Gamma_l + \sqrt{\mu})^{-1} \otimes (C_{l-1} + \sqrt{\mu})^{-1}]\frac{\partial\langle d(y_L(W,x),t)\rangle_{x,t}}{\partial W_l}. \qquad (4.3)$$

For large $\mu$ we still obtain a gradient step with learning parameter $\mu^{-1}$, whereas for small $\mu$, the update is dominated by the approximated "natural" gradient.[4]

As an illustration, we compared three different batch-mode procedures for training a neural network with eight hidden units on the Boston housing data set: standard Levenberg-Marquardt as in equation 4.2, "layered" Levenberg-Marquardt as in equation 4.3, and, for reference, (almost) steepest descent (equation 4.2 with $F(W) \equiv 0$ or equation 4.3 with $\Gamma_l \equiv 0$ and $C_{l-1} \equiv 0$). All 506 patterns were used for training. The training error as a function of CPU time is plotted in Figure 3. Especially in the early stages of learning, using the approximate Fisher matrix (dashed line) considerably speeds up learning compared with both using the exact Fisher matrix (solid line) and almost steepest descent (dash-dotted line). Other runs (not shown), with a validation set left aside, revealed that the minimum of the validation error is obtained somewhere around $10^1$ seconds CPU time for Levenberg-Marquardt learning using either the approximate or the exact Fisher matrix. The inset in Figure 3 shows how the CPU time per update scales as a function of the number of hidden units for this particular problem (the number of weights is 15 times the number of hidden units plus 1).

**4.4 Natural Pruning in the Layered Approximation.** Also here, in the context of pruning, one might wonder whether using the approximate Fisher matrix instead of the exact one leads to faster or more robust pruning

[4] Another speed-up can be obtained by using

$$\Delta W_1 = -\left[(\Gamma_1 + \mu)^{-1} \otimes C_0^{-1}\right]\frac{\partial\langle d(y_L(W,x),t)\rangle_{x,t}}{\partial W_1}$$

for the weight changes in the first layer, in which case we have to compute and invert the input covariance matrix only once.

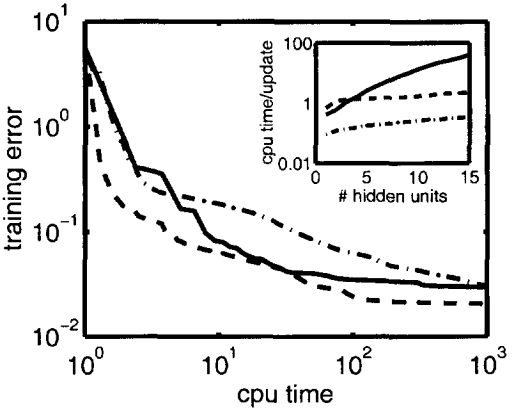


Figure 3: Levenberg-Marquardt learning on the Boston housing data using the exact Fisher matrix (solid lines), the approximate Fisher matrix (dashed lines), and the identity matrix (dash-dotted lines). The main figure shows the training error as a function of CPU time for a network with eight hidden units. The inset shows the CPU time per update for the three different procedures as a function of the number of hidden units.

procedures. The effect of replacing the exact Fisher matrix by its approximation, equation 3.4, in OBS is most easily understood by looking at the corresponding metric, equation 3.5. Finding the new weights $W$ boils down to solving single-layer problems of the form ($W'_{lk}$ denote the weight(s) to be pruned),

$$\hat{W}_l(k) = \underset{W'_l; W'_{lk}=0}{\text{argmin}} \frac{1}{2} \left\langle [x_l(W) - W'_l y_{l-1}(W)]^T \Gamma_l(W) [x_l(W) - W'_l y_{l-1}(W)]^T \right\rangle_x$$

$$= \underset{W'_l; W'_{lk}=0}{\text{argmin}} \frac{1}{2} \left\langle \|x_l(W) - W'_l y_{l-1}(W)\|^2 \right\rangle, \tag{4.4}$$

where the last step, that the solution is independent of $\Gamma_l$, can be easily verified by taking into account that $\Gamma_l$ is positive definite and symmetric. In fact, we are just fitting the weights in order to reproduce the hidden unit activities of the original network as accurately as possible. This idea is present in many independently proposed pruning algorithms (Castellano, Fanelli, & Pelillo, 1997; Egmont-Petersen, Talmon, Hasman, & Ambergen, 1998; Laar & Heskes, 1999). The link provided here, a layered approximation of the exact Fisher matrix, might explain some of their success.

In Laar and Heskes (1999), "partial retraining," an algorithm that, apart from a few minor details, is based on equation 4.4, was compared with (generalized) OBS. It came out that partial retraining is much more robust than

OBS, mainly because the quadratic approximation, equation 2.12, breaks down quite rapidly after removing a few weights. Interdependencies between weights in different layers are not accurately approximated, and the flexibility to compensate for pruned weights by changing weights in different layers is overestimated. The more conservative layered approximation (see equation 4.4) of the exact Fisher matrix appeared to be sufficiently powerful and much less sensitive to weight removals.

## 5 Discussion and Directions

We have described natural learning and pruning procedures from a slightly different perspective than Amari's orginal formulation. In all of this, the Fisher matrix, measuring the local curvature of the distance function, plays a key role. Making a block diagonal ("layered") approximation of this Fisher matrix, we arrived at fast learning and robust pruning procedures specific to MLPs.

The approximation proposed is rather straightforward and simplistic: it neglects all interdependencies between weights in different layers. Still, it appeared to be sufficiently powerful to break the symmetry between hidden units in the early stages of learning. However, in later stages, where subtle interactions between weights in different layers come into play, one may have to pay the price for this neglect.

The odds for a layered approximation in pruning, which has been discussed to a larger extent in Laar and Heskes (1999), are even better than for learning. In learning, the Fisher matrix is used only to transform the standard gradient in a smart way. To guarantee that the error indeed decreases (always for batch-mode or on average for on-line), some regularization can be added to prevent jumps that are too large in weight space. In pruning algorithms, the role of the Fisher matrix is much more critical. Here the Fisher matrix is used in a quadratic approximation of a distance function, and if this approximation breaks down, there is no feedback from an error function as in learning. This might explain why in pruning it pays to be somewhat more conservative and use a more local metric, as in equation 3.5, rather than the full quadratic approximation, in equation 2.12.

For on-line natural learning, the important assumption remains that the distribution of inputs is known. Instead, one might try to sample the required statistics dynamically, while learning. It would be interesting to see under which conditions and with what kind of schemes one could best approximate the "true" natural gradient to get the most benefit out of using the natural rather than the standard gradients.

In Yang and Amari (1998), exact expressions for the Fisher matrix are derived for gaussian input probability distributions. One could apply the same expressions as an approximation for nongaussian input distributions as well. The advantage compared to the approach taken in this article is that such an approximation can take into account dependencies among

different layers. The question is how useful the expressions still are when the shape of the input distribution starts to differ from a standard gaussian. One could also think of combining the two approaches, where the layered approximation yields a kind of zeroth order with interdependencies among weights in different layers estimated assuming gaussian inputs.

## Acknowledgments

I thank Bert Kappen and Martijn Leisink for discussions and comments on earlier drafts. This research was supported by the Technology Foundation STW, applied science division of NWO, and the technology program of the Ministry of Economic Affairs.

## References

Almeida, L., & Silva, F. (1992). Adaptive decorrelation. In I. Aleksander & J. Taylor (Eds.), *Artificial neural networks, 2* (pp. 149–156). Amsterdam: North-Holland.

Amari, S. (1985). *Differential-geometric methods in statistics.* New York: Springer-Verlag.

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation, 10,* 251–276.

Ampazis, N., Perantonis, S., & Taylor, J. (1999). Dynamics of multilayer networks in the vicinity of temporary minima. *Neural Networks, 12,* 43–58.

Castellano, G., Fanelli, A., & Pelillo, M. (1997). An iterative pruning algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks, 8,* 519–531.

Egmont-Petersen, M., Talmon, J., Hasman, A., & Ambergen, A. (1998). Assessing the importance of features for multi-layer perceptrons. *Neural Networks, 11,* 623–635.

Gelman, A., Carlin, J., Stern, H., & Rubin, D. (1995). *Bayesian data analysis.* London: Chapman & Hall.

Hassibi, B., & Stork, D. (1993). Second order derivatives for network pruning: Optimal brain surgeon. In S. Hanson, J. Cowan, & L. Giles (Eds.), *Advances in neural information processing systems, 5* (pp. 164–171). San Mateo, CA: Morgan Kaufmann.

Heskes, T., & Coolen, J. (1997). Learning in two-layered networks with correlated examples. *Journal of Physics A, 30,* 4983–4992.

Hondou, T., & Sawada, Y. (1994). Analysis of learning processes of chaotic time series by neural networks. *Progress in Theoretical Physics, 91,* 397–402.

Kivinen, J., & Warmuth, M. (1997). Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation, 132,* 1–63.

Laar, P. v. d., & Heskes, T. (1999). Pruning using parameter and neuronal metrics. *Neural Computation, 11,* 977–993.

Luenberger, D. (1984). *Linear and nonlinear programming.* Reading, MA: Addison-Wesley.

Pedersen, M., Hansen, L., & Larsen, J. (1996). Pruning with generalization based weight saliencies: $\gamma$OBD, $\gamma$OBS. In D. Touretzky, M. Mozer, & M. Hasselmo (Eds.), *Advances in neural information processing systems, 8* (pp. 521–527). Cambridge, MA: MIT Press.

Rattray, M., Saad, D., & Amari, S. (1998). Natural gradient descent for on-line learning. *Physical Review Letters, 81*, 5461.

Saad, D., & Solla, S. (1995). Exact solution for on-line learning in multilayer neural networks. *Physical Review Letters, 74*, 4337.

Stahlberger, A., & Riedmiller, M. (1997). Fast network pruning and feature extraction using the unit-OBS algorithm. In M. Mozer, M. Jordan & T. Petsche (Eds.), *Advances in neural information processing systems, 9* (pp. 655–661). Cambridge, MA: MIT Press.

Wiegerinck, W., & Heskes, T. (1996). How dependencies between successive examples affect on-line learning. *Neural Computation, 8*, 1743–1765.

Yang, H., & Amari, S. (1998). The efficiency and robustness of natural gradient descent learning rule. In M. Kearns, M. Jordan, & S. Solla (Eds.), *Advances in neural information processing systems, 10* (pp. 385–391). Cambridge, MA: MIT Press.

---

Received April 26, 1999; accepted July 12, 1999.