Technical Report IDSIA-19-97

April 19, 1997
revised August 21, 1998

Centering Neural Network Gradient Factors *

Nicol N. Schraudolph
nic@idsia.ch

IDSIA, Corso Elvezia 36
6900 Lugano, Switzerland
http://www.idsia.ch/

Abstract. It has long been known that neural networks can learn faster when their input and hidden unit activities are centered about zero; recently we have extended this approach to also encompass the centering of error signals [2]. Here we generalize this notion to *all* factors involved in the network's gradient, leading us to propose centering the slope of hidden unit activation functions as well. Slope centering removes the linear component of backpropagated error; this improves credit assignment in networks with shortcut connections. Benchmark results show that this can speed up learning significantly without adversely affecting the trained network's generalization ability.

1 Introduction

Centering is a general methodology for accelerating learning in adaptive systems of the type exemplified by neural networks — that is, systems that are typically nonlinear, continuous, and redundant; that learn incrementally from examples, generally by some form of gradient descent. Its basic tenet is:

> *All pattern-dependent factors entering the update equation for a neural network weight should be centered,* i.e., *have their average over patterns subtracted out.*

Prior work. It is well-known that the inputs to an LMS adaptive filter should be centered to permit rapid yet stable adaptation [3], and it has been argued [4] that the same applies to input and hidden unit activity in a multi-layer network. Although Sejnowski [5] proposed a variant of Hebbian learning in which both

* Reprinted from Orr and Müller (eds.), *Neural Networks: Tricks of the Trade* [1].

the pre- and postsynaptic factors of the weight update are centered, the idea was not taken up when backpropagation became popular. The benefits of centering error signals in multi-layer networks were thus reported only recently [2]; here we finally suggest centering as a general methodology, and present backpropagation equations in which *all* factors are centered.

Independence of architecture. Although centering is introduced here in the context of feedforward networks with sigmoid activation functions, the approach itself has a far wider reach. The implementation details may vary, but in essence centering is not tied to any particular architecture: its principles are equally applicable to feedforward and recurrent networks, with sigmoid, radial, or other basis functions, with or without topological structure, time delays, multiplicative gates, *etc.* — in short, the host of architectural elements used in neural network design.

Independence of learning algorithm. Similarly, centering is not wedded to any particular learning algorithm either. It may be applied to deterministic (batch) or stochastic (online) gradient descent; more importantly, it may be freely combined with more sophisticated optimization techniques such as expectation maximization, conjugate gradient and quasi-Newton methods. It also leaves available the many useful tricks often employed with stochastic gradient descent, such as momentum, learning rate adaptation, gradient normalization, and so forth. Due to this flexibility, centering has the potential to further accelerate even the fastest of these methods.

Overview. Section 2 introduces the centering approach in terms of the modifications it mandates for ordinary backpropagation learning. We then discuss implementation details in Section 3 before presenting benchmark results in Section 4. Section 5 concludes our paper with a brief analysis of how centering facilitates faster learning.

2 Centered Backpropagation

The backpropagation learning algorithm is characterized by three equations, describing the forward propagation of activity, the backpropagation of error, and the modification of weights, respectively. Here are the implications of centering for each of these three equations:

2.1 Activity Propagation

Conventional. Consider a neural network with activation of node j given by

$$x_j = f_j(y_j)\,, \quad y_j = \sum_{i \in A_j} w_{ij}\, \check{x}_i\,, \tag{1}$$

where f_j is a nonlinear (typically sigmoid) activation function, w_{ij} are the synaptic weights, and A_j denotes the set of *anterior* nodes feeding their activity $\check{x}_i$ into node j. Conventionally, the anterior nodes' output is fed forward directly, *i.e.*, $(\forall i)\ \check{x}_i \equiv x_i$. We imply that nodes are activated via Equation 1 in appropriately ordered (feedforward) sequence, and that some have their values clamped so as to represent external inputs to the network. In particular, we posit a bias input $x_0 \equiv 1$ and require that all nodes are connected to it: $(\forall j > 0)\ \ 0 \in A_j$.

Centered. As suggested by LeCun et al. [4], the activity of the network's input and hidden units should be centered to permit faster learning. We do so by setting

$$(\forall i > 0)\ \ \check{x}_i \ = \ x_i - \langle x_i \rangle\,, \tag{2}$$

where $\langle \cdot \rangle$ denotes averaging over training samples (see Section 3 for ways to implement this operator). Note that the bias input must *not* be centered — since $x_0 = \langle x_0 \rangle = 1$, it would otherwise become inoperative.

2.2 Weight Modification

Conventional. The weights w_{ij} of the network given in Equation 1 are typically optimized by gradient descent in some objective function E. With a local step size η_{ij} for each weight, this results in the weight update equation

$$\Delta w_{ij} \ = \ \eta_{ij}\, \delta_j\, \check{x}_i\,, \quad \text{where} \quad \delta_j \ = \ -\partial E / \partial y_j\,. \tag{3}$$

Centered. We have recently proposed [2] that the error signals δ_j should be centered as well to achieve even faster convergence. This is done by updating all non-bias weights via

$$(\forall i > 0)\ \ \Delta w_{ij} \ = \ \eta_{ij}\, \check{\delta}_j\, \check{x}_i\,, \quad \text{where} \quad \check{\delta}_j \ = \ \delta_j - \langle \delta_j \rangle\,. \tag{4}$$

As before, this centered update must *not* be used for bias weights, for otherwise they would remain forever stuck (except for stochastic fluctuations) at their present values:

$$* \qquad \langle \Delta w_{0j} \rangle \ \propto \ \langle \check{\delta}_j \rangle \ = \ \langle \delta_j \rangle - \langle \delta_j \rangle \ = \ 0\,. \tag{5}$$

Instead, bias weights are updated conventionally (Equation 3). Since this means that the average error $\langle \delta_j \rangle$ is given exclusively to the bias weight w_{0j}, we have previously called this technique *d.c. error shunting* [2].

2.3 Error Backpropagation

Conventional. For output units, the error δ_j can be computed directly from the objective function; for hidden units, it must be derived through error back-propagation:

$$\delta_j \;=\; f_j'(y_j)\,\gamma_j\,, \quad \gamma_j \;=\; \sum_{k \in P_j} w_{jk}\,\delta_k\,, \tag{6}$$

where P_j denotes the set of *posterior* nodes fed from node j, and $f_j'(y_j)$ is the node's current *slope* — the derivative of its nonlinearity f_j at the present level of activation.

Centered. By inserting Equation 6 into Equation 4, we can express the weight update for hidden units as a triple product of their activity, backpropagated error, and slope:

$$\Delta w_{ij} \;\propto\; f_j'(y_j)\,\check{\gamma}_j\,\check{x}_i\,, \tag{7}$$

where $\check{\gamma}_j$ denotes backpropagated *centered* errors. It is not necessary to center the $\check{\gamma}_j$ explicitly since

$$\langle \check{\gamma}_j \rangle \;=\; \left\langle \sum_{k \in P_j} w_{jk}\,\check{\delta}_k \right\rangle \;=\; \sum_{k \in P_j} w_{jk}\,\langle \check{\delta}_k \rangle \;=\; 0\,. \tag{8}$$

By centering activity and error signals we have so far addressed two of the three factors in Equation 7, leaving the remaining third factor — the node's slope — to be dealt with. This is done by modifying the nonlinear part of error backpropagation (Equation 6) to

$$\delta_j \;=\; \check{f}_j'(y_j)\,\check{\gamma}_j\,, \quad \text{where} \quad \check{f}_j'(y_j) \;=\; f_j'(y_j) - \langle f_j'(y_j) \rangle\,. \tag{9}$$

Decimation of linear errors. Note that for a *linear* node n we would have $f_n'(y_n) \equiv 1$, and Equation 9 would always yield $\delta_n \equiv 0$. In other words, slope centering (for any node) blocks backpropagation of the *linear component* of error signals — that component which a linear node in the same position would receive. Viewed in terms of error decimation, we have thus taken the logical next step past error centering, which removed the d.c. (constant) component of error signals.

Shortcuts. It was important then to have a parameter — the bias weight — to receive and correct the d.c. error component about to be eliminated. Likewise, we now require additional weights to implement the linear mapping from anterior to posterior nodes that the unit in question is itself no longer capable of. Formally, we demand that for each node j for which Equation 9 is used, we have

$$(\forall i \in A_j) \;\; P_j \subseteq P_i\,. \tag{10}$$

We refer to connections that bypass a node (or layer) in this fashion as *shortcuts*. It has been noted before that neural network learning sometimes improves with the addition of shortcut weights. In our own experiments (see Section 4), however, we find that it is slope centering that makes shortcut weights genuinely useful [6].

A complementary approach? Van der Smagt and Hirzinger [7] also advocate shortcuts as a means for accelerating neural network learning. Note, however, that their use of shortcuts is quite different from ours: in order to improve the conditioning of a neural network, they add shortcut connections whose weights are coupled to (shared with) *existing* weights. They thus suitably modify the network's topology without adding new weight parameters, or deviating from a strict gradient-based optimization framework. By contrast, we deliberately decimate the linear component of the gradient for hidden units in order to focus them on their nonlinear task. We then use shortcuts with *additional* weight parameters to take care of the linear mapping that the hidden units now ignore.

While both these approaches use shortcuts to achieve their ends, from another perspective they appear almost complementary: whereas we eliminate the linear component from our gradient, van der Smagt and Hirzinger in fact *add* just such a component to theirs. It may even be possible to profitably combine the two approaches in a single — admittedly rather complicated — neural network architecture.

3 Implementation Techniques

We can distinguish a variety of approaches to centering a variable in a neural network in terms of how the averaging operator $\langle \cdot \rangle$ is implemented. Specifically, averaging may be performed either exactly or approximately, and applied either *a priori*, or adaptively during learning in either batch (deterministic) or online (stochastic) settings:

centering method:		approximate	exact
	a priori	*by design*	*extrinsic*
adaptive	online	*running average*	—
	batch	*previous batch*	*two-pass, single-pass*

3.1 A Priori Methods

By design. Some of the benefits of centering may be reaped without *any* modification of the learning algorithm, simply by setting up the system appropriately. For instance, the hyperbolic tangent (tanh) function with its symmetric range from -1 to 1 will typically produce better-centered output than the commonly used logistic sigmoid $f(y) = 1/(1 + e^{-y})$ ranging from 0 to 1, and is therefore the preferred activation function for hidden units [4]. Similarly, the input representation can (and should) be chosen such that inputs will be roughly centered.

When using shortcuts, one may even choose *a priori* to subtract a constant (say, half their maximum) from hidden unit slopes to improve their centering.

We refer to these approximate methods as centering *by design*. Though inexact, they provide convenient and easily implemented tricks to speed up neural network learning. Regardless of whether further acceleration techniques will be required or not, it is generally a good idea to keep centering in mind as a design principle when setting up learning tasks for neural networks.

Extrinsic. Quantities that are extrinsic to the network — *i.e.*, not affected by its weight changes — may often be centered exactly prior to learning. In particular, for any given training set the network's inputs can be centered in this fashion. Even in online settings where the training set is not known in advance, it is sometimes possible to perform such *extrinsic* centering based upon prior knowledge of the training data: instead of a time series $\mathbf{x}(t)$ one might for instance present the temporal difference signal $\mathbf{x}(t) - \mathbf{x}(t-1)$ as input to the network, which will be centered if $\mathbf{x}(t)$ is stationary.

3.2 Adaptive Methods

Online. When learning online, the immediate environment of a single weight within a multi-layer network is highly non-stationary, due to the simultaneous adaptation of other weights, if not due to the learning task itself. A uniquely defined average of some signal $\mathbf{x}(t)$ to be centered is therefore not available online, and we must make do with *running averages* — smoothed versions of the signal itself. A popular smoother is the *exponential trace*

$$\bar{\mathbf{x}}(t+1) \,=\, \alpha\,\bar{\mathbf{x}}(t) \,+\, (1-\alpha)\,\mathbf{x}(t)\,, \tag{11}$$

which has the advantage of being *history-free* and *causal*, *i.e.*, requiring neither past nor future values of $\mathbf{x}$ for the present update. The free parameter α (with $0 \leq \alpha \leq 1$) determines the time scale of averaging. Its choice is not trivial: if it is too small, $\bar{\mathbf{x}}$ will be too noisy; if it is too large, the average will lag too far behind the (drifting) signal.

Note that the computational cost of centering by this method is proportional to the number of nodes in the network. In densely connected networks, this is dwarfed by the number of weights, so that the propagation of activities and error signals through these weights dominates the computation. The cost of online centering will therefore make itself felt in small or sparsely connected networks only.

Two-pass batch. A simple way to implement exact centering in a batch learning context is to perform *two* passes through the training set for each weight update: the first to calculate the required averages, the second to compute the resulting weight changes. This obviously may increase the computational cost of network training by almost a factor of two. For relatively small networks and training sets, the activity and error for each node and pattern can be stored during the first pass, so that the second pass only consists of the weight update (Equation 4). Where this is not possible, a feedforward-only first pass (Equation 1) is sufficient to compute average activities and slopes; error centering may then be implemented via one of the other methods described here.

Previous batch. To avoid the computational overhead of a two-pass method, one can use the averages collected over the *previous* batch in the computation of weight changes for the current batch. This approximation assumes that the averages involved do not change too much from batch to batch; this may result in stability problems in conjunction with very high learning rates. Computationally this method is quite attractive in that it is cheaper still than the online technique described above. When mini-batches are used for training, both approaches can be combined profitably by centering with an exponential trace over mini-batch averages.

Single-pass batch. It is possible to perform exact centering in just a *single* pass through a batch of training patterns. This is done by expanding the triple product of the fully centered batch weight update (*cf.* Equation 7). Using f'_j as a shorthand for $f'_j(y_j)$, we have

$$\begin{aligned}
\Delta w_{ij} &\propto \left\langle (x_i - \langle x_i \rangle)(\gamma_j - \langle \gamma_j \rangle)(f'_j - \langle f'_j \rangle) \right\rangle \\
&= \left\langle x_i \gamma_j f'_j \right\rangle - \left\langle \langle x_i \rangle \gamma_j f'_j \right\rangle - \left\langle x_i \langle \gamma_j \rangle f'_j \right\rangle - \left\langle x_i \gamma_j \langle f'_j \rangle \right\rangle \\
&\quad + \left\langle \langle x_i \rangle \langle \gamma_j \rangle f'_j \right\rangle + \left\langle \langle x_i \rangle \gamma_j \langle f'_j \rangle \right\rangle + \left\langle x_i \langle \gamma_j \rangle \langle f'_j \rangle \right\rangle - \left\langle \langle x_i \rangle \langle \gamma_j \rangle \langle f'_j \rangle \right\rangle \\
&= \left\langle x_i \gamma_j f'_j \right\rangle - \langle x_i \rangle \left\langle \gamma_j f'_j \right\rangle - \langle \gamma_j \rangle \left\langle x_i f'_j \right\rangle - \langle x_i \gamma_j \rangle \left\langle f'_j \right\rangle + 2 \langle x_i \rangle \langle \gamma_j \rangle \left\langle f'_j \right\rangle \qquad (12)
\end{aligned}$$

In addition to the ordinary (uncentered) batch weight update term $\left\langle x_i \gamma_j f'_j \right\rangle$ and the individual averages $\langle x_i \rangle$, $\langle \gamma_j \rangle$, and $\left\langle f'_j \right\rangle$, the single-pass centered update (12) thus also requires collection of the sub-products $\langle x_i \gamma_j \rangle$, $\left\langle x_i f'_j \right\rangle$, and $\left\langle \gamma_j f'_j \right\rangle$. Due to the extra computation involved, the single-pass batch update is not necessarily more efficient than a two-pass method. It is simplified considerably, however, when not all factors are involved — for instance, when activities have already been centered *a priori* so that $\langle x_i \rangle \approx 0$.

Note that the expansion technique shown here may be used to derive an exact single-pass batch method for *any* weight update that involves the addition (or subtraction) of some quantity that must be computed from the entire batch of training patterns. This includes algorithms such as BCM learning [8, 9] and binary information gain optimization [10].

4 Empirical Results

While activity centering has long been part of backpropagation lore, and empirical results for error centering have been reported previously [2], slope centering is being proposed for the first time here. It is thus too early to assess its general applicability or utility; here we present a number of experiments designed to show the typical effect that centering has on speed and reliability of convergence as well as generalization performance in feedforward neural networks trained by accelerated backpropagation methods.

The next section describes the general setup and acceleration techniques used in all our experiments. Subsequent sections then present our respective results for two well-known benchmarks: the toy problem of symmetry detection in binary patterns, and a difficult vowel recognition task.

4.1 Setup of Experiments

Benchmark design. For each benchmark task we performed a number of *experiments* to compare performance with *vs.* without various forms of centering. Each experiment consisted of 100 *runs* starting from different initial weights but identical in all other respects. For each run, networks were initialized with random weights from a zero-mean Gaussian distribution with standard deviation 0.3. All experiments were given the same sequence of random numbers for their 100 weight initializations; the seed for this sequence was picked only after the design of the benchmark had been finalized.

Training modality. In order to make the results as direct an assessment of centering as possible, training was done in batch mode so as to avoid the additional free parameters (*e.g.*, smoothing time constants) required by online methods. Where not done *a priori*, centering was then implemented with the exact two-pass batch method. In addition, we always updated the hidden-to-output weights of the network *before* backpropagating error through them. This is known to sometimes improve convergence behavior [11], and we have found it to increase stability at the large step sizes we desire.

Competitive controls. The ordinary backpropagation (plain gradient descent) algorithm has many known defects, and a large number of acceleration techniques has been proposed for it. We informally tested a number of such techniques, then picked the combination that achieved the fastest reliable convergence. This combination — *vario-η* and *bold driver* — was then used for all experiments reported here. Thus any performance advantage for centering reported thereafter has been realized *on top of* a state-of-the-art accelerated gradient method as control.

Vario-η [12, 13, page 48]. This interesting technique sets the local learning rate for each weight inversely proportional to the standard deviation of its stochastic gradient. The weight change thus becomes

$$\Delta w_{ij} = \frac{-\eta \, g_{ij}}{\varrho + \sigma(g_{ij})}, \text{ where } g_{ij} \equiv \frac{\partial E}{\partial w_{ij}} \text{ and } \sigma(u) \equiv \sqrt{\langle u^2 \rangle - \langle u \rangle^2}, \qquad (13)$$

with the small positive constant ϱ preventing division by near-zero values. Vario-η can be used in both batch and online modes, and is quite effective in that it not only performs gradient normalization, but also adapts step sizes to the level of noise in the local gradient signal.

We used vario-η for all experiments reported here, with $\varrho = 0.1$. In a batch implementation this leaves only one free parameter to be determined: the global learning rate η.

Bold driver [14, 15, 16, 17]. This algorithm for adapting the global learning rate η is simple and effective, but only works for batch learning. Starting from some initial value, η is increased by a certain factor after each batch in which the error did not increase by more than a very small constant ε (required for numerical stability). Whenever the error rises by more than ε, however, the last weight change is undone, and η decreased sharply.

All experiments reported here were performed using bold driver with a learning rate increment of 2%, a decrement of 50%, and $\varepsilon = 10^{-10}$. These values were found to provide fast, reliable convergence across all experiments. Due to the amount of recomputation they require, we do count the "failed" epochs (whose weight changes are subsequently undone) in our performance figures.

4.2 Symmetry Detection Problem

In our first benchmark, a fully connected feedforward network with 8 inputs, 8 hidden units and a single output is to learn the symmetry detection task: given an 8-bit binary pattern at the input, it is to signal at the output whether the pattern is symmetric about its middle axis (target = 1) or not (target = 0). This is admittedly a toy problem, although not a trivial one.

Since the target is binary, we used a logistic output unit and cross-entropy loss function. For each run the network was trained on all 256 possible patterns until the root-mean-square error of its output over the batch fell below 0.01. We recorded the number of epochs required to reach this criterion, but did not test for generalization ability on this task.

Error and activity centering. In our first set of experiments we examined the separate and combined effect of centering the network's activity and/or error signals. For convenience, activity centering was performed *a priori* by using -1 and 1 as input levels, and the hyperbolic tangent (tanh) as activation function for hidden units. The off-center control experiments were done with 0 and 1 as input levels and the logistic activation function $f(y) = 1/(1 + e^{-y})$. Note that all differences between the tanh and logistic nonlinearities are eliminated by the vario-η algorithm, *except* for the eccentricity of their respective outputs.

Results. Table 1 shows that centering either activity or error signals produced an approximate 7-fold increase in convergence speed. In no instance was a run

error signals:	conventional				centered
activities:	**mean ± st.d.** quartiles	direct comparison: # of faster runs			**mean ± st.d.** quartiles
off-center (0/1)	**669 ± 308**		0 − 100		**97.5 ± 21.8**
	453/580/852	0	0 35	7	82/95.5/109
		|	×	|	
centered (-1/1)	**93.1 ± 46.7**	100	63 100	93	**65.4 ± 15.9**
	67.5/79.5/94		14 − 84		57/62/70

Table 1. The effect of centering activities and/or error signals on the symmetry detection task without shortcuts. Reported are the empirical mean, standard deviation, and $25^{th}/50^{th}/75^{th}$ percentile (rounded to three significant digits) of the number of epochs required to converge to criterion. Also shown is the result of directly comparing runs with identical random seeds. The number of runs in each comparison may sum to less than 100 due to ties.

that used one (or both) of these centering methods slower than the corresponding control without centering. The similar magnitude of the speed-up suggests that it may be due to the improved conditioning of the Hessian achieved by centering either errors or activities (see Section 5). Note, however, that activity centering beat error centering almost 2/3 of the time in the direct comparison.

On the other hand, error centering appeared to improve the *reliability* of convergence: it cut the convergence time's coefficient of variation (the ratio between its standard deviation and mean, henceforth: c.v.) in half while activity centering left it unchanged. We speculate that this may be the beneficial effect of centering on the *backpropagated* error, which does not occur for activity centering.

Finally, a further speedup of 50% (while maintaining the lower c.v.) occurred when both activity and error signals were centered. This may be attributed to the fact that our centering of hidden unit activity *by design* (*cf.* Section 3) was only approximate. To assess the significance of these effects, note that since the data was collected over 100 runs, the standard error of the reported mean time to convergence is $1/\sqrt{100} = 1/10$ its reported standard deviation.

Shortcuts and slope centering. In the second set of experiments we left both activity and error signals centered, and examined the separate and combined effect of adding shortcuts and/or slope centering. Note that since the complement of a symmetric bit pattern is also symmetric, the symmetry detection task has *no* linear component at all — we would therefore expect shortcuts to be of minimal benefit here.

Results. Table 2 shows that indeed adding shortcuts alone was not beneficial — in fact it slowed down convergence in over 80% of the cases, and significantly

slopes:		conventional				centered
topology:		**mean $\pm$ st.d.** quartiles	direct comparison: # of faster runs			**mean $\pm$ st.d.** quartiles
short-cuts?	no	**65.4 $\pm$ 15.9** 57/62/70	81	52 − 48 0 × 61	4	* **51.6 $\pm$ 16.2** 43/64.5/∞
	yes	**90.4 $\pm$ 31.1** 69.5/80/102	17	39 99 0 − 100	95	**33.1 $\pm$ 8.6** 28/31/35

* Mean and standard deviation exclude 34 runs which did not converge.

Table 2. The effect of centering slopes and/or adding shortcuts on the symmetry detection task with centered activity and error signals. Results are shown in the same manner as in Table 1.

increased the c.v. Subsequent addition of slope centering, however, brought about an almost 3-fold increase in learning speed, and restored the original c.v. of about 1/4. When used together, slope centering and shortcuts never increased convergence time, and on average cut it in half. By contrast, slope centering *without* shortcuts failed to converge at all about 1/3 of the time. This may come as a surprise, considering that the given task had no linear component. However, consider the following:

Need for shortcuts. Due to the monotonicity of their nonlinear transfer function, hidden units always carry some linear moment, in the sense of a positive correlation between their net input and output. In the absence of shortcuts, the hidden units must arrange themselves so that their linear moments together match the overall linear component of the task (here: zero). This adaptation process is normally driven by the linear component of the error — which slope centering removes.

The remaining nonlinear error signals can still jostle the hidden units into an overall solution, but such an indirect process is bound to be unreliable: as it literally removes slope from the error surface, slope centering creates numerous local minima. Shortcut weights turn these local minima into global ones by modeling the missing (linear) component of the gradient, thereby freeing the hidden units from any responsibility to do so.

In summary, while a network without shortcuts trained with slope centering may converge to a solution, the addition of shortcut weights is necessary to ensure that slope centering will not be detrimental to the learning process. Conversely, slope centering can prevent shortcuts from acting as redundant "detractors" that impede learning instead of assisting it. These two techniques should therefore always be used in conjunction.

4.3 Vowel Recognition Problem

Our positive experiences with centering on the symmetry detection task immediately raise two further questions: 1) will these results transfer to more challenging, realistic problems, and 2) is the gain in learning speed — as often happens — bought at the expense of generalization ability? In order to address these questions, we conducted further experiments with the speaker-independent vowel recognition data due to Deterding [18], a popular benchmark for which good generalization performance is rather difficult to achieve.

The task. The network's task is to recognize the eleven steady-state vowels of British English in a speaker-independent fashion, given 10 spectral features (specifically: LPC-derived log area ratios) of the speech signal. The data consists of 990 patterns to be classified: 6 instances for each of the 11 vowels spoken by each of 15 speakers. We follow the convention of splitting it into a training set containing the data from the first 8 (4 male, 4 female) speakers, and a test set containing those of the remaining 7 (4 male, 3 female). Note that there is no separate validation set available.

Prior work. Robinson [19] pioneered the use of Deterding's data as a benchmark by comparing the performance of a number of neural network architectures on it. Interestingly, none of his methods could outperform the primitive single nearest neighbor approach (which misclassifies 44% of test patterns), thus posing a challenge to the pattern recognition community. Trained on the task as formulated above, conventional backpropagation networks in fact appear to reach their limits at error rates of around 42% [20, 21], while an adaptive nearest neighbor technique can achieve 38% [22]. Flake [23] reports comparably favorable results for RBF networks as well as his own hybrid architectures. Even better performance can be obtained by using speaker sex/identity information [24, 25], or by training a separate model for each vowel [26]. By combining these two approaches, a test set error of 23% has been reached [27], the lowest we are aware of to date.

Training and testing. We trained fully connected feedforward networks with 10 inputs, 22 hidden units, and 11 logistic output units by minimization of cross-entropy loss. The target was 1 for the output corresponding to the correct vowel, 0 for all others. Activity centering was done *a priori* by explicitly centering the inputs (separately for training and test set), and by using the tanh nonlinearity for hidden units. The uncentered control experiments used the original input data, and logistic activation functions.

The relatively small size of our networks enabled us to run all experiments out to 2 000 epochs of training. After each epoch, the network's generalization ability was measured in terms of its misclassification rate on the test set. For the purpose of testing, a maximum likelihood approach was adopted: the network's highest output for a given test pattern was taken to indicate its classification of that pattern.

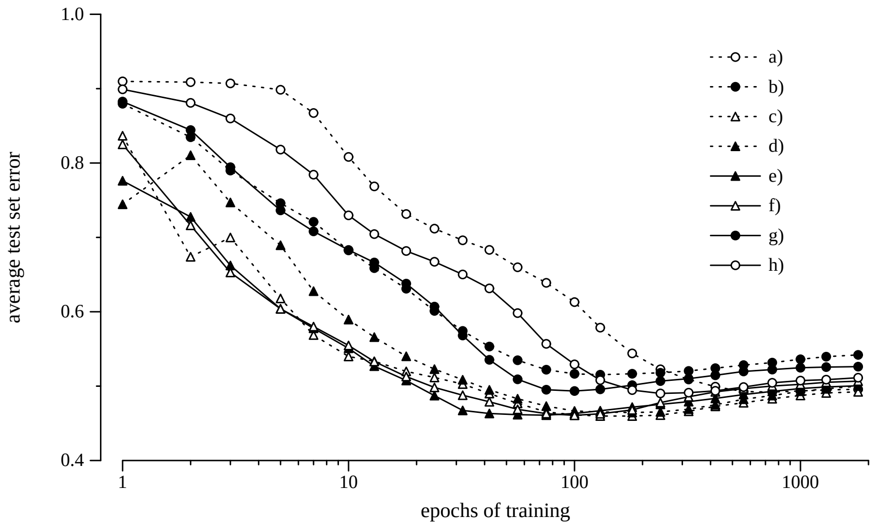

Fig. 1. Evolution of the average test set error while learning the vowel recognition task with activity centering (triangular marks), error centering (filled marks), and/or slope centering with shortcut weights (solid lines), *vs.* their uncentered controls. Experiments are denoted a)–h) as in Table 3.

First results. Figure 1 shows how the average test set error (over 100 runs) evolved during training in each of the 8 experiments we performed for this benchmark. For all curves, error bars were at most the size of the marks shown along the curve, and have therefore been omitted for clarity. Following our experience on the symmetry detection task, shortcuts and slope centering were always used in conjunction whereas activity and error centering were examined independently. The following effects can be discerned:

1. All experiments with activity centering (triangular marks) clearly outperformed all experiments without it (circular marks) in both average convergence speed and minimum average test set error.
2. All experiments with shortcuts and slope centering (solid lines) outperformed the corresponding experiment without them (dashed lines).
3. With one notable exception (experiment d), error centering (filled marks) sped up convergence significantly. Its effect was greatest in the experiments without activity centering.
4. The best experiment in terms of both convergence speed and minimum average test set error was e), the fully centered one; the worst was a), the fully conventional one.

The qualitative picture that emerges is that centering appears to significantly speed up convergence without adversely affecting the trained network's generalization ability. We will now attempt to quantify this finding.

Quantifying the effect. Since the curves in Figure 1 are in fact superpositions of 100 nonlinear curves each, they are ill-suited to quantitative analysis: value and location of the minimum average test set error do not tell us anything about the distribution of such minima across individual runs — not even their average value or location. In order to obtain such quantitative results, we need to identify an appropriate minimum in test set error for each run. This will allow us to directly compare runs with identical initial weights across experiments, as well as to characterize the distribution of minima within each experiment by aggregate statistics (*e.g.*, mean, standard deviation, quartiles) for both the minimum test set error, and the time taken to reach it.

A fair and consistent strategy to identify minima suitable for the quantitative comparisons we have in mind is not trivial to design. Individual runs may have multiple minima in test set error, or none at all. If we were to just use the global minimum over the duration of the run (2 000 epochs), we would not be able to distinguish a fast method which makes some insignificant improvement to a long-found minimum late in the run from a slow method which takes that long to reach its first minimum. Given that we are concerned with both the quality of generalization performance and the speed with which it is achieved, a greedy strategy for picking appropriate minima is indicated.

Identification of minima. We follow the evolution of test set error over the course of each run, noting new minima as we encounter them. If the best value found so far is not improved upon within a certain period of time, we pick it as *the* minimum of that run for the purpose of quantitative analysis. The appropriate length of waiting period before giving up on further improvement is a difficult issue [28]. For a fair comparison between faster and slower optimization methods, it should be proportional to the time it took to reach the minimum in question: a slow run then has correspondingly more time to improve its solution than a fast one.

Unfortunately this approach fails if a minimum of test set error occurs during the initial transient, within the first few epochs of training: the waiting period would then be too short, causing us to give up prematurely. On the other hand, we cannot wait longer than the overall duration of the run. We therefore stop looking for further improvement in a run after $\min(2\,000,\ 2\epsilon + 100)$ epochs, where ϵ records when the network first achieved the lowest test set error seen so far in that run. Only 9 out of the 800 runs reported here expired at the upper limit of 2 000 epochs, so we are confident that its imposition did not significantly skew our results.

Test set used as validation set. Note that since we use the test set to determine at which point to compare performance, we have effectively appropriated it as a validation set. The minimum test set errors reported below are therefore *not* unbiased estimators for the network's ability to generalize to novel speakers, and should not be compared to proper measurements of this ability (for which the test set must not affect the training procedure in any way). Nonetheless, let

experiment	features:				performance measure:											
	centering			shortcuts	minimum test set error						epochs required					
	activ.	error	slope		**mean ± st.d.** quartiles	dir. comparison: # of better runs					**mean ± st.d.** quartiles	dir. comparison: # of faster runs				
a)					**48.0 ± 3.6**						**554 ± 321**					
					45.7/47.3/50.0	67	17	13	19	37	365/486/691	3	10	4	1	14
b)		√			**49.1 ± 2.9**	31					**125 ± 82**	97				
					47.0/49.6/50.9	10					67.5/104/163	51				
c)	√				**43.9 ± 2.5**		82				**156 ± 110**		90			
					42.3/43.9/46.0		51				75/137/215		47			
d)	√	√			**44.3 ± 2.3**	89	46	84			**158 ± 141**	48	52	96		
					42.9/44.2/45.9		49	65			72/124/186		21	85		
e)	√	√	√	√	**44.2 ± 2.5**		49		80		**72.4 ± 55.5**		78		99	
					42.3/44.4/46.3	70	47		68		37.5/51.5/81	76	75		92	
f)	√		√	√	**44.2 ± 2.8**		51				**113 ± 64**		24			
					42.3/44.4/46.1		68				69.5/97.5/148		88			
g)		√	√	√	**46.8 ± 3.7**	27					**126 ± 139**	22				
					44.0/47.0/48.9	43					64/94/138	84				
h)			√	√	**46.5 ± 3.1**	56	31	29	30	61	**270 ± 164**	15	12	15	8	86
					44.5/46.8/48.5						162/235/316					

Table 3. Minimum test set error (misclassification rate in %), and the number of epochs required to reach it, for the vowel recognition task. Except for the different layout, results are reported in the same manner as in Tables 1 and 2. Due to space limitations, only selected pairs of experiments are compared directly.

us not forget that the lowest test set error does measure the network's generalization ability in a consistent fashion after all: even though these scores are all biased to favor a particular set of novel speakers (the test set), by no means does this render their comparison *against each other* insignificant.

Overview of results. Table 3 summarizes quantitative results obtained in this fashion for the vowel recognition problem. To assess their significance, recall that the standard error in the mean of a performance measure reported here is $1/\sqrt{100} = 1/10$ of its reported standard deviation. Figure 2 depicts the same data (except for the direct comparisons) graphically in form of cumulative histograms for the minimum test set error and the number of epochs required to reach it.

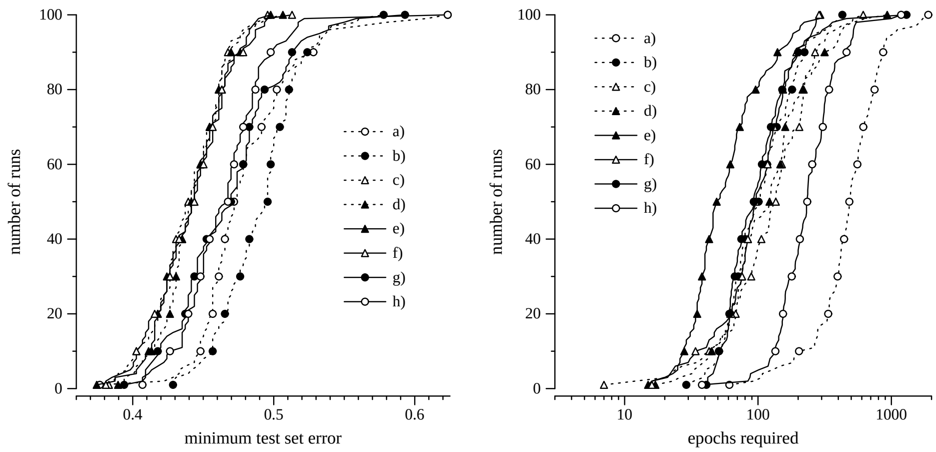

Fig. 2. Cumulative histograms for the minimum test set error (left), and the number of epochs required to reach it (right), for the vowel recognition task. Curves are labelled as in Figure 1, and marked every 10^{th} percentile.

The results generally confirm the trends observed in Figure 1. Runs in the fully centered experiment e) clearly converged most rapidly — and to test set errors that were among the best. Compared to the conventional setup a), full centering converged almost 8 times faster on average while generalizing better 80% of the time.

Generalization performance. Average misclassification rates on the test set ranged from 44% to 49%, which we consider a fair result given our comparatively small networks. They cluster into three groups: networks with activity centering achieved around 44%, the two others with shortcuts and slope centering came in under 47%, while the remaining two only reached 48–49%. The cumulative histogram (Figure 2, left) shows that all activity-centered networks had an almost identical distribution of minimum test set errors.

Note that centering the inputs changes the task, and that the addition of shortcuts changes the network topology. It is possible that this — rather than centering *per se* — accounts for their beneficial effect on generalization. Error centering was the one feature in our experiments that changed the dynamics of learning exclusively. Its addition appeared to slightly *worsen* generalization, particularly in the absence of other forms of centering. This could be caused by a reduction (due to centering) of the effective number of parameters in what is already a rather small model. Such an effect should not overly concern us: one could easily recoup the lost degrees of freedom by slightly increasing the number of hidden units for centered networks.

Convergence speed. All three forms of centering examined here clearly sped up convergence, both individually and in combination. A slight anomaly appeared in that the addition of error centering in going from experiment c) to d) had no significant effect on the average number of epochs required. A look at the cumulative histogram (Figure 2, right) reveals that while experiment d) is ahead between the 20th and 80th percentile, c) had fewer unusually slow runs than d), and a few exceptionally fast ones.

With the other forms of centering in place, the addition of error centering was unequivocally beneficial: average convergence time decreased from 113 epochs in f) to 72.4 epochs in e). The histogram shows that the fully centered e) is far ahead of the competition through almost the entire percentile range.

Finally, it is interesting to note that the addition of shortcuts and slope centering, both on their own and to a network with activity and error centering, roughly doubled the convergence speed — the same magnitude of effect as observed on the symmetry detection task.

5 Discussion

The preceding section has shown that centering can indeed have beneficial effects on the learning speed and generalization ability of a neural network. Why is this so? In what follows, we offer an explanation from three (partially overlapping) perspectives, considering in turn the effect of centering on the condition number of the Hessian, the level of noise in the local gradient, and the credit assignment between different parts of the network.

Conditioning the Hessian. It is well known that the minimal convergence time for first-order gradient descent on quadratic error surfaces is inversely related to the condition number of the Hessian matrix, *i.e.*, the ratio between its largest and its smallest eigenvalue. A common strategy for accelerating gradient descent is therefore to seek to improve the condition number of the Hessian.

For a single linear node $y = \mathbf{w}^T\mathbf{x}$ with squared loss function, the Hessian is simply the covariance matrix of the inputs: $H = \left\langle \mathbf{x}\,\mathbf{x}^T \right\rangle$. Its largest eigenvalue is typically caused by the d.c. component of $\mathbf{x}$ [4]. Centering the inputs removes that eigenvalue, thus conditioning the Hessian and permitting larger step sizes. For batch learning, error centering has exactly the same effect on the local weight update:

$$\Delta\mathbf{w} \;\propto\; \langle(\delta - \langle\delta\rangle)\,\mathbf{x}\rangle \;=\; \langle\delta\,\mathbf{x}\rangle - \langle\delta\rangle\,\langle\mathbf{x}\rangle \;=\; \langle\delta\,(\mathbf{x} - \langle\mathbf{x}\rangle)\rangle \qquad (14)$$

Error centering does go further than activity centering, however, in that it also affects the error backpropagated to anterior nodes. Moreover, Equation 14 does not hold for online learning, where the gradient is noisy.

Noise reduction. It can be shown that centering improves the signal-to-noise ratio of the local gradient. Omitting the slope factor for the sake of simplicity, consider the noisy weight update

$$\Delta w_{ij} \propto (\delta_j + \phi)(x_i + \xi) = \delta_j x_i + \xi\delta_j + \phi x_i + \phi\xi \tag{15}$$

where ϕ and ξ are the noise terms, presumed to be zero-mean, and independent of activity, error, and each other. In the expansion on the right-hand side, the first term is the desired (noise-free) weight update while the others represent noise that contaminates it. While the last (pure noise) term cannot be helped, we can reduce the variance of the two mixed terms by centering δ_j and x_i so as to minimize $\langle \delta_j^2 \rangle$ and $\langle x_i^2 \rangle$, respectively.

One might of course contend that in doing so, we are also shrinking the signal $\delta_j x_i$, so that in terms of the signal-to-noise ratio we are no better — in fact, worse — off than before. This cuts right to the heart of the matter, for centering rests upon the notion that the error signal relevant to a non-bias, non-shortcut weight *is* the fully centered weight update, and that any d.c. components in $\delta_j x_i$ should therefore also be regarded as a form of noise. This presumption can of course be maintained only because we do have bias and shortcut weights to address the error components that centering removes.

Improved credit assignment. From the perspective of a network that has these additional parameters, then, centering is a way to improve the assignment of responsibility for the network's errors: constant errors are shunted to the bias weights, linear errors to the shortcut weights, and the remainder of the network is bothered only with those parts of the error signal that actually require a nonlinearity. Centering thus views hidden units as a scarce resource that should only be called upon where necessary. Given the computational complications that arise in the training of nonlinear nodes, we submit that this is an appropriate and productive viewpoint.

Future work. While the results reported here are quite promising, more experiments are required to assess the general applicability and effectiveness of centering. For feedforward networks, we would like to explore the use of centering with multiple hidden layers, stochastic (online) gradient descent, and for function approximation (rather than classification) problems. The centering approach *per se*, however, is rather more general than that, and so further ahead we anticipate its application to a range of more sophisticated network architectures, learning algorithms, and problem domains.

Acknowledgments

I would like to thank Klaus-Robert Müller, Jenny Orr, Jürgen Schmidhuber, Marco Wiering, and Rafał Sałustowicz for their helpful comments. This work was supported by the Swiss National Science Foundation under grant numbers 2100–045700.95/1 and 2000–052678.97/1.

References

1. Genevieve B. Orr and Klaus-Robert Müller, editors. *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1998.
2. Nicol N. Schraudolph and Terrence J. Sejnowski. Tempering backpropagation networks: Not all weights are created equal. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 563–569. The MIT Press, Cambridge, MA, 1996.
3. Bernard Widrow, John M. McCool, Michael G. Larimore, and C. Richard Johnson, Jr. Stationary and nonstationary learning characteristics of the LMS adaptive filter. *Proceedings of the IEEE*, 64(8):1151–1162, 1976.
4. Yann LeCun, Ido Kanter, and Sara A. Solla. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, 66(18):2396–2399, 1991.
5. Terrence J. Sejnowski. Storing covariance with nonlinearly interacting neurons. *Journal of Mathematical Biology*, 4:303–321, 1977.
6. Nicol N. Schraudolph. Slope centering: Making shortcut weights effective. In Lars Niklasson, Mikael Bodén, and Tom Ziemke, editors, *Proceedings of the 8th International Conference on Artificial Neural Networks*, Perspectives in Neural Computing, pages 523–528, Skövde, Sweden, 1998. Springer Verlag, Berlin. ◁ ftp://ftp.idsia.ch/pub/nic/slope.ps.gz ▷.
7. Patrick van der Smagt and Gerd Hirzinger. Solving the ill-conditioning in neural network learning. In *Neural Networks: Tricks of the Trade* [1], pages 193–206.
8. E.L. Bienenstock, L.N. Cooper, and P.W. Munro. Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2, 1982. Reprinted in [29].
9. N. Intrator. Feature extraction using an unsupervised neural network. *Neural Computation*, 4(1):98–107, 1992.
10. Nicol N. Schraudolph and Terrence J. Sejnowski. Unsupervised discrimination of clustered data via optimization of binary information gain. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 499–506. Morgan Kaufmann, San Mateo, CA, 1993.
11. Samir Shah, Francesco Palmieri, and Michael Datum. Optimal filtering algorithms for fast learning in feedforward neural networks. *Neural Networks*, 5:779–787, 1992.
12. Ralph Neuneier and Hans Georg Zimmermann. How to train neural networks. In *Neural Networks: Tricks of the Trade* [1], pages 373–423.
13. Hans Georg Zimmermann. Neuronale Netze als Entscheidungskalkül. In Heinz Rehkugler and Hans Georg Zimmermann, editors, *Neuronale Netze in der Ökonomie: Grundlagen und finanzwirtschaftliche Anwendungen*, pages 1–87. Vahlen Verlag, Munich, 1994.
14. A. Lapedes and R. Farber. A self-optimizing, nonsymmetrical neural net for content addressable memory and pattern recognition. *Physica*, D 22:247–259, 1986.
15. T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon. Accelerating the convergence of the back-propagation method. *Biological Cybernetics*, 59:257–263, 1988.
16. Roberto Battiti. Accelerated back-propagation learning: Two optimization methods. *Complex Systems*, 3:331–342, 1989.

17. R. Battiti. First- and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation*, 4(2):141–166, 1992.
18. David H. Deterding. *Speaker Normalisation for Automatic Speech Recognition.* PhD thesis, University of Cambridge, 1989.
19. Anthony J. Robinson. *Dynamic Error Propagation Networks.* PhD thesis, University of Cambridge, 1989.
20. Michael Finke and Klaus-Robert Müller. Estimating a-posteriori probabilities using stochastic network models. In Michael C. Mozer, Paul Smolensky, David S. Touretzky, Jeffrey L. Elman, and Andreas S. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School,*, Boulder, CO, 1994. Lawrence Erlbaum Associates, Hillsdale, NJ.
21. Sepp Hochreiter and Jürgen Schmidhuber. Feature extraction through LOCOCODE. To appear in *Neural Computation*, 1998.
22. Trevor J. Hastie and Robert J. Tibshirani. Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6):607–616, 1996.
23. Gary William Flake. Square unit augmented, radially extended, multilayer perceptrons. In *Neural Networks: Tricks of the Trade* [1], pages 145–163.
24. Peter D. Turney. Exploiting context when learning to classify. In *Proceedings of the European Conference on Machine Learning*, pages 402–407, 1993.
25. Peter D. Turney. Robust classification with context-sensitive features. In *Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 268–276, 1993.
26. Michael Herrmann. On the merits of topography in neural maps. In Teuvo Kohonen, editor, *Proceedings of the Workshop on Self-Organizing Maps*, pages 112–117. Helsinki University of Technology, 1997.
27. Joshua B. Tenenbaum and William T. Freeman. Separating style and content. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 662–668. The MIT Press, Cambridge, MA, 1997.
28. Lutz Prechelt. Early stopping — but when? In *Neural Networks: Tricks of the Trade* [1], pages 55–69.
29. J.A. Anderson and E. Rosenfeld, editors. *Neurocomputing: Foundations of Research.* MIT Press, Cambridge, 1988.

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