In: Neural Networks: Tricks of the Trade, G. Orr and K.-R. Müller (eds), Lecture Notes in Computer Science 1524, Springer-Verlag, pp. 193–206, 1998

Solving the Ill-Conditioning in Neural Network Learning

Patrick van der Smagt and Gerd Hirzinger

German Aerospace Research Establishment, Institute of Robotics and System Dynamics, P. O. Box 1116, D–82230 Wessling, GERMANY, Phone: +49 8153 281152, fax: +49 8153 281134, email: smagt@dlr.de

Abstract. In this paper we investigate the feed-forward learning problem. The well-known ill-conditioning which is present in most feed-forward learning problems is shown to be the result of the structure of the network. Also, the well-known problem that weights between 'higher' layers in the network have to settle before 'lower' weights can converge is addressed. We present a solution to these problems by modifying the structure of the network through the addition of linear connections which carry shared weights. We call the new network structure the *linearly augmented feed-forward network*, and it is shown that the universal approximation theorems are still valid. Simulation experiments show the validity of the new method, and demonstrate that the new network is less sensitive to local minima and learns faster than the original network.

Keywords: conditioning, local minima, feed-forward network, learning problem, linearly augmented feed-forward network, conjugate gradient learning.

1 Introduction

One of the major problems with feed-forward network learning remains the accuracy and speed of the learning algorithms. Since the learning problem is a complex and highly nonlinear one [12,4], iterative learning procedures must be used to solve the optimisation problem [2,14]. A continuing desire to improve the behaviour of the learning algorithm has lead to many excellent optimisation algorithms which are especially tailored for feed-forward network learning.

However, an important problem is the particular form of the error function that represents the learning problem. It has long been noted [10,16] that the derivatives of the error function are usually ill-conditioned. This ill-conditioning is reflected in error landscapes which contain many saddle points and flat areas.

Although this problem can be solved by using stochastic learning methods (e.g., [9,1,13]), these methods require many learning iterations in order to find an optimum, and are therefore not suited for problems where fast learning is a requirement. We therefore remain focused on gradient-based learning methods.

Algorithms exist which attempt to find well-behaving minima [7], yet an important factor of the learning problem remains the structure of the feed-forward network.

In this chapter an explanation of the ill-conditioned learning problem is provided as well as a solution to alleviate this problem. Section 2 formally introduces the learning problem, and describes the problem of singularities in the learn matrices. In section 3 the cause of the singularities are analysed and an adapted learning rule is introduced which alleviates this problem. Section 4 discusses a few applications.

2 The Learning Process

With a neural network $\mathcal{N} : \Re^N \times \Re^n \rightarrow \Re^M$ we create an approximation to a set of p learning samples $\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_p, \mathbf{y}_p)\}$, with $\mathbf{x}_i \in \Re^N$ and $\mathbf{y}_i \in \Re^M$, for which holds that $\forall 1 \leq i \leq p : \mathcal{F}(\mathbf{x}_i) = \mathbf{y}_i$. The function $\mathcal{F} : \Re^N \rightarrow \Re^M$ is called the **model function**.

Let n be the number of free parameters W of the network. In this particular case we are interested in approximating the learning samples rather than the underlying function $\mathcal{F}$, or assume that the p learning samples are representative for $\mathcal{F}$.

The oth output of the function that is represented by the neural network can be written as

$$\mathcal{N}(\mathbf{x}, W)_o = \sum_h w_{ho} s \left(\sum_i w_{ih} x_i + \theta_h \right) + \theta_o \tag{1}$$

where $s(x)$ the transfer function, o indicates an output unit, h a hidden unit, i an input unit. The symbol w_{ho} indicates an element of W corresponding with the connection from hidden unit h to output unit o; w_{ih} is similarly used for a connection from input unit i to hidden unit h. Finally, θ is a bias weight and therefore an element of W. An exemplar feed-forward network with one input and output unit and two hidden units is depicted in figure 1.

The learning task consists of minimising an **approximation error**, which is usually defined as

$$E_{\mathcal{N}}(W) = \sum_{i=1}^{p} \|\mathcal{N}(\mathbf{x}_i, W) - \mathbf{y}_i\| \tag{2}$$

where for $\|\cdot\|$ we prefer to use the L_2 norm. We will leave the subscript $\mathcal{N}$ out when no confusion arises. $E(W)$ is (highly) nonlinear in W, such that iterative search techniques are required to find the W for which $E(W)$ is sufficiently small.

Finally we define the **residual pattern error**

$$e_{M(i-1)+j} = \|\mathcal{N}(\mathbf{x}_i, W)_j - \mathbf{y}_{ij}\|, \tag{3}$$

i.e., the error in the j'th output value for the i'th learning sample.

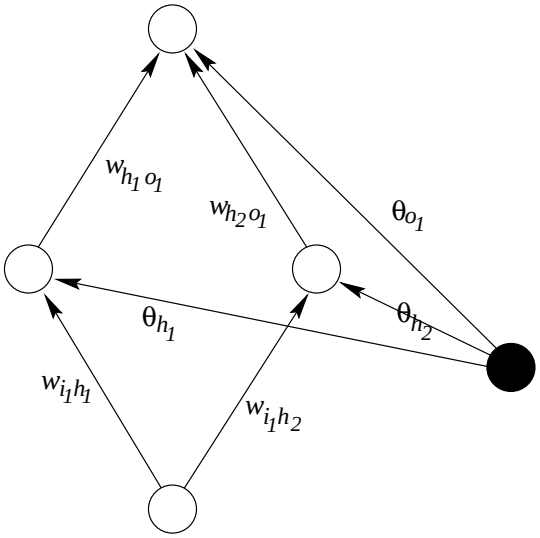

Fig. 1. An exemplar feed-forward neural network. The circles represent neurons; the black filled circle is a bias unit, and always carries an activation value of '1'.

2.1 Learning Methodology

Gradient based learning methods are characterised by considering low-order terms from the Taylor expansion to the approximation error,

$$E(W + W_0) = E(W_0) + \nabla E|_{W_0} W + W^T \nabla^2 E\big|_{W_0} W + \dots . \tag{4}$$

In most cases more than the second order term is neglected. We define

$$\tilde{E}_1(W + W_0) = E(W_0) + \nabla E|_{W_0} W \tag{5}$$

and

$$\tilde{E}_2(W + W_0) = \tilde{E}_1(W) + W^T \nabla^2 E\big|_{W_0} W \tag{6}$$

being the first-order and second-order approximation to E, respectively.

By locally considering the approximation error as a first- or second-order function of W, we can use several existing approximation methodologies to minimise E. When, according to the local second-order approximation information, E is minimised, the local information is updated and a second minimisation step is carried out. This process is repeated until a minimum is found.

Well-known minimisation methods are steepest descent (a variant of which is known as error back-propagation), conjugate gradient optimisation, Levenberg-Marquardt optimisation, variable metric methods, and (quasi-) Newton methods. Each of these methods has its advantages and disadvantages which are discussed elsewhere [14].

The optimisation methods work in principle as follows. By considering the second-order approximation to E, an optimum can be analytically found if ∇E

and $\nabla^2 E$ are known. After the optimum has been located, ∇E and $\nabla^2 E$ are recomputed using the local information, and the minimum is relocated. This process is repeated until a minimum is found.

Naturally, the success of this approach stands or falls with the form of the error function. If the error function is not too complex but smooth, and can be reasonably approximated by a quadratic function, the discussed optimisation methods are a reliable and fast way of finding minima. In feed-forward network training, however, the error functions appear to have a large number of flat areas where minimisation is a difficult task due to limited floating point accuracy.

2.2 Condition of the Learning Problem

The flat areas of the error function can be formalised as follows. We define the $(Mp) \times n$ Jacobian matrix as

$$J \equiv \begin{pmatrix} \nabla \mathbf{e}_1^T \\ \nabla \mathbf{e}_2^T \\ \vdots \\ \nabla \mathbf{e}_{Mp}^T \end{pmatrix} \quad \text{where} \quad \nabla \mathbf{e}_k \equiv \begin{pmatrix} \frac{\partial e_k}{\partial w_1} \\ \frac{\partial e_k}{\partial w_2} \\ \vdots \\ \frac{\partial e_k}{\partial w_n} \end{pmatrix} \tag{7}$$

such that we can write J as

$$J \equiv \begin{pmatrix} \frac{\partial e_1}{\partial w_1} & \frac{\partial e_1}{\partial w_2} & \cdots & \frac{\partial e_1}{\partial w_n} \\ \frac{\partial e_2}{\partial w_1} & \frac{\partial e_2}{\partial w_2} & \cdots & \frac{\partial e_2}{\partial w_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial e_{Mp}}{\partial w_1} & \frac{\partial e_{Mp}}{\partial w_2} & \cdots & \frac{\partial e_{Mp}}{\partial w_n} \end{pmatrix}. \tag{8}$$

In the learning process we thus encounter that $\nabla E = 2J^T \mathbf{e}$.

We furthermore define the Hessian $H = \nabla^2 E$, which is the matrix of second order derivatives of E. We are interested in the eigenvalues and eigenvectors of H. Since H is a positive semidefinite symmetric matrix close to a local minimum, its eigenvalues are all positive real numbers. When an eigenvalue is very small, the effect of moving in the direction of the corresponding eigenvector on the approximation error is very small. This means that, in that direction, the error function is (nearly) singular. The singularity of the error function can be expressed in the condition of H, which is defined as the quotient of its largest and its smallest eigenvalues.

As mentioned above, a bad condition of H may occur at minima or flat points in the error function $E(W)$. It appears that feed-forward learning tasks are generally characterised by having a singular or near-singular Hessian matrix. Although the said learning methods are mathematically not influenced by a badly conditioned Hessian, it does lead to inaccuracies due to the limited floating point accuracy of the digital computer.

2.3 What Causes the Singularities

Reference [10] lists a few cases in which the Hessian may become singular or near-singular. The listed reasons are associated with the ill character of the sigmoid transfer function. Typical for this function is the fact that $\lim_{x\to\infty} s(x) = c_+$ and $\lim_{x\to-\infty} s(x) = c_-$. Also, bad conditioning can be the result of uncentered data, and can also be alleviated [11].

Assuming that some neuron is in this 'saturated' state for all learning patterns, its input weights will have a delta equal to 0 (according to the back-propagation rule) such that these weights will never change. For each of the incoming weights of this neuron, this leads to a 0-row in H, and therefore singularity.

However, there is another important reason for singularity, which may especially occur after network initialisation. When a multi-layer feed-forward network has a small (e.g., less than 0.1) weight leaving from a hidden unit, the influence of the *weights that feed into this hidden unit* is significantly reduced. Therefore the $\partial e/\partial w_k$ will be close to 0, leading to near-null rows in J and a near-singular H.

We observe that this kind of singularity is very common and touches a characteristic problem in feed-forward network learning: The gradients in the lower-layer weights are influenced by the higher-layer weights. A related problem is the influence that the change of the weights between the hidden and output units have on the change of the input weights; when they rapidly and frequently change, as will be the case during the initial stages of learning, the lower weights will have nonsensical repeated perturbations.

2.4 Definition of Minimum

A minimum is said to be reached when the derivative of the error function is zero, i.e., $\partial E/\partial w_{1\leq k\leq n} = 0$. Since the gradient of the error function equals the column-sum of the Jacobian, a minimum has been reached when

$$\forall 1 \leq k \leq n : \sum_{i=1}^{Mp} \frac{\partial e_i}{\partial w_k} = 0, \tag{9}$$

i.e., when each of the columns adds up to 0. Eq. (9) defines the minimum for a batch-learning system: the gradient, when summed over all learning samples, should be 0. This also means, however, that it may occur that elements of a column-sum cancel each other out, even when not all elements of the Jacobian are 0. Differently put, it may occur that the gradient for some patterns are non-zero, whereas the gradients sum up to zero.

The optimal case is reached when *all* elements of the Jacobian are 0. This means, of course, that the residual error of each pattern is 0.

3 Local Minima are Caused by Back-Propagation

In this section we propose a new neural network structure which alleviates the above problems. In the standard back-propagation learning procedure, the gradient of the error function with respect to the weights is determined by computing the following steps for each learning pattern:

1. For each output unit o, compute the delta $\delta_o = y_o - a_o$ where a_o is the activation value for that unit, when a learning sample is presented.
2. Compute the weight derivative for the weights w_{ho} from the hidden to output units:
$$\Delta w_{ho} = \delta_o a_h$$
where a_h is the activation value for the hidden unit.
3. Compute the delta for the hidden unit:
$$\delta_h = \sum_o \delta_o w_{ho} s'(a_h).$$
4. Compute the weight derivative for the weights w_{ih} from the input to hidden units:
$$\Delta w_{ih} = \delta_h a_i = \sum_o \delta_o w_{ho} s'(a_h) a_i. \tag{10}$$

The gradient is then computed as the summation of the Δw's.

From (10) we can see that there are four cases when the gradient for a weight from an input to a hidden unit is negligible, such that the corresponding row and column in the Hessian are near-zero:

- When δ_o is small. This is correct, since that case means that the output of the network is close to its desired output.
- When w_{ho} is small. This is an undesired situation: A small weight from hidden to output unit paralyses weights from input to hidden units. This is especially important since the weight might have to change its value later.
- When $s'(a_h)$ is small; this occurs when the weight w_{ih} is large. Again, this saturation type of paralysis is undesired.
- Finally, when a_i is small. This is desired: When the input value is insignificant, it should have no influence on the output.

3.1 A New Neural Network Structure

In order to alleviate these problems, we propose a change to the learning system of (10) as follows:

$$\Delta w_{ih} = \sum_o \delta_o (w_{ho} s'[a_h] + c) a_i = \delta_h a_i + c \sum_o \delta_o a_i.$$

By adding a constant c to the middle term, we can solve both paralysis problems. In effect, an extra connection from each input unit to each output unit is created, with a weight value coupled to the weight from the input to hidden unit.

Although c can be made a learnable parameter, in the sequel we will assume $c = 1$. The o'th output of the neural network is now computed as

$$\mathcal{M}(\mathbf{x}, W)_o = \sum_h \left(w_{ho} s \left[\sum_i w_{ih} x_i + \theta_h \right] + \sum_i w_{ih} x_i \right) + \theta_o. \tag{11}$$

We call the new network the **linearly augmented feed-forward network**. The structure of this network is depicted in figure 2. Note the equivalence of $\mathcal{N}$

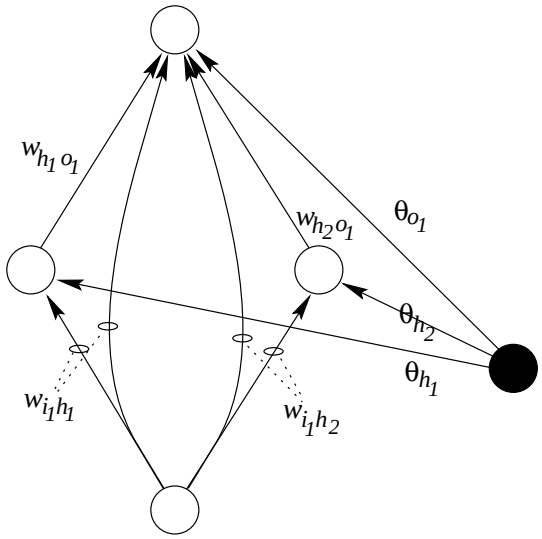

Fig. 2. An exemplar adapted feed-forward neural network.

and $\mathcal{M}$, viz.,

$$\mathcal{M}(\mathbf{x}, W)_o \equiv \mathcal{N}(\mathbf{x}, W)_o + \sum_h \sum_i w_{ih} x_i. \tag{12}$$

3.2 Influence on the Approximation Error *E*

Although the optimal W will be different for the networks $\mathcal{N}$ and $\mathcal{M}$, we can still compare the forms of the error functions $E_{\mathcal{N}}$ vs. $E_{\mathcal{M}}$. Using (2) and (11) we can compute the error in the approximation of a single learning sample $(\mathbf{x}, y)$ with N inputs, κ hidden units, and a single output:

$$\begin{aligned} E_{\mathcal{M}}(W)^2 &= (\mathcal{M}[\mathbf{x}, W] - y)^2 \\ &= \left(\mathcal{N}(\mathbf{x}, W) - y + \sum_i \sum_h w_{ih} x_i \right)^2 \end{aligned}$$

$$= E_{\mathcal{N}}(W)^2 + 2\sum_i \sum_h w_{ih} x_i (\mathcal{N}[\mathbf{x}, W] - y) + \left(\sum_i \sum_h w_{ih} x_i \right)^2 \quad (13)$$

The error for the network $\mathcal{M}$ differs from the error for $\mathcal{N}$ in two terms. When we consider $E_{\mathcal{M}}$ at those values of W where $E_{\mathcal{N}}$ is minimal, we can see that the difference between $E_{\mathcal{N}}$ and $E_{\mathcal{M}}$ consists of a normalisation term $\| \sum_h \mathbf{w}_h^T \mathbf{x} \|$; $\mathbf{w}_h$ is the vector of weights connecting the inputs to hidden unit h. This non-negative term will only be zero when the vectors $\mathbf{w}_1^T \mathbf{x}$, $\mathbf{w}_2^T \mathbf{x}$, ..., $\mathbf{w}_\kappa^T \mathbf{x}$ cancel each other out for each input vector $\mathbf{x}$ which is in the training set. In words: $E_{\mathcal{M}}(W)^2$ introduces a penalty for hidden units doing the same thing, thus making the network use its resources more efficiently.

3.3 $\mathcal{M}$ and the Universal Approximation Theorems

It has been shown in various publications [3,6,8] that the ordinary feed-forward neural network $\mathcal{N}$ can represent any Borel-measurable function with a single layer of hidden units which have sigmoidal or Gaussian activation functions in the hidden layer.

Theorem 1. *The network $\mathcal{M}$ can represent any Borel-measurable function with a single layer of hidden units which have sigmoidal or Gaussian activation functions in the hidden layer.*

Proof. We show that any network $\mathcal{N}$ can be written as a network $\mathcal{M}$; therefore, the class of networks $\mathcal{M}$ are universal approximators.

By using (1) and (11),

$$\begin{aligned}
\mathcal{N}(\mathbf{x}, W)_o &= \sum_{h=1}^{\kappa} w_{ho} s \left(\sum_i w_{ih} x_i + \theta_h \right) + \theta_o \\
&= \sum_{h=1}^{\kappa} \left(w_{ho} s \left[\sum_i w_{ih} x_i + \theta_h \right] + \sum_i w_{ih} x_i \right) + \theta_o - \sum_{h=1}^{\kappa} \sum_i w_{ih} x_i \\
&= \mathcal{M}(\mathbf{x}, W)_o + \sum_{l=\kappa+1}^{2\kappa} \left(0 \left[\sum_i -w_{i,l-\kappa} x_i + 0 \right] + \sum_i -w_{i,l-\kappa} x_i \right) + 0 \\
&= \mathcal{M}(\mathbf{x}, W)_o + \mathcal{M}(\mathbf{x}, V)_o
\end{aligned}$$

where V is a weight matrix such that the elements of V corresponding to the weights from hidden to output units are 0, and the other weights equal the negation of its W counterparts. Furthermore, bias weights are set to 0.

The sum $\mathcal{M}(\mathbf{x}, W)_o + \mathcal{M}(\mathbf{x}, V)_o$ represents two $\mathcal{M}$-networks, which can also be written as a single $\mathcal{M}(\mathbf{x}, W')$-network with the double amount of hidden units, where $W' = [WV]$.

Using the theorems from [3,6,8] the proof is complete.

Note that it is also possible to write each $\mathcal{M}$-network as an $\mathcal{N}$-network, by doubling the number of hidden units and using infinitesimal weights from the input units to these hidden units, and their multiplicative inverse for the weights from these hidden to the output units.

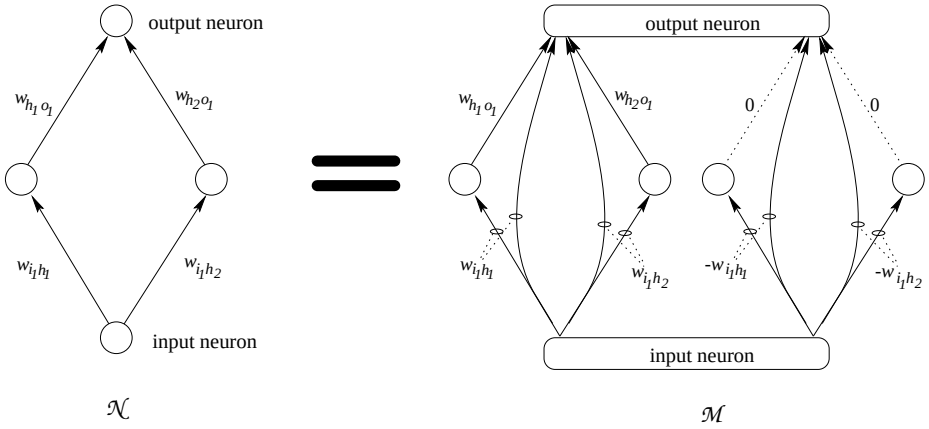

Fig. 3. Equivalence of an $\mathcal{N}$-network (left) and an $\mathcal{M}$-network (right). Note that bias units are left out for clarity.

Figure 3 shows the equivalence of an $\mathcal{N}$ and $\mathcal{M}$ network for the two-hidden unit case.

3.4 Example

As an example, we train a network with a single hidden unit, and no bias connections, to represent the learning samples $(1, 1)$ and $(2, 2)$. The hidden unit has a sigmoid activation function. The function that is computed by the network thus is $\mathcal{N}(x, W) = w_2 s(w_1 x)$ for the original neural network, and $\mathcal{M}(x, W) = w_2 s(w_1 x) + w_1 x$ for the adapted neural network. We use the sigmoid function $s(x) = 1/(1 + e^{-x})$ as activation function, which has the following well-known properties: $\lim_{x\to\infty} s(x) = 1$, $\lim_{x\to-\infty} s(x) = 0$, and $\lim_{x\to\pm\infty} s'(x) = 0$.

Figure 4 shows the error function and its derivatives for this neural network. In the top row of this figure we see the original neural network. Notice in the top middle figure, depicting $\partial E/\partial w_1$, that $\partial E/\partial w_1 \approx 0$ for small values of w_2. In other words, when w_2 is small, w_1 will hardly change its value. Similarly, when w_1 is large, then $\partial E/\partial w_1$ will be small due to the fact that the derivative of the transfer function is nearly 0. In the bottom row the modified neural network is depicted. Left, again, the error function. The middle figure clearly shows that the derivative has no areas anymore which are zero or very small. The right figure still shows that, if w_1 has a large negative value, $\partial E/\partial w_2$ is negligible: after all, the activation value of the hidden unit is near-zero.

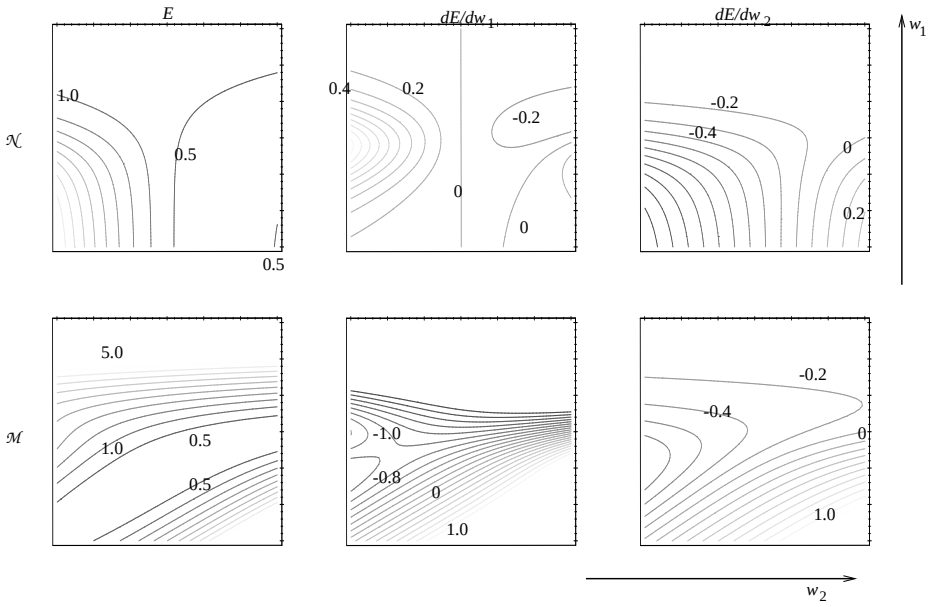

Fig. 4. Error function and derivatives using the original and adapted learning rule. The top row shows the error function for the original learning rule (left), as well as its derivative $\partial E/\partial w_1$ (centre) and $\partial E/\partial w_2$ (right). The bottom row shows the same graphs for the adapted learning rule. The contour lines have a distance of 0.5 (left graphs) and 0.2 (middle and right graphs).

4 Applications

We have applied the new learning scheme to several approximation problems. In all problems, each network has been run 3,000 times with different initial random values for the weights. In order to train the network, we used a Polak-Ribière conjugate gradient optimisation technique with Powell restarts [14].

The applications with real data (problems 3 and 4) use two independent sets of data: a learn set and a cross-validation set. In all cases, the network was trained until the error over the cross-validation set was minimal (i.e., up to but not beyond the point where the network started to over-fit). The approximation errors that are reported are computed using the samples in the cross-validation set.

Problem 1: (synthetic data) the XOR problem. The well-known XOR problem consists of representing four learning samples $[(0,0),0]$, $[(0,1),1]$, $[(1,0),1]$, and $[(1,1),0]$. The network has two inputs, two hidden units, and one output.

It has been noted [5] that the XOR problem is very atypical in neural network learning, because it is penalised for generalisation. Nevertheless, the XOR problem is generally considered as a small standard learning benchmark problem.

Whereas the network $\mathcal{N}$ reaches local minima in 22.4% of the runs, the linearly augmented network $\mathcal{M}$ always reached a global minimum. For $\mathcal{N}$, the average number of steps to obtain an approximation error of 0.0 (within the 32-bit floating point accuracy of the computer) equals 189.1; for $\mathcal{M}$, 98.3 steps were required.

When using the ordinary error back-propagation learning rule (i.e., no conjugate gradient learning), the XOR learning problem has been reported to lead to 8.7% local minima [14].

Problem 2: (synthetic data) approximating $\tan(x)$. As a second test, the networks have been used for the approximation of the function $\mathcal{F}(x) = \tan(x)$. The function has been uniformly sampled in the domain $[0, \pi]$ using 20 learning samples in total. For the approximation we used a network with one input, five hidden units in a single hidden layer, and one output.

With network $\mathcal{N}$, 14.6% of the runs lead to a local minimum. The linearly augmented neural network is not perfect; it is stuck in a local minimum in 6.2% of the runs. In all other cases, a global minimum of very close to 0.0 was found.

Problem 3: (real data) approximating robot arm inverse kinematics. Thirdly we approximated data describing a hand-eye coordination learning problem with a Manutec R2 robot arm.

The data is organised as follows. There are five input variables, describing the current position of the robot arm in joint angles θ_2, θ_3, as well as the visual position of an object with respect to the hand-held camera in pixel coordinates x, y, z. The output part of the data consists of the required joint rotations $\Delta\theta_1$, $\Delta\theta_2$, and $\Delta\theta_3$ necessary to reach the object. See [15] for further description of this hand-eye coordination problem. In this particular problem, 1103 learning

samples are used; 1103 samples are used for cross-validation. The optimal of six hidden units in a single layer [15] is used. Only the cross-validating data is used for evaluating the network.

With the normal network $\mathcal{N}$, in 2.3% of the cases the network got stuck in a minimum with an unacceptably high error for both the learn and test sets. This never occurred in the 3,000 trials in which the linearly augmented network was used. The cross-validated approximation error after 10,000 learning steps equals $4.20 \cdot 10^{-4}$ for network $\mathcal{N}$, and $4.04 \cdot 10^{-4}$ for network $\mathcal{M}$ (both values are average per learning sample). The new method shows a slight improvement here.

Problem 4: (real data) gear box deformation data. The final test consists of the following problem, which is encountered in the control of a newly developed lightweight robot arm. A gear box connects a DC motor with a robot arm segment. In order to position the robot arm segment at a desired joint angle θ_a, the DC motor has to exert a force τ. However, in the normal setup a joint angle decoder is only available at the DC motor side, which measures the angle θ_m. By mounting an extra decoder at the arm segment in a laboratory setup we can measure θ_a. The actual joint angle θ_a differs slightly from θ_m because of the gear-box elasticity. We attempt to learn θ_a from $(\theta_m, \dot{\theta}_m, \tau)$.

In order to learn these data, we use a network with 3 inputs, 6 hidden units in a single layer, and one output. The data consists of 4,000 learning samples as well as 4,000 samples used for the cross-validation. In each run, up to 10,000 learning iterations are performed but not beyond the point of overfitting.

Both network $\mathcal{N}$ as $\mathcal{M}$ do not get stuck in a minimum with an unacceptably high error for both the learn and test sets. A surprise, however, is encountered in the cross-validated approximation error that is computed after 10,000 learning steps. For network $\mathcal{N}$, this error equals $2.85 \cdot 10^{-4}$ per learning sample; for $\mathcal{M}$, however, this error is as low as $1.87 \cdot 10^{-6}$ per sample. Further data analysis has shown that the data has strong linear components, which explains the fact that the approximation error is two orders of magnitude lower.

Results are summarised in table 1.

Table 1. Results of the ordinary feed-forward network $\mathcal{N}$ and the linearly augmented feed-forward network $\mathcal{M}$ on the four problems.

		$\mathcal{N}$	$\mathcal{M}$
XOR	% local minima	22.4	0.0
	avg. steps	189.1	98.3
tan	% local minima	14.6	6.2
robot	% local minima	2.3	0.0
3D data	avg. error	$4.20 \cdot 10^{-4}$	$4.04 \cdot 10^{-4}$
gear box	% local minima	0.0	0.0
data	avg. error	$2.85 \cdot 10^{-4}$	$1.87 \cdot 10^{-6}$

5 Conclusion

It has been shown that the ordinary back-propagation learning rule leads to bad conditioning in the matrix of second-order derivatives of the error function which is encountered in feed-forward neural network learning. This again leads to local minima and saddle points in the error landscape. In order to alleviate this problem, we have introduced an adaptation to the back-propagation rule, which can be implemented as an adapted feed-forward neural network structure. We call this network the the *linearly augmented feed-forward neural network*. The adaptation leads to a learning rule which obtains stable values for the weights which connect the input units with the hidden units, even while the weights from hidden to output units change.

A mathematical analysis shows the validity of the method; in particular, the universal approximation theorems are shown to remain valid with the new neural network structure. The application of the new method to two sets of synthetic data and two sets of real data shows that the new method is much less sensitive to local minima, and reaches an optimum in fewer iterations.

Acknowledgments

The authors acknowledge the help of Alin Albu-Schäffer in supplying the gear box data.

References

1. E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons, 1989.
2. R. Battiti. First- and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation*, 4:141–166, 1992.
3. G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.
4. Bhaskar DasGupta, Hava T. Siegelmann, and Eduardo D. Sontag. On the complexity of training neural networks with continuous activation functions. *IEEE Transactions on Neural Networks*, 6(6):1490–1504, November 1995.
5. S. E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU–CS–88-0-162, Carnegie Mellon University, September 1988.
6. K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):193–192, 1989.
7. Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.
8. K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
9. H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(1):400–407, 1951.
10. S. Saarinen, R. Bramley, and G. Cybenko. Ill-conditioning in neural network training problems. *Siam Journal of Scientific Computing*, 14(3):693–714, May 1993.

11. N. N. Schraudolph. On centering neural network weight updates. Technical Report IDSIA-19-97, IDSIA, April 1997.
12. E. D. Sontag and H. J. Sussmann. Backpropagation can give rise to spurious local minima even for networks without hidden layers. *Complex Systems*, 3(1):91–106, February 1989.
13. K. P. Unnikrishnan and K. P. Venugopal. Alopex: A correlation based learning algorithm for feedforward and recurrent neural networks. *Neural Computation*, 6:469–490, 1994.
14. P. van der Smagt. Minimisation methods for training feed-forward networks. *Neural Networks*, 7(1):1–11, 1994.
15. Patrick van der Smagt. *Visual Robot Arm Guidance using Neural Networks*. PhD thesis, Dept of Computer Systems, University of Amsterdam, March 1995.
16. Q. J. Zhang, Y. J. Zhang, and W. Ye. Local-sparse connection multilayer networks. In *Proc. IEEE Conf. Neural Networks*, pages 1254–1257. IEEE, 1995.