TRAINING RECURRENT NETWORKS

Morten With Pedersen
CONNECT, Department of Mathematical Modelling, Building 321
Technical University of Denmark
DK-2800 Lyngby, Denmark
Phone: + 45 45253920
Fax: + 45 45872599
email: mwp@imm.dtu.dk

Abstract - Training recurrent networks is generally believed to be a difficult task. Excessive training times and lack of convergence to an acceptable solution are frequently reported. In this paper we seek to explain the reason for this from a numerical point of view and show how to avoid problems when training. In particular we investigate ill-conditioning, the need for and effect of regularization and illustrate the superiority of second-order methods for training.

1 introduction

Recurrent neural networks are an interesting class of models for signal processing as they are able to build up internal memory suited for the task at hand and thus often lead to compact model representations. However, it is generally believed to be a difficult task to train this type of networks. Several authors have addressed the learning problem for recurrent networks, e.g., in the context of sequence classification when required to store information for an arbitrary period of time [1, 5] but to the best of the authors knowledge no one have treated the problem from a general *numerical* point of view.

Feedforward networks were treated extensively from a numerical point of view in [8] where it was illustrated how training forms an extremely ill-conditioned optimization problem. In this contribution we extend this analysis to include recurrent networks. In particular we identify redundant connections and illustrate how ill-conditioning may otherwise arise, which motivates the use of regularization.

Having acknowledged the need for regularization makes way for the highly effective second-order methods for training. In this contribution we particulary focus on the *damped* Gauss-Newton method and illustrate how this method by far outperforms gradient descent on a time series prediction problem, namely the Santa Fe laser data. The focus in this contribution is on time series prediction, but the results generalize to other applications as well.

2 Architecture

The general architecture of the networks considered here are fully connected feedback networks with one hidden layer of nonlinear units and a single linear output unit as illustrated in figure 1. The output $y(t)$ of the network is linear in order to allow for arbitrary dynamical range, and is given by

$$y(t) = \sum_{i=1}^{N_h} w_{oi} s_i(t) + w_{ob} \tag{1}$$

where N_h is the number of hidden units, w_{oi} is the weight to the output unit from hidden unit j and w_{ob} is a bias weight. The output $s_i(t)$ from hidden unit i at time t is computed as

$$s_i(t) = f\left(\sum_{j=1}^{N_h} w_{ij} s_j(t-1) + w_{io} y(t-1) + \sum_{k=1}^{N_I} w_{ik} x_k(t) + w_{ib}\right) \tag{2}$$

where w_{ij} is the weight *to* hidden unit i *from* hidden unit j, w_{io} is the weight *to* hidden unit i *from* the output unit and w_{ib} is the bias weight for hidden unit i. $x_k(t)$ is the k'th element in the external input vector $\mathbf{x}(t)$ at time t and N_I is the total number of external inputs. $f(\cdot)$ is the nonlinear activation function, in this work we use $f(x) = \tanh(x)$.

Note that the update of the recurrent network presented above is *layered*, as the outputs $s_i(t)$ from the hidden units are computed immediately *before* the computation of the output unit output. This is opposed to the update presented in e.g. [11] where all the units are updated simultaneously. In [6] it was shown that when using fully recurrent networks for forecasting, layered update is preferable since synchronous update of the units effectively results in a two-step ahead predictor. Also note that the linear output unit does not have feedback of its own previous value. This is in order to avoid stability problems that are otherwise likely to occur.

2.1 Training

In this work we focus on time series prediction in which case the input vector contains delayed elements of the time series, $\mathbf{x}(t) = [x(t), \ldots, x(t - N_I + 1)]$, and the network output is a prediction of the next value in the series,

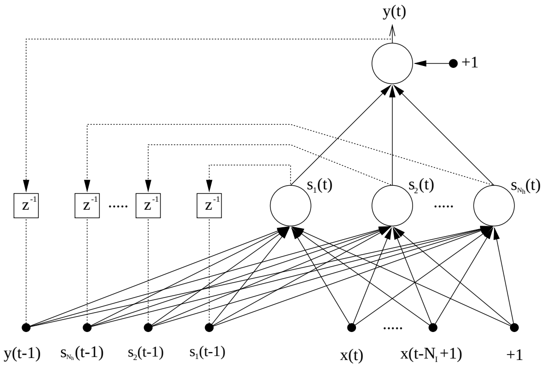

Figure 1: Architecture of the recurrent networks considered in this work.

$\widehat{x}(t+1) = y(t)$. Training the network means adjusting the weights so as to minimize a cost function. Most applications are based on the sum of squared errors,

$$E(\mathbf{w}) = \frac{1}{2}\sum_{t=1}^{T}[e(t)]^2 \ \ , \ \ e(t) = x(t+1) - y(t) \tag{3}$$

where T denotes the number of training examples and $\mathbf{w}$ is the concatenated set of parameters. The adjustment of the parameters is done *off line* by an iterative sheme, $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta\Delta\mathbf{w}_k$, where $\Delta\mathbf{w}_k$ indicates the direction of change and η is the (adaptive) size of the step. When training recurrent neural networks the most commonly used scheme is gradient descent, where the direction $\Delta\mathbf{w}_k$ is equal to the gradient $\mathbf{g}$, $g_i = \partial E(\mathbf{w}_k)/\partial w_i$. Unfortunately this method suffers from extremely slow convergence, and the quality of resulting solutions is often not satisfactory.

Experiments have shown that much more efficient training can be obtained by using second-order methods [6]. Here we particulary focus on the *damped* Gauss-Newton method [3], in which the search direction $\Delta\mathbf{w}_k$ is determined by

$$\Delta\mathbf{w}_k = \mathbf{H}^{-1}\mathbf{g} \tag{4}$$

where $\mathbf{H}$ is the positive semidefinite approximation to the Hessian,

$$H_{ij} = \sum_{t=1}^{T}\frac{\partial y(t)}{\partial w_i}\frac{\partial y(t)}{\partial w_j} \approx \frac{\partial^2 E(\mathbf{w}_k)}{\partial w_i \partial w_j} = -\sum_{t=1}^{T}\left[e(t)\frac{\partial^2 y(t)}{\partial w_i \partial w_j} - \frac{\partial y(t)}{\partial w_i}\frac{\partial y(t)}{\partial w_j}\right] \tag{5}$$

In each iteration k the step size η is determined by line search which makes the method globally convergent [3]; here we recommend a simple approach where η is halved until a decrease in the cost is obtained [3]. The iterations are continued until convergence, determined by a sufficiently small length of the gradient, $\|\mathbf{g}\|_2 < \varepsilon$.

The Gauss-Newton method involves finding the solution to a linear system of equations $\mathbf{H}\Delta\mathbf{w}_k = \mathbf{g}$ in each iteration, but the increased computational burden is justified by a dramatic increase in convergence and thus reduction of overall training time, even for large networks as we shall see. However, the success of the damped Gauss-Newton method relies heavily on the conditioning of the training problem, as is the case for gradient descent.

3 Ill-conditioning

When training using either gradient descent or the Gauss-Newton method, a measure of great importance for the convergence is the condition number of the Hessian $\mathbf{H}$. For a symmetric positive definite matrix $\mathbf{H}$, the condition number is defined as $\kappa(\mathbf{H}) = \lambda_{max}/\lambda_{min}$, the ratio between the largest and smallest eigenvalue of $\mathbf{H}$. If the condition number is *large*, the Hessian becomes *ill-conditioned*. The convergence rate will suffer and the solution to the linear system of equations (4) in the Gauss-Newton method becomes unreliable. As a rule of thumb the solution may not be trustworthy if $\kappa(\mathbf{H}) > \epsilon^{-1/2}$, where ϵ denotes the machine precision [3]. For the widely used IEEE 64-bit floating point representation this is equivalent to $\kappa(\mathbf{H}) > 6.7 \cdot 10^7$. This may seem as a large number, but this order of magnitude is not uncommon in the framework of either feedforward networks [8] or recurrent networks as we shall see.

In [2] it was shown that an eigenvalue of the order of the number of input variables could be avoided if the mean was subtracted from each of the input variables $x_k(t)$ and if a symmetric activation function is used. However, these simple countermeasures are not adequate for avoiding ill-conditioning in recurrent networks, as the analysis in the following will show.

The Hessian (5) can also be written as

$$\mathbf{H} = \mathbf{J}^{\mathrm{T}}\mathbf{J} \ , \ J_{ti} = \frac{\partial y(t)}{\partial w_i} \tag{6}$$

where $\mathbf{J}$ is the Jacobian matrix, whose columns are the partial derivatives of the network output at each timestep in the training series. If $\mathbf{J}$ is rank-deficient some of the columns are linearly dependent, which is indicated by singular values with the value zero in an SVD analysis [7]. This again leads to a singular Hessian and thus an infinite condition number. In practice it is rare to find columns in $\mathbf{J}$ that are *exactly* dependent and thus singular values that are exactly zero [8]. However, it is often the case that columns are *nearly* linearly dependent, which leads to very small singular values of $\mathbf{J}$ and thus large condition numbers for the Hessian $\mathbf{H}$. In the following sections we describe situations that lead to ill-conditioning of $\mathbf{J}$ for recurrent networks, arising from both exact and approximate linearly dependencies between columns in $\mathbf{J}$.

3.1 Exact dependency

For the type of recurrent networks defined by (1) and (2) there is built-in rank deficiency in the Jacobian since it is easy to show that some of the columns in $\mathbf{J}$ will *always* be linear combinations of each other. This is illustrated by an example for a small network, but the result apply for networks with an arbitrary number of hidden units. The network considered here involves only one external input and one hidden unit, and the output is thus defined as

$$\begin{aligned} y(t) &= w_{o1}s_1(t) + w_{ob} & (7)\\ s_1(t) &= f\left(w_{11}s_1(t-1) + w_{1o}y(t-1) + w_{1x}x(t) + w_{1b}\right) & (8)\\ &= f\left((w_{11} + w_{1o}w_{o1})s_1(t-1) + w_{1x}x(t) + (w_{1b} + w_{1o}w_{ob})\right) & (9) \end{aligned}$$

where (9) is obtained by insertion of (7) in (8). We see that the network output will remain unchanged as long as the total weighting k_1 of $s_1(t-1)$, $k_1 = w_{11} + w_{1o}w_{o1}$, and the total bias k_2 on the hidden unit, $k_2 = w_{1b} + w_{1o}w_{ob}$, remains constant. w_{o1} and w_{ob} can not be changed without directly affecting the network output (7) and are therefore kept fixed which we denote by $*$. However, changes in w_{11}, w_{1o} and w_{1b} that satisfies *both* expressions

$$\begin{array}{ccccccccc} w_{11} & + & w_{o1}^* \cdot w_{1o} & + & 0 & = & k_1 \\ 0 & + & w_{ob}^* \cdot w_{1o} & + & w_{1b} & = & k_2 \end{array} \qquad (10)$$

will leave the network output unchanged. The expressions (10) form hyperplanes in parameter space spanned by w_{11}, w_{1o} and w_{1b} and their line of intersection is computed as

$$(w_{11}, w_{1o}, w_{1b}) = (k_1, 0, k_2) + t(-w_{o1}^*, 1, -w_{ob}^*) \qquad (11)$$

parametrized by t. The line defines a direction in parameter space in which the network output is constant. The constant network output means that derivatives are zero in this direction. Thus, columns in the Jacobian corresponding to (w_{11}, w_{1o}, w_{1b}) are linearly dependent.

When investigating Jacobians for the dependency problem outlined above it is however uncommon to encounter singular values *exactly* equal to zero; but according to the derivations this clearly ought to be the case. The reason for this is the initialization of previous state values when starting up the network. If the recurrent network starts iteration at time $t = 1$ it is common practice [11] to set the previous states of the hidden units as well as their derivatives to zero,

$$s_i(0) = 0 \ , \quad \frac{\partial s_i(0)}{\partial \mathbf{w}} = 0 \, . \qquad (12)$$

This startup procedure clearly marks an initial discontinuity in the recursive equations (7) and (8) governing the feedback network. Thus initially the partial derivatives wrt. the involved weights in the Jacobian will generally *not* be linearly dependent. But after a few iterations indicating a transient, the dependency arises with increasing accuracy.

The linear dependency is eliminated if we omit the feedback weights w_{io} *from* the output *to* the hidden units i, as the degeneracy can then no longer occur. This elimination has no influence on the modeling capabilities of the network since the remaining weights can be adjusted so that the network output remains unaffected.

3.2 Approximate dependency

Even though removal of the feedback weights w_{io} leading from the linear output to the hidden units removes the problem of almost exact rank-deficiency in the Jacobian for recurrent networks it does not eliminate ill-conditioning as experiments show. In [8] the problem of ill-conditioning was analyzed for *feedforward* networks by careful examination of the components entering the partial derivatives $\partial y(t)/\partial w_i$ of the network output and it was found that ill-conditioning in the Jacobian can arise from at least these three reasons (assuming that the external inputs are not proportional):

1. The output from a hidden unit is saturated and constant ($\equiv \pm 1$).
2. The outputs from two hidden units are approximately proportional.
3. The derivatives of two hidden unit outputs wrt. their activations are approximately proportional.

Theoretical and empirical examinations of the components entering the partial derivatives for *recurrent* networks reveal that ill-conditioning may arise here as well from the same reasons; such analysis is however not included here.

Situation 2 where the outputs of two hidden units are proportional and thus highly correlated often occurs in practice; e.g., in [10] high correlation between hidden unit outputs was found and studied for feedforward networks.

The effects of situation 2 are similar to those described in section 3.1, as we can determine directions in parameter space in which the cost function is approximately constant. For recurrent networks this situation is much more severe than for feedforward networks since the degeneracy will not only affect weights leading to the output, but also many weights connecting the hidden units as the experiments will show.

The scenarios listed above lead to nearly linearly dependencies between the columns of $\mathbf{J}$ and thus to small eigenvalues in $\mathbf{H}$. However, the condition number of a matrix is determined by the *ratio* between the largest and smallest eigenvalues, thus problems do not only arise from small singular values but also from large values.

As mentioned, the situations described above will lead to directions in parameter space where the cost is approximately constant, thus when training using the Gauss-Newton method the search direction will be dominated by these directions leading to an unrestrained growth in the magnitude of the affected weights. This again leads to a significant growth in the magnitude of several of the columns in the Jacobian since many derivatives are dominated by terms of the form [11]

$$\frac{\partial s_k(t)}{\partial w_{pq}} \propto \sum_{j=1}^{N_h} w_{kj} \frac{\partial s_j(t-1)}{\partial w_{pq}} \tag{13}$$

which becomes large if the weights w_{kj} become large. The large elements in the Jacobian lead to an overall upward scale of the elements in $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ and thus to an upward shift of the eigenvalues.

4 Regularization

A traditional method for handling ill-conditioning is by *regularizing* the cost function [3, 4]. A simple yet highly effective regularization can be obtained by augmenting the cost function by a simple quadratic weight decay [4],

$$C(\mathbf{w}) = E(\mathbf{w}) + \frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \tag{14}$$

Simple weight decay is often primarily considered as a means for avoiding overfitting as it puts constraints on the parameters and thus reduces the degrees of freedom. Weight decay should however also be considered from its regularizing effects. The immediate effect is that α gets added to the diagonal of the Hessian which puts a lower bound on the smallest eigenvalues, since it is easy to show that $\lambda(\mathbf{H} + \alpha\mathbf{1}) = \lambda(\mathbf{H}) + \alpha$. Another effect is the limit imposed on the growth of the weights which prevents near singular directions in parameter space from dominating the search directions obtained by the Gauss-Newton method, thus greatly improving the efficiency of the optimization.

The constraints put on the weights by the regularization has a smoothing effect on the cost function which was clearly illustrated in [6]. Here it was also demonstrated that the significance of the second order term ignored in (5) diminishes when using simple weight decay as regularization.

5 Experiments

In the first experiment we illustrate how ill-conditioning results from the situations described in section 3.2 and how regularization improves training. For this experiment we used a simple recurrent network to predict the laser data from the Santa Fe time series prediction competition [9]. The data were scaled so that the first 1000 points used for training had zero mean and unit variance and the following 100 values were used as a test set. The network used had one external input and three hidden units; there were *no feedback from the linear output unit* to the hidden units, in line with section 3.1. Training was performed initially using five iterations of gradient descent followed by the damped Gauss-Newton method. In the left panel of figure 2 is shown the evolution of the *mean* squared errors normalized by the variance of the sets (NMSE, [9]) when training without regularization. It seems that training is converging to a solution, but this is *not* the case as the evolution of the weights in the right panel of figure 2 shows. What happens is that the outputs of two hidden units become almost proportional; this is revealed by the cosine to the angle θ between vectors containing their outputs on the training set which at iteration 100 is $\cos\theta = 0.9998$. This corresponds to situation 2 described in section 3.2. The weights that grow in magnitude are the pairs of weights leading *from* these two units *to* every unit in the network including the output. Note that the error and thus the network output is unaffected since the effects of the changes in the growing weights cancel out due to the dependency between the hidden units.

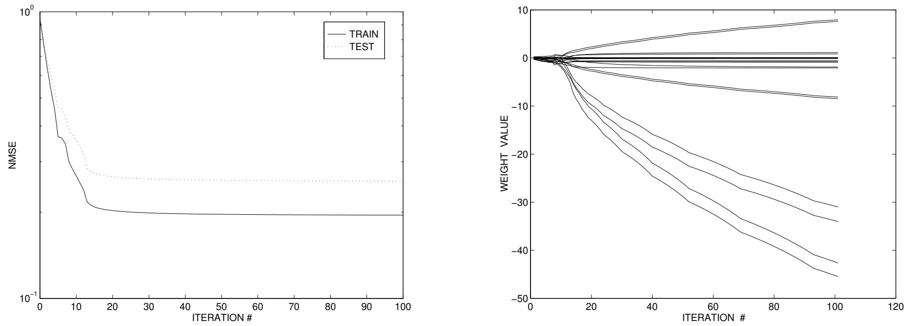

Figure 2: Training without regularization. Left panel: Evolution of training and test errors. Right panel: Evolution of the weight values.

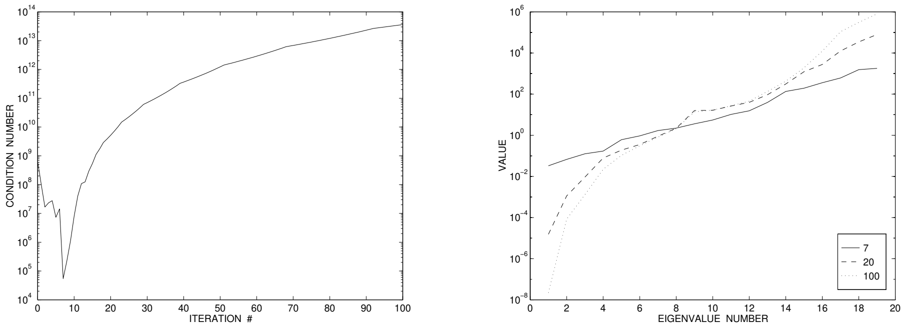

Figure 3: Training without regularization. Left panel: Evolution of the condition number for $\mathbf{H}$. Right panel: Eigenvalues after iterations 7, 20 and 100.

The condition number during training is shown in the left panel of figure 3 and is seen to grow enormously. The rapid increase occurs shortly after the initiation of the second-order method which quickly 'discovers' the dependency between the hidden unit outputs. The near singular Hessian $\mathbf{H}$ leads to very large weight changes in some directions when solving (4). The large steps are however handled by the line search which returns very small step sizes, indicated by the smooth increase in the weight magnitudes. In the right panel of figure 3 is shown the eigenvalues of the Hessian after iterations 7, 20 and 100. At each of the iterations it can be seen that the condition number results from both very small as well as very large eigenvalues and we note that as training progresses the eigenvalues extend both upward and downward.

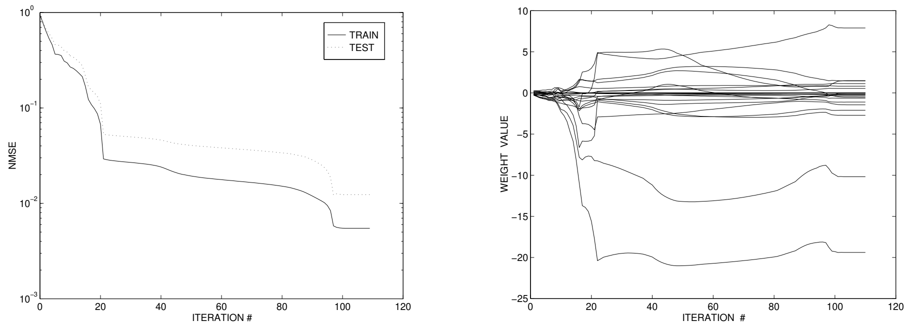

Figure 4: Training with regularization, $\alpha = 10^{-3}$. Left panel: Evolution of training and test errors. Right panel: Evolution of the weight values.

The training was then repeated using the exact same initial weights and the same training approach, but now with a regularization term added to the cost function, using $\alpha = 10^{-3}$. In the left panel of figure 4 is shown the resulting evolution of the errors. The positive effect of the regularization is evident, as the final errors are several orders of magnitude below the levels shown in figure 2 obtained without regularization. Furthermore the stopping criterion $\|\mathbf{g}\|_2 < 10^{-4}$ was satisfied; in the previous experiment using no regularization the gradient norm grew proportional to the condition number.

In the right panel of figure 4 we see that the regularization term limits the growth of the weights compared to figure 2. Some however still grow large as does the condition number shown in the left panel of figure 5. Even though the condition number grows to 10^8 the damped Gauss-Newton method still manages to find a minimum. Experience shows that for this method successful training to a (local) minimum can be obtained for condition numbers up to about 10^8 in magnitude. This may depend on the decomposition algorithm used when solving (4), here we use the fast and stable Cholesky factorization [3, 7].

From the right panel of figure 5 we learn that the reduction in condition number is obtained *only* from an increase in the smallest eigenvalues resulting from the regularization. The largest eigenvalues are of the same order of magnitude as when training without weight decay as we see in figure 3. This is due to the still fairly large weight magnitudes, in line with section 3.2. If the regularization term α is further increased the larger eigenvalues will also be affected; but so will the modeling capabilities of the network, leading to increased errors.

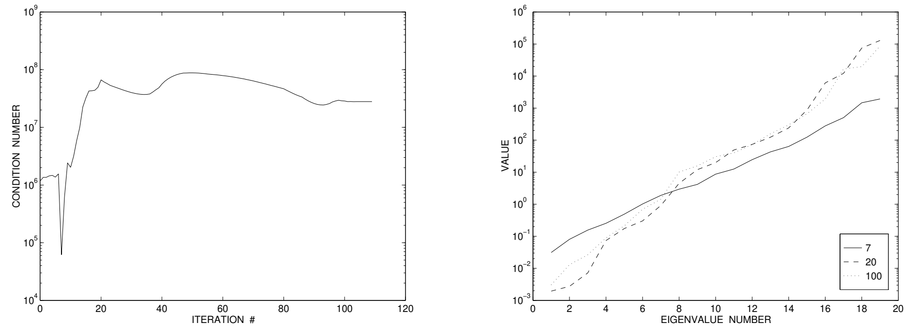

Figure 5: Training with regularization, $\alpha = 10^{-3}$. Left panel: Evolution of the condition number for $\mathbf{H}$. Right panel: Eigenvalues after iterations 7, 20 and 100.

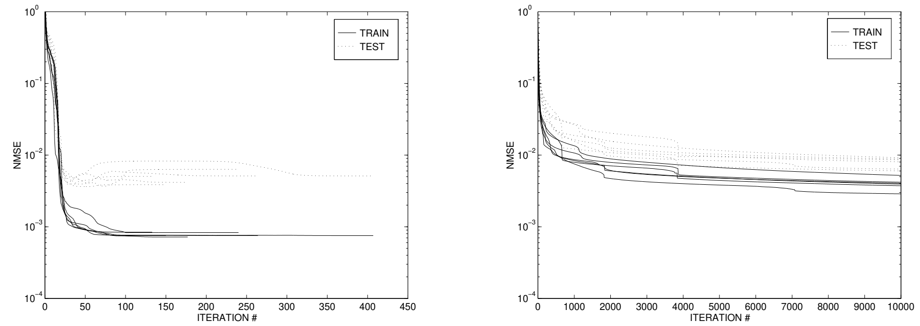

Figure 6: Evolution of errors using different optimization methods. Left panel: Damped Gauss-Newton method. Right panel: Gradient descent with line search.

In the final experiment we compare the performance of damped Gauss-Newton with a gradient descent algorithm also using the step-size halving line search. The problem is still prediction of the laser series but using larger networks with a single input and nine hidden units, 109 weights in total (no feedback from the output to the hidden units). Thus, each iteration using damped Gauss-Newton involved solution of a 109 by 109 linear system of equations. Six initial networks were generated by initializing their weights with values drawn from a uniform distribution over the interval $[-0.3; 0.3]$. The training algorithms were then compared when starting from the same six initial networks, both using regularization $\alpha = 0.02$. The resulting evolution of errors is shown in figure 6; in the left panel we see the resulting errors using the damped Gauss-Newton method, in the right panel using gradient descent. Using both methods the stopping criteria was set to $\|\mathbf{g}\|_2 < 10^{-4}$ or maximum 10000 iterations.

For the damped Gauss-Newton method the stopping criterion was met in all six runs. The average training error (Normalized Mean Squared Error) was $7.7 \cdot 10^{-4}$, the average test error was $4.9 \cdot 10^{-3}$. The *average* time for a complete training run was 200 seconds. For gradient descent the stopping criterion was never met, the termination of the algorithm in each run was due to maximum number of iterations reached. The average training error obtained after the maximum allowed 10000 iterations was $4.0 \cdot 10^{-3}$, the average test error was $7.8 \cdot 10^{-3}$. The average time used for obtaining these error levels was 8140 seconds.

Note that the levels of both training and test errors obtained using gradient descent are much higher than the levels obtained using the damped Gauss-Newton method even though gradient descent used a factor of 50 times more iterations and a factor of 40 times more computer time. Thus, even though an iteration of the damped Gauss-Newton method is computationally more costly than an iteration of gradient descent, the additional cost is highly justified by the vastly increased convergence rate. Similar justification has been observed for networks with up to 300 parameters.

6 Conclusion

In this paper we have focused on sources of ill-conditioning and thus the need for regularization when training recurrent networks especially using second-order methods. Once this need has been recognized dramatic improvement in convergence rate and quality of solution is obtained, even for large size problems.

Acknowledgements

The author would like to thank Lars Kai Hansen and Jan Larsen for support. This research is supported by the Danish Natural and Technical Research Coucils through the Computational Neural Network Center (CONNECT).

REFERENCES

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[2] Y. Le Cun, I. Kanter, and S. A. Solla. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, 66(18):2396–2399, 1991.

[3] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, NJ, 1983.

[4] S. Haykin. *Neural Networks, A Comprehensive Foundation.* Macmillan, New York, NY, 1994.

[5] S. Hochreiter and J. Schmidhuber. Long short term memory. Technical Report FKI-207-95, Fakultat fur Informatik, Technische Universitat Munchen, Munchen, 1995.

[6] M. W. Pedersen and L. K. Hansen. Recurrent networks: Second order properties and pruning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 673–680. The MIT Press, 1995.

[7] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, Cambridge, second edition, 1992.

[8] S. Saarinen, R. Bramley, and G. Cybenko. Ill-conditioning in neural network training problems. *SIAM Journal on Scientific Computing*, 14:693–714, 1993.

[9] A. S. Weigend and N. A. Gershenfeld, editors. *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison-Wesley, Reading, MA, 1993.

[10] A. S. Weigend and D. E. Rumelhart. The effective dimension of the space of hidden units. In Elaine Keramides, editor, *Proceedings of INTERFACE'91: Computing Science and Statistics.* Springer Verlag, 1992.

[11] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.