Report Number: WUCS-90-21

1990-06-01

# Super Linear Learning in Back Propagation Neural Nets

Barry L. Kalman

The important feature of this work is the combination of minimizing a function with desirable properties, using the conjugate gradient method (cgm). The method has resulted in significant improvements for both easy and difficult training tasks. Two major problems slow the rate at which large back propagation neural networks (bpnns) can be taught. First is the linear convergence of gradient descent used by modified steepest descent method (msdm). Second is the abundance of saddle points which occur because of the minimization of the sum of squared errors. This work offers a solution to both difficulties. The cgm which is... **Read complete abstract on page 2.**

# Super Linear Learning in Back Propagation Neural Nets

Barry L. Kalman

## Complete Abstract:

The important feature of this work is the combination of minimizing a function with desirable properties, using the conjugate gradient method (cgm). The method has resulted in significant improvements for both easy and difficult training tasks. Two major problems slow the rate at which large back propagation neural networks (bpnns) can be taught. First is the linear convergence of gradient descent used by modified steepest descent method (msdm). Second is the abundance of saddle points which occur because of the minimization of the sum of squared errors. This work offers a solution to both difficulties. The cgm which is super linearly convergent replaces gradient descent. Division of each squared error term by its derivative and then summing the terms produces a minimization function with a significantly reduced number of saddle points.

# SUPER LINEAR LEARNING IN BACK PROPAGATION NEURAL NETS

Barry L. Kalman

WUCS-90-21

June 1990

Center for Intelligent Computing Systems
Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

CICS is supported by grants from McDonnell Douglas and Southwestern Bell.

# ABSTRACT

The important feature of this work is the combination of minimizing a function with desirable properties, using the conjugate gradient method(cgm). The method has resulted in significant improvements for both easy and difficult training tasks.

Two major problems slow the rate at which large back propagation neural networks(bpnns) can be taught. First is the linear convergence of gradient descent used by modified steepest descent methods (msdm). Second is the abundance of saddle points which occur because of the minimization of the sum of squared errors.

This work offers a solution to both difficulties. The cgm which is super linearly convergent replaces gradient descent. Division of each squared error term by its derivative and then summing the terms produces a minimization function with a significantly reduced number of saddle points.

## INTRODUCTION

This paper presents work on faster learning methods for bpnns[RM]. Two major ideas are presented in this study. The first is the minimization of a function which imposes a severe penalty for having the wrong output at a node and has few local extrema and saddle points. The second is the use of the super linearly convergent cgm instead of the linearly convergent steepest descent method. Both methods are presented in the book by Polak[PE]. The author incorporated these ideas into software to compute optimal weights for several bpnns including two large examples. This work also demonstrates that for the two optimization methods discussed comparison of numbers of epochs required to find a global minimum is a fair representation of the speed of the methods.

The combination of minimizing a function with desirable properties using the cgm and applying the new technique to a large and important problem are the unique features in this work. Fahlman[F] discusses an approach which operates on a function with similar properties to ours. According to Fahlman this function does not achieve much improvement to learning and it is hard to implement. The function presented here is simple to implement and when combined with the cgm shows marked improvement in comparison to ordinary descent methods applied to any minimization function. Battiti [B] discusses the use of the cgm with the usual error function. He finds that the cgm requires fewer pattern presentations but is more expensive per presentation. He does not discuss the propensity for cgm to hang up at saddle points on large problems. Fahlman[F] extensively discusses this point for minimizers in general.

## TRAINING BACK PROPAGATION NEURAL NETWORKS

This section describes the training of neural networks. It explains how optimization is used. It also explains the history of the term "back propagation." See Appendix A for details about neural networks.

Training a bpnn consists of two steps. The first step is to specify a set of teaching patterns for the input layer and a set of target patterns that the units of the output layer should match. The second step is to adjust the weights and biases so that the activation level of each output unit for a given set of input patterns reasonably matches the corresponding target pattern. Weight adjustment is done by minimizing an error function based on the difference between the output activations and the target patterns.

The networks are called back propagation networks due to the way optimal weights and biases are found. First the weights between the hidden layer and the output layer and the biases of the output layer are adjusted based on the derivatives of the error function with respect to these weights and biases. Then the weights between the input layer and the hidden layer and the biases of the hidden layer are adjusted based on the derivatives of the error function with respect to these weights and biases. These derivatives are computed from the first set of derivatives. Hence the weight adjustment propagates back from the output layer to the input layer. The weight adjustment process is repeated until a set of weights is found that causes all output activations to reasonably match all target patterns. An epoch is processing the weights after the error function has been evaluated for one complete set of training patterns.

## THE ERROR FUNCTION AND LOCAL EXTREMA

The technique msdm[RM] finds optimum weights by varying the error function ferr over the teaching patterns. See Appendix B for a description of ferr and its derivatives. Since the usual minimization techniques work by finding a set of weights which make all the derivatives zero, any set of weights which makes all output activations ±1 will make all the derivatives zero. If not all the errors are 0 then the minimization method finds a local extremum or a saddle point. If $n_0$ is the number of output units then there are $2^{n_0} - 1$ such conditions for each configuration of the hidden units.

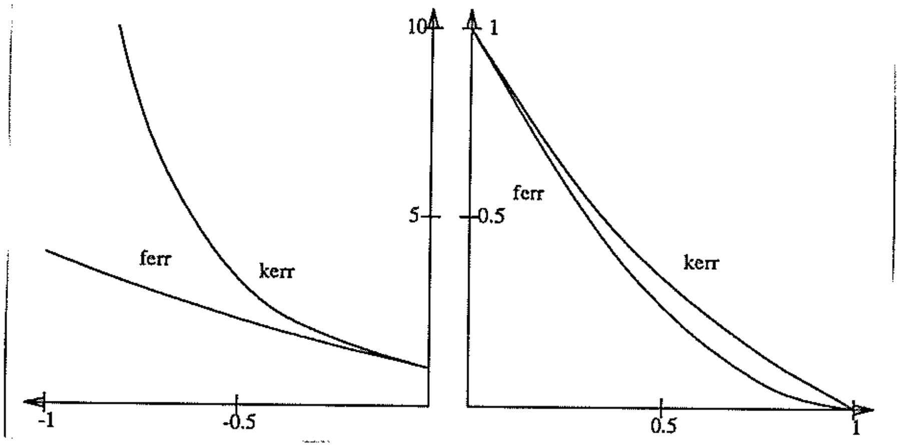
The msdm package uses heuristics like the momentum term, a small learning rate and periodic random perturbation of the weights to attempt to steer clear of local extrema and saddle points. The conjugate gradient method quite readily tends to locate local extrema and saddle points. Problems like "exor", where $n_0 = 1$, have no saddle points where all the derivatives are 0. Problems for which $n_O$ <= 10 probably will not cause difficulty for the cgm. The bpnn for the deterministic parser has $n_O = 40$. In this case the cgm with ferr always terminated at a saddle point.

If a function to replace ferr reduces the probability of the occurrence of saddle points and local extrema, then the conjugate gradient method should more quickly locate a global minimum which makes all the errors = 0. Such a function is kerr. See Appendix B for a description of kerr and its derivatives. The function kerr becomes infinite when for a particular pattern an output activation is exactly the opposite of its target value, which corresponds to having an exactly wrong result. This will drive any minimizer away from sets of weights for which ferr obviously has a saddle point or local extremum. The deterministic parser bpnn with the cgm and kerr always terminated at a global minimum. This is conclusive evidence that the cgm with kerr is a viable approach to train medium to large bpnns.

When for a particular pattern the output activation approaches its target value its contribution to kerr linearly approaches zero. This behavior will cause any minimizer of kerr to behave well near a global minimum. All minimizers of ferr will slow down near a global minimum because of the quadratic nature of the terms in ferr. Figure 1 illustrates the behavior of ferr and kerr for two output nodes along a constant contour of one of them.

If $n_H$ is the number of hidden nodes then by symmetry arguments there must be $n_H!$ equivalent global minima if there is one global minimum. $n_H!$ also multiplies the number of saddle points previously discussed. Observations based on varying $n_H$ while holding the rest of a network constant indicate that there often may be several sets of $n_H!$ global minima. Just forcing all output activations to= ±1 will not cause a local extremum or saddle point of kerr. Because of the summation terms in the derivatives it is still possible for local extrema and saddle points to occur but their likelihood is considerably reduced. (The "exor" problem is an example of this type of local extremum.) Saddle points and local extrema may be avoided by periodically shocking the weights. Another approach is to use different centers for the distributions used to randomly initialize weights and biases.

FIGURE 1: BEHAVIOR OF ferr AND kerr



Substitution of kerr for ferr should make it considerably faster for a minimizer to find a global minimum for a back propagation neural network and its training patterns. A super linear minimizer like the conjugate gradient method is more effective. Note that kerr will be zero whenever ferr is and vice versa.

## THE CONJUGATE GRADIENT METHOD

The conjugate gradient method(cgm) is used in this work because it is super linearly convergent and it does not require $O(W^2)$ space to store the second derivative(Hessian) matrix. W is the number of weights in the network. Proof of the convergence behavior of cgm is presented by Polak in [P]. The basis for the algorithms used in this work is developed in "Numerical Recipes in C" [PFTV]. High level descriptions of the algorithms are in Appendix C. The rest of this section discusses why it is fair to compare the number of epochs taken by msdm to number taken by cgm and the improved effectiveness of cgm when it is used in conjunction with kerr.

Computing activations, errors and derivatives is equivalent to back propagation. If P is the number of teaching patterns, the time complexity of back propagation is proportional to P W. The factor P comes from the fact that each weight is accessed P times. The value I is the average number of quadratic interpolation iterations per iteration of cgm. Each line minimization requires one back propagation for each iteration of quadratic interpolation and each iteration of cgm requires one back propagation for computation of the next descent direction. Each line minimization requires I W computations for directional derivatives and W weight updates. Each cgm iteration requires W computations for a directional derivative. The time complexity of an iteration of cgm for a bpnn is proportional to $W (P I + I + 2)$. The term $W (I + 2)$ is the overhead induced by the cgm. The overhead is essentially the time required by cgm in excess of the time required by I iterations of msdm. The fractional overhead of using cgm is $1 / P$ when $P \gg I$, which is nearly negligible for large bpnns. For a large bpnn one cgm iteration is equivalent to I msdm epochs. Comparing the number of epochs needed by cgm to the number required by msdm is a fair evaluation.

In the msdm package each step along the negative gradient is kept small and combined with previous descent directions through the use of a momentum term. The conjugate gradient method requires a full step along the descent direction for each iteration. The cgm descent direction is only collinear with the gradient for the first iteration. Because of the the full step computed by line minimization the cgm is more susceptible to local extrema and saddle points than msdm. In fact in experiments with cgm it has often failed to find global minima of the function ferr. However, the use of kerr makes the cgm much better than msdm used on ferr.

## RESULTS AND DISCUSSION

This section presents results of using the cgm with kerr on small medium and large bpnns. This method is currently being tried on several other networks and will be used on still others in the future. For large networks the results are very encouraging.

The new method was applied to the standard exor problem of Rumelhart and McClelland[RM]. This problem has a 2-2-1 arrangement and corresponds to 9 independent variables(weights and biases). All 4 possible training patterns were used. With the standard error function ferr and msdm this problem is solved in 270 epochs. The new method was able to solve the same problem in 105 epochs or over 2.5:1 improvement. One of the keys to this improvement is to use separate centers for the distributions which are used to randomly initialize the weights and biases.

A network was built to emulate looking up sin(x) in a table. The network was arranged as 8-16-8 and it corresponds to 280 independent variables. 20 of 256 training patterns were used. The new method successfully trained the network in 150 epochs. No data for the msdm were available.

The new method was also applied to rule-based grammar learning of Faisal and Kwasny[KF]. For one large grammar learning problem, a 66-40-40 network with biases is required. This represents 4320 independent variables. With the standard error function and msdm the solution required 500000 pattern presentations to converge. With etol set to 0.75 the new method requires 213 epochs of 433 pattern presentations or 92229 pattern presentations. (The value etol is discussed in Appendix B.) The value of I is 5.2 and $P \gg I$ is satisfied. The weights of this solution showed generalization properties quite similar to the solution found by msdm. With the standard error function cgm tends to find saddle points and is unable to achieve a satisfactory solution. The combination of kerr with cgm reduces the number of saddle points and makes global minima more attractive.

Laine and Ball[LB] are developing a neural net to be used in optical character recognition. They are using a 64-40-36 arrangement which corresponds to 4076 independent variables. An interesting feature of the training patterns of this case is that the inputs in [-1,1] instead of {-1,1}. A preliminary study yielded 224 prototypical training patterns. The new method took 174 epochs compared to 500 by msdm.

Kwasny and his group are looking at adding connectionist functionality to a deterministic parser. One thing that is desired is to represent a stack by a neural network. The network for a 2 symbol 3 level stack is

5-3-5 which corresponds to 38 independent variables. This arrangement is due to Jordan Pollack[PJ]. Pollack calls this a recursive associative automatic memory(raam). This is a small network. An interesting feature of this case is that the teaching patterns which represent the state of the stack are only known for its initial state(empty). To find the optimal weights for this structure requires convergence on the weights and the other possible states of the stack; that is a double iteration. The new method was used for the inner iteration and the outer iteration was based on convergence of the stack states. Since the epochs are of different sizes we report the time it took. Using a SUN 4/110 with floating point accelerator took 12 sec. No data are available from msdm yet.

We are investigating applications of the new method to a cross section of problems. Since it is complementary to other methods we are also assessing how a combination of methods performs. We are adding the new method to a neural network software package that already supports msdm.

## APPENDIX A -- DETAILS OF NEURAL NETWORKS

Rumelhart and McClelland[RM] describe a bpnn as a layer of input units, a layer of output units and at least one layer of hidden units. Figure A-1 lists some of the terminology of neural networks. Figure A-2 shows an example. Each unit in each layer is connected every unit in the next layer by an arc. Each arc has a weight associated with it. Each hidden unit and output unit has a bias associated with it. A network is specified by listing $n_I$- $n_H$- $n_O$. For example a network with 5 input units, 3 hidden units and 5 output units is specified as 5-3-5. Figure A-2 shows a generic network.

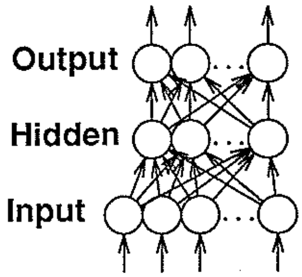
FIGURE A-1. TERMINOLOGY OF NEURAL NETWORKS

$n_O$ : number of output units.

$n_H$ : number of hidden units.

$n_I$ : number of input units.

FIGURE A-2 : A NEURAL NETWORK



## APPENDIX B -- DETAILS ABOUT ERROR FUNCTIONS

FIGURE B-1. TERMINOLOGY OF ERROR FUNCTIONS

i : index for output units.

j : index for hidden units.

k : index for input units.

p : index for teaching patterns.

v(x) : sigmoidal function centered around zero;

$a_{p,k}$ : input activation of unit k for pattern p (usually ±1).

$a_{p,j}$ : activation of hidden unit j for pattern p.

$a_{p,i}$ : activation of output unit i for pattern p.

$t_{p,i}$ : target value of output unit i for pattern p (usually ±1).

$e_{p,i}$ : error at output unit i for pattern p.

$w_{j,i}$ : weight on connection between hidden unit j and output unit i.

$w_{k,j}$ : weight on connection between hidden unit j and input unit k.

$g_{p,i}$ : component of special error function, kerr.

ferr : the usual error function.

kerr : the new improved error function.

$\delta_{p,l}$ : the part of d ferr which depends on unit l (output or hidden).

$\gamma_{p,l}$ : the part of d kerr which depends on unit l (output or hidden).

$\beta_{l}$ : the bias associated with unit l (output or hidden).

This appendix presents the details of the two error functions discussed in the section on saddle points. Figure B-1 has the definitions of terms used in the discussion about the error functions.

The function v(x) is defined by:

$$v(x) = (1 - e^{-x}) / (1 + e^{-x}) \text{ and } v'(x) = (1 - v(x)^2) / 2.$$

Define $a_{p,j} = v(\sum_k w_{k,j} a_{p,k} + \beta_j)$, $a_{p,i} = v(\sum_j w_{j,i} a_{p,j} + \beta_i)$, and $e_{p,i} = t_{p,i} - a_{p,i}$.

The usual error function is $ferr = \sum_{p,i} e_{p,i}^2$. Define $\delta_{p,i} = e_{p,i}(1 - a_{p,i}^2)$ and $\delta_{p,j} = \sum_i \delta_{p,i} w_{j,i}(1 - a_{p,j}^2) / 2$. The derivatives of ferr which the msdm uses are:

$$\partial\, ferr / \partial w_{j,i} = - \sum_p \delta_{p,i}\, a_{p,j}\, ,\ \partial\, ferr / \partial w_{k,j} = - \sum_p \delta_{p,j}\, a_{p,k}\, ,$$

$$\partial\, ferr / \partial\, \beta_i = - \sum_p \delta_{p,i} \text{ and } \partial\, ferr / \partial\, \beta_j = - \sum_p \delta_{p,j}$$

The new improved error function is: $kerr = \sum_{p,i} g_{p,i}$ where $g_{p,i} = e_{p,i}^2 / (1 - a_{p,i}^2)$. Define $\gamma_{p,i} = e_{p,i} - g_{p,i} a_{p,i}$ and $\gamma_{p,j} = \sum_i \gamma_{p,i} w_{j,i}(1 - a_{p,j}^2) / 2$. The derivatives of kerr which are used to determine a descent direction are:

$$\partial\, kerr / \partial w_{j,i} = -\sum_p \gamma_{p,i}\, a_{p,j},\ \partial\, kerr / \partial w_{k,j} = -\sum_p \gamma_{p,j}\, a_{p,k}$$

$$\partial\, kerr / \partial\, \beta_i = - \sum_p \gamma_{p,i} \text{ and } \partial\, kerr / \partial\, \beta_j = - \sum_p \gamma_{p,j}$$

For computational purposes replace the 1 in the denominator of $g_{p,i}$ by $1 + \epsilon$ where $\epsilon$ is the machine epsilon. This will prevent divide by zero errors and $g_{p,i}$ will still have its desirable behavior for all practical purposes.

## APPENDIX C -- DETAILS OF THE CONJUGATE GRADIENT METHOD

This appendix presents the details of the conjugate gradient method. Figure B-1 defines some of the terminology used in the algorithms of the cgm. Figure C-1 defines the rest. Figure C-2 describes the algorithms of the cgm.

FIGURE C-1. TERMINOLOGY OF THE CONJUGATE GRADIENT METHOD

etol : main convergence criterion; based on "winner take all".

cgm driver : main procedure of cgm; controls termination.

$\hat{a}$ : set of activations; input, hidden and output.

$\hat{w}$ : set of weights and biases.

$\hat{e}$ : set of errors.

$\hat{x}$ : gradient vector with respect to weights and biases.

$\hat{z}$ : temporary vector used in computation of descent direction.

$\hat{h}$ : next direction of descent.

cgm : procedure for conjugate gradient method.

line minimization : procedure for method to find minimum along direction of descent.

mnbrak : procedure for method to find an initial interval which contains a minimum along direction of descent.

dbrent : procedure to find minimum along direction of descent; it requires an initial interval which contains a minimum.

$\lambda$ : distance along direction of descent from starting point to minimum.

$\gamma$ : coefficient used in computing new descent direction.

The value of etol determines convergence. The maximum value of $|e_{p,i}|$ is 2, which occurs when $a_{p,i} = -t_{p,i}$. Perfect training makes each $a_{p,i} = t_{p,i}$. However, perfect training is not necessary. If every $|e_{p,i}| < 1$ then every output activation will be closer to the desired value than to the wrong value. Under the "winner take all" strategy this convergence criteria will guarantee that the correct output will result for each training pattern. Computing activations and errors constitutes an epoch.

Further analysis shows that mnbrak can be replaced by a simpler adaptive bracketing algorithm which doubles the maximum step size whenever $\lambda$ in dbrent is greater than a certain tolerance times the current maximum step size. This simple substitution can save more than half of the

FIGURE C-2. BACK PROPAGATION CGM ALGORITHMS

```
begin   "back propagation neural network cgm -- main algorithm"
     input    network parameters
     create   network
     invoke   "cgm driver"
     report   results; e.g. weights, maximum error etc.
end

begin  "cgm driver"
     compute initial weights $\hat{w}$ $\{w_{j,i}, w_{k,j}\}$
     compute activations $\hat{a}$ $\{a_{p,i}, a_{p,j}, a_{p,k}\}$ and errors $\hat{e}$ $\{e_{p,i}\}$
     compute derivatives of kerr to get gradient $\hat{x}$
     initialize descent direction $\hat{z} \leftarrow -\hat{x}$, $\hat{h} \leftarrow \hat{z}$, $\hat{x} \leftarrow \hat{h}$
     repeat   invoke cgm   until $\max_{p,i} |e_{p,i}| < $ etol
end

begin  cgm
     invoke  "line minimization" along $\hat{x}$;
     set  $\hat{w} \leftarrow \hat{w} + \lambda \hat{x}$
     compute activations  $\hat{a}$ and errors $\hat{e}$.
     compute derivatives of kerr to get gradient $\hat{x}$
     compute  $\gamma \leftarrow (\hat{x} + \hat{z}) \hat{x} / |\hat{x}|^2$
     compute $\hat{z} \leftarrow -\hat{x}$, $\hat{h} \leftarrow \hat{z} + \gamma \hat{h}$,  $\hat{x} \leftarrow \hat{h}$
end

begin  "line minimization"
     invoke mnbrak;  mnbrak uses golden mean search to locate a
          region along the descent direction $\hat{x}$ that contains a
          minimum of kerr.
     invoke dbrent; dbrent uses quadratic interpolation to isolate
          the minimum just bracketed.  It finds $\lambda$ such that
          kerr($\hat{w} + \lambda \hat{x}$) is a minimum.
end
```

invocations of back propagation. A few more invocations of back propagation can be saved if the maximum step size is halved whenever $\lambda$ is less than another tolerance times the current maximum step size. Observations on moderately large problems indicate that a small initial maximum step size leads to fewer epochs needed for convergence.

## BIBLIOGRAPHY

[B] Roberto Battiti, Optimization Methods for Back-propagation: Automatic Parameter Tuning and Faster Convergence, Proceedings of the International Joint Conference on Neural Networks, January, 1990.

[F] Scott E. Fahlman, An Empirical study of Learning Speed in Back-Propagation Networks, Technical Report CMU-CS-88-162, Carnegie-Mellon University, 1988.

[FK] Kanaan A. Faisal and Stan C. Kwasny, Deductive and Inductive Learning in a Connectionist Deterministic Parser, Proceedings of the International Joint Conference on Neural Networks, January, 1990.

[KF] Stan C. Kwasny and Kanaan A. Faisal, Deterministic Parsing of Novel and Lexically Ambiguous Sentences, IJCAI-89 Workshop: Symbolic Problem Solving in Noisy, Novel and Uncertain Task Environments, 1989.

[LB] Andrew Laine and William Ball, Application of a New Space-Frequency Transform for Text Recognition, Technical Report WUCS-23-90, Washington University, 1990.

[M] Mitchell P. Marcus, A Theory Of Syntactic Recognition for Natural Language, MIT Press, 1980.

[PE] E. Polak, Computational Methods in Optimization: A Unified Approach, Academic Press, 1971.

[PFTV] William H. Press, Brian P. Flannery, Saul A. Teukolsky and William T. Vettering, Numerical Recipes in C, Cambridge University Press, 1988.

[PJ] Pollack, J.B. (in press), Recursive Distributed Representations.

[RM] David E. Rumelhart and James L. McClelland, Parallel Distributed Processing, MIT Press, 1986.