Neural Network Learning and Learning Control Through Optimization Techniques

Presented to Columbia University
in Partial Fulflillment for
the degree of Doctor of Engineering Science

Homayoon Sadr Mohammad Beigi

Copyright: Homayoon Beigi

February 1991

Chapter 1

Introduction

The objective of this research is to develop new learning methods based upon optimization techniques. Two different but related areas are focused on. The first is to develop new learning algorithms for neural networks. These new learning techniques could later be used in a variety of applications such as control and pattern recognition applications. The second is to develop new learning controllers which improve in their performance each time they are requested to repeat a task.

1.1. Introduction to Neural Network Learning

Neural net (NN) models have been studied for many years with the hope that the superior learning and recognition capability of the human brain could be emulated by man-made machines. Similar massive networks, in the human brain, make the complex pattern and speech recognition of humans possible. In contrast to the Von Neumann computers which compute sequentially, neural nets employ huge parallel networks of many densely interconnected computational elements called neurons. Neural networks have been used in many different applications such as Adaptive and Learning Control, Pattern Recognition, Image Processing, Signature Recognition, Signal Processing and Speech Recognition, Financial Problems, etc.

A neuron is the most elementary computational unit in a neural network which sums a number of weighted inputs and passes the result through a non-linear activation function. Multi-layer neural networks (figure 1.1) consist of a large number of neurons. Before a neural network can be used for any purpose, the weights connecting inputs to neurons and the parameters of the activation functions of neurons should be adjusted so that outputs of the network will match desired values for specific sets of inputs. The methods used for adjusting these weights and parameters are usually referred to as learning algorithms.

Rumelhart et al.[1] introduced a learning theory for multi-layer neural nets in 1986 and to some extent demonstrated that a general learning algorithm for multi-layer Neural Networks is possible. In a learning algorithm called the Back-Propagation technique, they used the so called generalized delta rule to calculate an approximate gradient vector which is then used by a steepest descent search to minimize the difference between the NN output and the desired output. However, as demonstrated by simulations obtained by Rumelhart et al., low rates of convergence were seen in practically every problem. Lippmann[2] states, "One difficulty noted by the Back-Propagation algorithm is that in many cases the number of presentations of training data required for convergence has been large (more than 100 passes through all the training data.)"

We have tried the usage of neural networks for developing learning control schemes for repetitive manufacturing processes. These processes include machining, milling, robotics, etc. In the neural networks approach to solving the learning control problem in for instance machining, a scrap piece has to be machined to provide the neural network with a training set of data. If a slow learning neural network is used, it would generate a great deal of waste in scrap pieces to achieve learning. This problem makes it imperative to develop new learning algorithms for neural networks with orders of magnitude of faster learning compared to the Back-Propagation technique. Also, in applications such as speech or hand writing recognition, the one percent error margin of the Back-Propagation learning is no longer acceptable and a more precise convergence would be desirable. Once new faster converging and more accurate learning algorithms are present, these problems and similar problems in other applications of neural networks could be alleviated, thus making neural networks much more practical.

In the literature, several methods[2-3] have been proposed to increase the rate of convergence of learning by making strong assumptions such as linearity for multi-layer networks. In addition, other more practical methods have recently been proposed for speeding the convergence of the Back-Propagation technique.[4-7]

The Back-Propagation technique is a special case of the steepest descent technique with some additional assumptions. In general, steepest descent techniques perform well while away from local minima and require many iterations to converge when close to the minima. On the other hand, Newton's method usually converges fast in the vicinity of the minima. In addition, Newton's minimization technique handles functions with ill-conditioned Hessian matrices elegantly [8]. It would be desirable to take advantage of the properties of steepest descent when the state is far from minimal and then to use Newton's method in the vicinity of the minimum.

To use Newton's method, the first gradient and the matrix of second partial derivatives (Hessian) matrix should be evaluated. The moment one talks about evaluating the Hessian matrix, it becomes clear that a layer by layer adjustment of the weights is not possible because there are elements of the Hessian which are related to neurons in different layers. In general, two difficulties have prohibited the use of Newton's method for neural network learning: 1) the complexity of the evaluation of the Hessian, and 2) the inversion of the Hessian. One way to alleviate these difficulties is to use a momentum method which would approximate the diagonal elements of the Hessian matrix and would stay ignorant of the off-diagonal elements. [9]

On the other hand, Quasi-Newton methods provide another solution to the problem, by providing an iterative estimate for the inverse of the Hessian matrix. If one

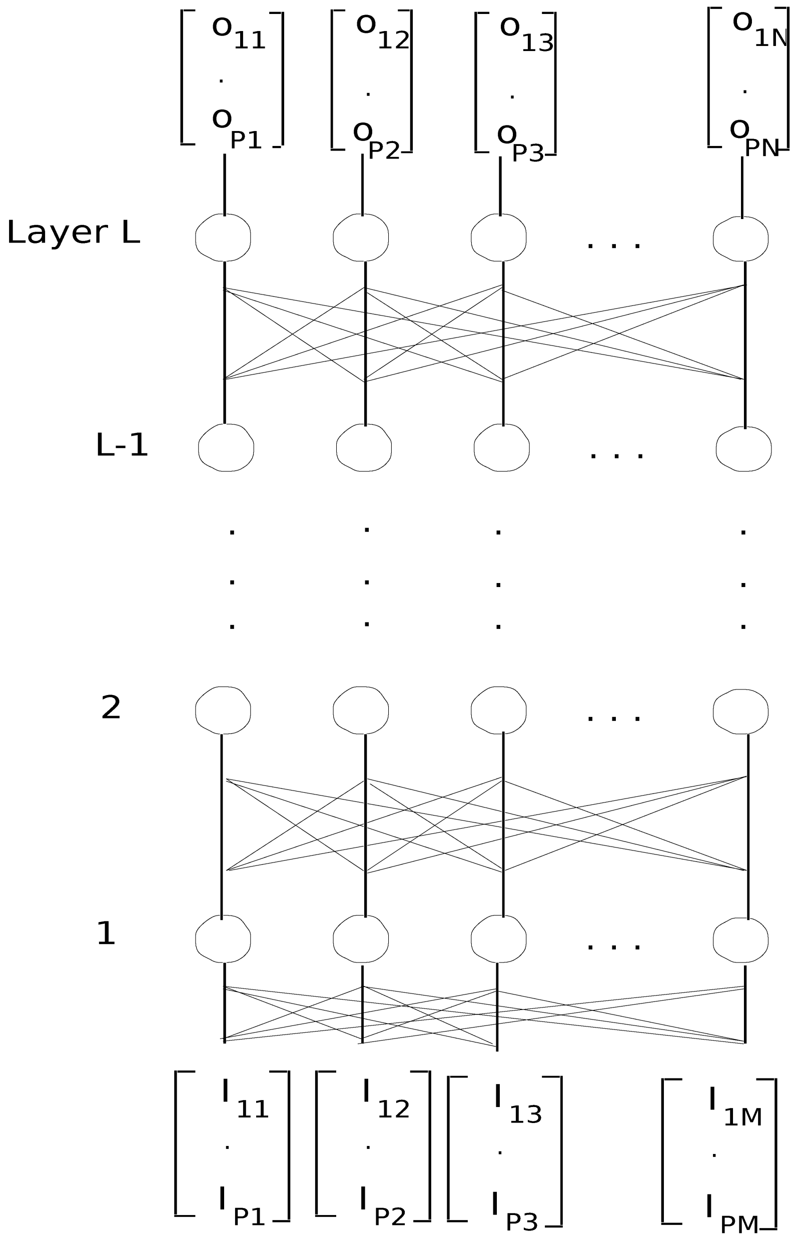
o_{11}
o_{P1}
o_{12}
o_{P2}
o_{13}
o_{P3}
o_{1N}
o_{PN}
Layer L
L-1
2
1
I_{11}
I_{P1}
I_{12}
I_{P2}
I_{13}
I_{P3}
I_{1M}
I_{PM}

figure 1.1 (General Multi-Layer Feed-Forward Neural Network) selects the initial estimate to be an identity matrix, initially, the method coincides with the steepest descent technique and gradually changes into Newton's method as the estimate approaches the inverse of the Hessian. This thesis will first review the state-of-the-art in Quasi-Newton and conjugate gradient methods and will develop new learning methods based on these minimization techniques, increasing the speed and accuracy of learning of feedforwad neural networks drastically. Then, these newly developed learning methods will be evaluated through simulations.

In many cases such as in control problems, one might try to train a neural network which might include one or more neurons with unknown activation functions, such as the case when an unknown nonlinear plant to be controlled is assumed to be a neuron among several other neurons for which the activation functions are known. In such cases, the whole network should be trained without any knowledge of the functions and gradients of individual neurons. In gradient based learning techniques, the gradient evaluation requires knowledge of the activation functions of all neurons including the controlled plant. In some other cases, activation functions are desired which might not have continuous derivatives. One such activation function is the threshold function or the step function which is thought to be used in biological neurons. For these, functions, gradient based techniques will fail. In these cases, the step function is approximated with a logistic or similar smooth function. These and some other special and very important problems could be alleviated if fast gradient-free learning techniques were available.

A series of gradient-free minimization schemes are reviewed and used to develop new learning algorithms for neural networks to be applicable to the aforementioned problems and to minimize the need for extra software/hardware. These methods do not require the evaluation of the gradient or Hessian of the objective function and only require the objective function evaluation which can be performed by the network itself. This property makes learning algorithms based on gradient-free techniques in some sense independent of the structure and connectivity of neurons in the hidden layers. An attractive consequence of this independence is that upon possible changes in the configuration, of the network, the network could still be trained in the same manner. As an example of this feature, consider a neural network with several neurons which uses some gradient learning technique. For this technique to be used, an algorithm should be developed which would include theoretical formulas for the gradient of the network. At once, however, a neuron becomes disabled or a connection is severed. The gradient based learning algorithm is no longer valid and cannot be used for learning. In this case, either the network should be thrown away or a new algorithm should be formulated for its new structure. The gradient-free techniques, however, would operate with either structure and no change is necessary to the network or to the learning algorithm. This is another practical feature which is sought by developing fast gradient-free learning techniques. The new learning methods will be evaluated through simulations.

1.2. Introduction to Discrete-Time Learning Control

A large percentage of the practical applications of control systems involve cases in which a system is repeatedly asked to perform the same task. Examples include tracking problems for robots on assembly lines, as well as a large number of manufacturing applications. Standard controller design methods produce systems that repeat the same errors everytime the command is repeated. It is a bit primitive to persist in repeating the same errors. In the last few years several theorems of learning control have been developed in the literature, generating controllers which can learn from previous experience at performing a specific task[10-15]. Two learning control algorithms are proposed here.

The first algorithm minimizes the output errors using a generalized secant method for recursive identification. In this approach, a time variant, discrete-time model of the control system is devised in the form of a system of linear algebraic equations relating the change in the state of the system to the change in the control action from one repetition of the task to another. This set of linear equations gives the transition between any two repetitions. This system is then solved for the appropriate control action that will minimize the tracking error of the controlled dynamic system, only requiring the availability of the order of the system, without any prior knowledge of the system parameters. This leads to a learning-adaptive controller, the performance of which is tested by applying it in simulation to controlling two different nonlinear plants.

In a second approach, a self-tuning (adaptive) controller based on the recursive least square parameter estimation is presented. The learning recursive least square parameter estimator applies the recursive least square parameter estimator, which is normally used in self-tuning regulators, in the repetition domain for each discrete time step. This will allow learning to be established. In applying this learning-adaptive controller, again, the order of the system should be known, but no prior knowledge of the system parameters is required. The performance of this control scheme is tested by simulation and experiment in controlling three different nonlinear plants.

1.3. Thesis Structure

This thesis consists of two major parts. In these two parts, the problems of learning in neural networks and learning control are treated, respectively.

Part I (chapters 2-6) presents several new learning schemes for neural networks. Chapter 2 formulates the neural network learning problem as a minimization problem. Chapter 3 applies gradient-based minimization techniques on this problem to develop fast new learning algorithms for the adjustment of neural network intercellular weights and activation function parameters. In chapter 4, more implementable NN learning algorithms are developed based on gradient-free minimization schemes. These methods are in general independent of the architecture of the neural networks. Chapters 5 and 6 discuss and conclude the simulation results conducted for these new learning algorithms. Simulation results show two to three orders of magnitude improvement on the rate of learning when compared with present state-of-the-art learning techniques.

Part II (chapters 7 and 8) presents two new learning control algorithms. In Chapter 7, an optimization approach is taken in solving the learning control problem. An objective function is formulated for the general time-variant learning control problem so that the tracking error of the controlled system can be minimized repetition after repetition. Using this objective function, a learning-adaptive control scheme is generated based on the Generalized Secant Method and is shown in simulation results, as applied to the control of two nonlinear dynamic systems performing highly nonlinear tasks, to perform very well. Chapter 8 uses a different approach for solving the learning control problem. A learning parameter estimator based on the recursive least squares algorithm is developed which in conjunction with a suitable control law forms a learning self-tuning regulator. Simulations of this learning parameter estimator have also been conducted on controlling the same nonlinear plants as in chapter 7 and have shown to be very effective. In addition, very good experimental results have been obtained in the application ot this learning control technique to the control of a Piezoelectric tool in a diamond cutting lathe with highly nonlinear dynamics including hysteresis.

Appendix A provides a summary of the mathematics required for the formulations of parts I and II. Finally, Appendix B provides a list of the abbreviations used in this thesis.

Part I: LEARNING IN NEURAL NETWORKS

Chapter 2

Objective Function and Gradient Vector Formulation for Learning in Feed-Forward Neural Networks

The objective of learning, in the problem of learning in neural networks, is to minimize the output error of the top (output) layer (layer L in figure 1.1) of a neural network over a set of P input-output patterns.

Define,

$l \in [1, L]$	(Layer number in the network)
$n_l \in [1, N_l]$	(Neuron number in layer l)
$p \in [1, P]$	(Pattern number)
ω^l_{nm}	(Weighting factor between the mth input and neuron n in layer l)
o^l_{pn}	(Output of neuron n of layer l for input pattern p)
t_{pn}	(Desired output of neuron n in layer L)
i_{pm}	(Input m of pattern p to the network)

Then, the objective of learning becomes,

$$\text{minimize}\, E = \sum_{p=1}^{P}\sum_{n_L=1}^{N_L}(o^L_{pn_L} - t_{pn_L})^2 \qquad (1) \qquad (2.1)$$

Define,

$$E_p = \sum_{n_L=1}^{N_L}(o^L_{pn_L} - t_{pn_L})^2 \qquad (2.2)$$

then,

$$E = \sum_{p=1}^{P} E_p \qquad (2) \qquad (2.3)$$

Let us define a few variables and eventually a state vector which would include all the variables to be optimized for minimum E:

$$\varphi^{lT} = [\, \varphi^{l}_{1}, \varphi^{l}_{2}, \dots , \varphi^{l}_{N_l}] \qquad \text{(Activation function parameter vector for level } l\text{)}$$

$$\omega^{lT}_{n_l} = [\, \omega^{l}_{n_l 1}, \omega^{l}_{n_l 2}, \dots , \omega^{l}_{n_l N_{l+1}}] \qquad \text{(Vector of intercellular weights to neuron } n_l \text{ at layer } l\text{)}$$

$$\omega^{lT} = [\, \omega^{lT}_{1} , \omega^{lT}_{2} , \dots , \omega^{lT}_{N_l}] \qquad \text{(Supervector of intercellular weights of level } l\text{)}$$

$$x^{lT} = [\, \varphi^{lT} , \omega^{lT}] \qquad \text{(State vector for level } l\text{)}$$

$$x^{T} = [\, x^{1T} , x^{2T} , \dots , x^{LT}] \qquad \text{(Super state vector)}$$

Let j denote any element of the state vector x, then,

$$\frac{\partial E}{\partial x_j} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial x_j} \qquad (3) \qquad (2.4)$$

and by the chain rule,

$$\frac{\partial E_p}{\partial x_j} = 2 \sum_{n_L=1}^{N_L} (o^{L}_{pn_L} - t_{pn_L}) \frac{\partial o^{L}_{pn_L}}{\partial x_j} \qquad (4) \qquad (2.5)$$

where,

$$\frac{\partial o^{L}_{pn_L}}{\partial \varphi^{l}_{n_l}} = \frac{\partial o^{L}_{pn_L}}{\partial o^{l}_{pn_l}} \frac{\partial o^{l}_{pn_l}}{\partial \varphi^{l}_{n_l}} \qquad (5a) \qquad (2.6a)$$

$$\frac{\partial o^{L}_{pn_L}}{\partial \omega^{l}_{n_l n_{l+1}}} = \frac{\partial o^{L}_{pn_L}}{\partial o^{l}_{pn_l}} \frac{\partial o^{l}_{pn_l}}{\partial \omega^{l}_{n_l n_{l+1}}} \qquad (5b) \qquad (2.6b)$$

and,

$$\frac{\partial o^{L}_{pn_L}}{\partial o^{l}_{pn_l}} = \prod_{i=l+1}^{L} \frac{\partial o^{(L+l-i+1)}_{pn_{(L+l-i+1)}}}{\partial o^{(L+l-i)}_{pn_{(L+l-i)}}} \qquad \text{(using the index notation)} \qquad (6)$$

(using indicial notation) (2.7)

In equation (2.7) the index notation has been employed, i.e.,

$$\frac{\partial o^{l+1}_{p\eta_{l+1}}}{\partial o^{l}_{p\eta_l}}\frac{\partial o^{l}_{p\eta_l}}{\partial o^{l-1}_{p\eta_{l-1}}} \longleftrightarrow \sum_{\eta_l=1}^{N_l}\frac{\partial o^{l+1}_{p\eta_{l+1}}}{\partial o^{l}_{p\eta_l}}\frac{\partial o^{l}_{p\eta_l}}{\partial o^{l-1}_{p\eta_{l-1}}} \text{ (traditional notation)}$$

Take, for example, the logistic function,

$$o^{l}_{p\eta_l}=\frac{1}{1+e^{-s^{l}_{p\eta}}} \qquad (7) \qquad (2.8)$$

where,

$$s^{l}_{p\eta_l}=\sum_{\eta_{l-1}=1}^{N_{l-1}}\left(o^{l-1}_{p\eta_{l-1}}\,\omega^{l}_{\eta_l\eta_{l-1}}\right)+\varphi^{l}_{\eta_l} \qquad (2.9)$$

then,

$$\frac{\partial o^{l}_{p\eta_l}}{\partial \varphi^{l}_{\eta_l}}=d^{l}_{p\eta_l} \qquad (8) \qquad (2.10)$$

$$\frac{\partial o^{l}_{p\eta_l}}{\partial \omega^{l}_{\eta_l\eta_{l-1}}}=d^{l}_{p\eta_l}\,o^{l-1}_{p\eta_{l-1}} \qquad (9) \qquad (2.11)$$

and,

$$\frac{\partial o^{l}_{p\eta_l}}{\partial o^{l-1}_{p\eta_{l-1}}}=d^{l}_{p\eta_l}\,\omega^{l}_{\eta_l\eta_{l-1}} \qquad (10) \qquad (2.12)$$

where,

$$d^{l}_{p\eta_l}=\frac{e^{-s^{l}_{p\eta}}}{\left(1+e^{-s^{l}_{p\eta}}\right)^{2}} \qquad (11) \qquad (2.13)$$

However,

$$o^{l}_{pn_l} = \frac{1}{1 + e^{-s^{l}_{pn}}} \qquad (2.14)$$

from which,

$$e^{-s^{l}_{pn}} = \frac{1}{o^{l}_{pn_l}} - 1 \qquad (12) \qquad (2.15)$$

Rewrite equation (2.13) using (2.15),

$$d^{l}_{pn_l} = o^{l}_{pn_l}(1 - o^{l}_{pn_l}) \qquad (13) \qquad (2.16)$$

From (2.16), (2.10), (2.11), and (2.12),

$$\frac{\partial o^{l}_{pn_l}}{\partial \varphi^{l}_{n_l}} = o^{l}_{pn_l}(1 - o^{l}_{pn_l}) \qquad (14) \qquad (2.17)$$

$$\frac{\partial o^{l}_{pn_l}}{\partial \omega^{l}_{n_l n_{l-1}}} = o^{l-1}_{pn_{l-1}} o^{l}_{pn_l}(1 - o^{l}_{pn_l}) \qquad (15) \qquad (2.18)$$

and,

$$\frac{\partial o^{l}_{pn}}{\partial o^{l-1}_{pn_{l-1}}} = \omega^{l}_{n_l n_{l-1}} o^{l}_{pn_l}(1 - o^{l}_{pn_l}) \qquad (16) \qquad (2.19)$$

We can therefore use equations 2.17-2.19 and 2.4-2.7 to evaluate the elements of gradient of E with respect to the super state vector x (—$_x$E). Define,

$$g = \nabla_x E = \sum_{p=1}^{P} \nabla_x E_p \qquad \text{(Gradient Vector)} \qquad (2.20)$$

and,

$$G = \nabla_x^2 E = \sum_{p=1}^{P} \nabla_x^2 E_p \qquad \text{(Hessian Matrix)} \tag{2.21}$$

Most minimization techniques require gradient evaluations of the objective function. The above formulation is used in most of the minimization techniques discussed in this thesis.

Example: The Exclusive-OR (XOR) Problem

Take the example of a network of three neurons (figure 2.1) which is employed to simulate the exclusive-OR (XOR) logic pattern (table 2.1.)

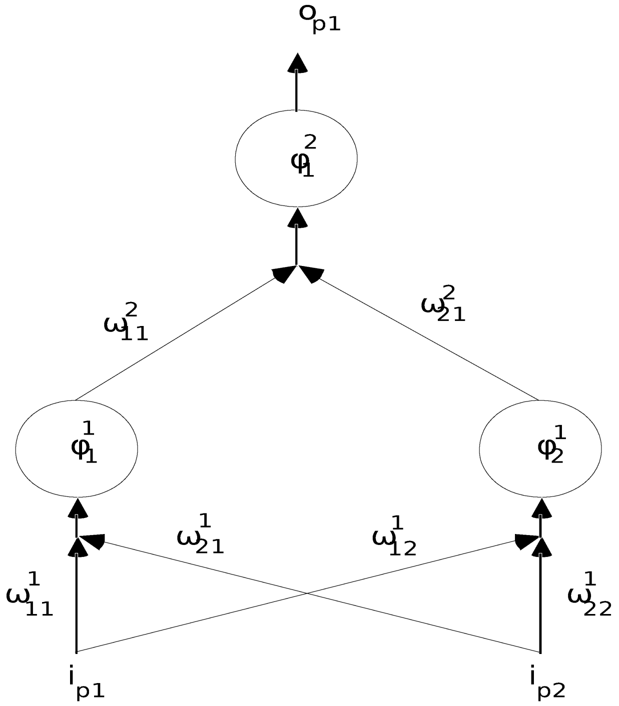
o_{p1}
φ_1^2
ω_{11}^2
ω_{21}^2
φ_1^1
φ_2^1
ω_{21}^1
ω_{12}^1
ω_{11}^1
ω_{22}^1
i_{p1}
i_{p2}

figure 2.1 (Neural Network Simulating the Exclusive-OR Logic)

Input Patterns		Output Patter
0	0	0
1	0	1
0	1	1
1	1	0

table 2.1 (XOR Logic)

The objective function of minimization will then be,

$$E = \sum_{p=1}^{4} (o_{p1}^{2} - t_{p1})^{2} \tag{2.22}$$

and the super state vector is defined by the following sequence of definitions,

$$\varphi^{1T} = [\, \varphi_1^1, \varphi_2^1 \,]$$

$$\varphi^{2} = [\, \varphi_1^2 \,]$$

$$\omega^{1T} = [\, \omega_{11}^1, \omega_{12}^1, \omega_{21}^1, \omega_{22}^1 \,]$$

$$\omega^{2T} = [\, \omega_{11}^2, \omega_{12}^2 \,]$$

$$x^{T} = [\, \varphi_1^1, \varphi_2^1, \omega_{11}^1, \omega_{12}^1, \omega_{21}^1, \omega_{22}^1, \varphi_1^2, \omega_{11}^2, \omega_{12}^2 \,]$$

From equation (2.5),

$$\frac{\partial E_p}{\partial x_j} = 2\,(\, o_{p1}^{2} - t_{p1}\,)\,\frac{\partial o_{p1}^{2}}{\partial x_j} \tag{2.23}$$

and from equations 2.17-2.19,

$$\frac{\partial o_{p1}^{2}}{\partial \varphi_1^{2}} = o_{p1}^{2}\,(\, 1 - o_{p1}^{2}\,) \tag{2.24}$$

$$\frac{\partial o_{p1}^{2}}{\partial \omega_{11}^{2}} = o_{p1}^{1}\, o_{p1}^{2}\,(\, 1 - o_{p1}^{2}\,) \tag{2.25}$$

$$\frac{\partial o_{p1}^{2}}{\partial \omega_{12}^{2}} = o_{p2}^{1}\, o_{p1}^{2}\, (1 - o_{p1}^{2}) \tag{2.26}$$

$$\frac{\partial o_{p1}^{2}}{\partial \varphi_{1}^{1}} = \frac{\partial o_{p1}^{2}}{\partial o_{p1}^{1}} \frac{\partial o_{p1}^{1}}{\partial \varphi_{1}^{1}} = \omega_{11}^{2}\, o_{p1}^{2}\, (1 - o_{p1}^{2})\, o_{p1}^{1}\, (1 - o_{p1}^{1}) \tag{2.27}$$

$$\frac{\partial o_{p1}^{2}}{\partial \varphi_{2}^{1}} = \frac{\partial o_{p1}^{2}}{\partial o_{p2}^{1}} \frac{\partial o_{p2}^{1}}{\partial \varphi_{2}^{1}} = \omega_{12}^{2}\, o_{p1}^{2}\, (1 - o_{p1}^{2})\, o_{p2}^{1}\, (1 - o_{p2}^{1}) \tag{2.28}$$

$$\frac{\partial o_{p1}^{2}}{\partial \omega_{11}^{1}} = \frac{\partial o_{p1}^{2}}{\partial o_{p1}^{1}} \frac{\partial o_{p1}^{1}}{\partial \omega_{11}^{1}} = \omega_{11}^{2}\, o_{p1}^{2}\, (1 - o_{p1}^{2})\, i_{p1}\, o_{p1}^{1}\, (1 - o_{p1}^{1}) \tag{2.29}$$

$$\frac{\partial o_{p1}^{2}}{\partial \omega_{12}^{1}} = \frac{\partial o_{p1}^{2}}{\partial o_{p1}^{1}} \frac{\partial o_{p1}^{1}}{\partial \omega_{12}^{1}} = \omega_{11}^{2}\, o_{p1}^{2}\, (1 - o_{p1}^{2})\, i_{p2}\, o_{p1}^{1}\, (1 - o_{p1}^{1}) \tag{2.30}$$

$$\frac{\partial o_{p1}^{2}}{\partial \omega_{21}^{1}} = \frac{\partial o_{p1}^{2}}{\partial o_{p2}^{1}} \frac{\partial o_{p2}^{1}}{\partial \omega_{21}^{1}} = \omega_{12}^{2}\, o_{p1}^{2}\, (1 - o_{p1}^{2})\, i_{p1}\, o_{p2}^{1}\, (1 - o_{p2}^{1}) \tag{2.31}$$

$$\frac{\partial o_{p1}^{2}}{\partial \omega_{22}^{1}} = \frac{\partial o_{p1}^{2}}{\partial o_{p2}^{1}} \frac{\partial o_{p2}^{1}}{\partial \omega_{22}^{1}} = \omega_{12}^{2}\, o_{p1}^{2}\, (1 - o_{p1}^{2})\, i_{p2}\, o_{p2}^{1}\, (1 - o_{p2}^{1}) \tag{2.32}$$

Using the above equations, the elements of —$_x$E$_p$ and using equation (2.4) the elements of —$_x$E are found.

Chapter 3

Gradient-Based Minimization Algorithms

This Chapter describes a set of gradient based minimization techniques which require the evaluation of the objective function, its gradient and sometimes the matrix of its second partial derivatives (Hessian) for certain states. Another group of minimization techniques are described in chapter 4 which require evaluations of the objective function only. In the latter methods, no direct gradient evaluations are needed.

3.1. The Steepest Descent Technique

The steepest descent technique is a gradient-minimization method which uses a first order approximation to the objective function (E) to generate directions of descent with the knowledge of the gradient at each iteration k. The steepest descent method is very reliable since it always provides a descent direction. This technique is ideal for points which are far away from the local minima. However, close to the local minima, the steepest descent technique will generally require lots of iterations to converge due to its nature of approximating the objective function with a linear function.

Let x^* denote the state vector which results in a minimum objective function E. Further, assume that the current state is x_k and define Δx_k to be the difference between the states at k and k+1 iterations, namely,

$$\Delta x_k = x_{k+1} - x_k \qquad (17) \qquad (3.1)$$

Decompose the step Δx_k into a direction s_k and a magnitude λ_k,

$$\Delta x_k = \lambda_k s_k \qquad (18) \qquad (3.2)$$

Since the gradient of E points in the ascent direction, the steepest descent is given by the direction opposite to the gradient direction at every step k. Therefore for steepest descent minimization, However, a line search should be used to provide an optimum step size λ_k^* to minimize E in the s_k direction. This will produce the following recursive steepest descent technique for minimization of the sum of squares of errors of the output of a neural network.

$$x_{k+1} = x_k - \lambda_k^* \frac{g_k}{\|g_k\|} \qquad (20) \qquad (3.4)$$

The steepest descent technique of equation (3.4) is different from the Back-Propagation[1] technique in the sense that it uses the true gradient of E at step k and not an approximation. The Back-Propagation technique finds a descent direction based on an approximation to the gradient of the objective function by using only local data in a multi-layer network. Furthermore, the early implementations of the Back-Propagation technique took steps based on the gradient of E_p for each pattern p as soon as a new pattern was presented. The true steepest descent is done, however, by summing the gradients —E_p over all the P patterns and taking a final step in the direction opposite to the gradient of E, —E. The steepest descent method only provides good minimizing directions if the condition number of the Hessian of the objective function is close to 1. As this condition number gets larger, the steepest descent becomes slower and more advanced minimization techniques (preferably with quadratic convergence) will be preferred. To further enhance the speed of learning of the network, let us consider the following more advanced minimization techniques.

3.2. Newton's Minimization Technique

Write the Taylor series expansion of E at x^* and about x_k assuming that every update of the state vector should drive the state vector to optimal value x^*, namely, $x^* = x_k + \Delta x_k$.

```
ERROR:
ioerror
OFFENDING COMMAND:
image
STACK:
-mark-
-mark-
-mark-
```