Divide and Conquer Strategies for MLP Training

Smriti Bhagat, *Student Member, IEEE*, Dipti Deodhare

***Abstract*— Over time, neural networks have proven to be extremely powerful tools for data exploration with the capability to discover previously unknown dependencies and relationships in the data sets. However, the sheer volume of available data and its dimensionality makes data exploration a challenge. Employing neural network training paradigms in such domains can prove to be prohibitively expensive. An algorithm, originally proposed for supervised on-line learning, has been improvised upon to make it suitable for deployment in large volume, high-dimensional domains. The basic strategy is to divide the data into manageable subsets or blocks and maintain multiple copies of a neural network with each copy training on a different block. A method to combine the results has been defined in such a way that convergence towards stationary points of the global error function can be guaranteed. A parallel algorithm has been implemented on a Linux-based cluster. Experimental results on popular benchmarks have been included to endorse the efficacy of our implementation.**

I. Introduction

Huge data sets create combinatorially explosive search spaces for model induction. Traditional neural network paradigms can prove to be computationally very expensive for such domains. In this paper, an attempt has been made to improvise existing technology and explore the possibility of deploying it more easily for data exploration tasks. The basic strategy is described in [9]. It was originally proposed for supervised on-line learning. On-line learning (also known as incremental gradient methods [4]) operate only on the error contributed by a single input (or a subset of inputs) as against learning algorithms that operate on the global error function. To make this more precise, let P be the total number of patterns in the data set. Let $W \in \Re^w$ denote the neural network parameters (weights and biases). w is the total number of weights and biases in the neural network. Let $(\mathbf{x}_l, \mathbf{d}_l),\ l = 1, \ldots, P$ be the set of training patterns for the network where $\mathbf{x}_l \in \Re^n$ and $\mathbf{d}_l \in \Re^m$. Let $\mathbf{y}_l$ be the m dimensional output of the network corresponding to input $\mathbf{x}_l$. Define the error that the neural network makes on input $\mathbf{x}_l$ as

$$E_l(W) = \frac{1}{2m} \parallel \mathbf{d}_l - \mathbf{y}_l \parallel_2^2,$$

where the l_2 norm is defined as,

$$\parallel \mathbf{x} \parallel_2^2 = \sum_{a=1}^{n} (x_a)^2.$$

Smriti Bhagat is doing her PhD at the Department of Computer Science, Rutgers University, New Jersey 08854, USA. (email: smbhagat@rutgers.edu)

Dipti Deodhare is with the Centre for Artificial Intelligence and Robotics, Bangalore, INDIA. (email: dipti@cair.res.in)

Further define,

$$E(W) = \frac{1}{P} \sum_{l=1}^{P} E_l(W).$$

Simply stated, batch methods would compute the next step in the iteration based on gradient information calculated on E whereas an on-line method would compute it based on a single $E_l,\ l \in \{1, \ldots, P\}$.

The basic strategy proposed in [9] is to divide the data into manageable subsets or blocks and then combine the result. The algorithm maintains multiple copies of the neural network. Each neural network trains on a different block of data. It has been shown that suitable training algorithms can be defined in such a way that the disagreement between the different copies of the network is asymptotically reduced and convergence towards stationary points of the global error function can be guaranteed. In the following Sections we describe the basic algorithms and our improvisations to them to make them suitable for large volume, high dimensional domains. The implementation has been tested on popular benchmarks.

II. The Block Training Algorithm

In this Section we describe in some detail the general class of block training algorithms described in [9] for online learning.

A. Defining the Exact Augmented Lagrangian

We have a large volume, n-dimensional set of P data points and corresponding to each data point an m dimensional vector indicating the desired neural network output. Let us assume that this training set is divided into N data blocks, $D_1, D_2, ..., D_N,\ N \leq P$. Therefore, $D_i \subseteq \Re^n \times \Re^m$ where n is the dimension of the input space and m is the dimension of the output space. Let,

$$F_j(W) = \frac{1}{P} \sum_{(\mathbf{x}_l, \mathbf{d}_l) \in D_j} E_l(W).$$

Then we can express $E(W)$ as follows:

$$E(W) = \sum_{j=1}^{N} F_j(W).$$

The supervised learning problem can now be rewritten as:

$$\begin{aligned} & \min \textstyle\sum_{j=1}^{N} F_j(W) \\ \text{subject to} \quad & V_j - U = 0, \qquad j = 1, \ldots, N \\ \text{where} \quad & U \in \Re^w, V_j \in \Re^w, \quad j = 1, \ldots, N. \end{aligned} \tag{1}$$

Here, U and $V_j,\ j = 1, \ldots, N$ are the $N + 1$ copies of the weight vector W. This is a constrained minimization problem

and the Lagrangian function that we can define corresponding to it is

$$L(V, \lambda, U) := \sum_{j=1}^{n} F_j(V_j) + \sum_{j=1}^{N} \lambda_j^T (V_j - U). \quad (2)$$

Here, $\lambda_j \in \Re^w$ are the vectors of Lagrangian multipliers. V and λ are matrices such that V_j and λ_j are the j^{th} column vectors respectively, $j = 1, \ldots, N$. To solve this problem the *alternating direction method* of multipliers described in [3] exists under convexity assumption on F_j. The method has also been studied for the general non-convex case in [2]. However, it is difficult to extend this analysis to the problem on hand. To solve the problem defined in expression 1 a single unconstrained problem equivalent to expression 1 is obtained by defining the augmented Lagrangian function denoted by $\Phi : \Re^{wN} \times \Re^{wN} \times \Re^w \rightarrow \Re$ and including *penalty terms* $\mathbf{c} \in \Re^N$.

$$\Phi(V, \lambda, U; \mathbf{c}) = \sum_{j=1}^{N} \left(F_j(V_j) + \Pi_j(V_j, \lambda_j, U; c_j) \right). \quad (3)$$

Here,

$$\begin{aligned} \Pi_j \quad := \quad & \lambda_j^T (V_j - U) + \left[c_j + \tau \parallel \lambda_j \parallel^2 \right] \parallel V_j - U \parallel^2 + \\ & \eta \parallel \bigtriangledown F_j(V_j) + \lambda_j \parallel^2 . \end{aligned} \quad (4)$$

c_j, τ, η are positive scalar parameters. The second term in equation 4 penalizes the equality constraints and also bounds the growth of the multipliers λ_j. The third term in equation 4 is a penalty term on the equations,

$$\bigtriangledown F_j(V_j) + \lambda_j = 0, j = 1, \ldots, N, \quad (5)$$

which would follow if we impose that the partial derivative of the Lagrangian L as defined in equation 2 be zero with respect to $V_j,\ j = 1, \ldots, N$. The augmented Lagrangian defined in equation 3, taking into consideration the structure of the supervised learning problem on hand is described as an exact augmented Lagrangian in the literature [9]. This is because it can be shown that the solution of the constrained problem, together with the associated multipliers constitutes a minimum point rather than a saddle point of Φ. (See [12], [13] and the references therein.) Let,

$$\phi_j(V_j, \lambda_j, U; c_j) := F_j(V_j) + \Pi_j(V_j, \lambda_j, U; c_j). \quad (6)$$

We can now define the new unconstrained minimization problem as

$$\min_{(V, \lambda, U)} \Phi(V, \lambda, U; \mathbf{c}) = \sum_{j=1}^{N} \phi_j(V_j, \lambda_j, U; c_j). \quad (7)$$

Note that redefinition of the problem as above, gives us the advantage that if U is kept fixed the unconstrained minimization problem decomposes into N independent sub-problems. An iterative algorithm to solve Φ can be described. The details are given in the Appendix. Having established a theoretical framework for handling the given problem as independent sub-problems, one can attempt various algorithms for handling the sub-problems. In particular, **Step 2** of the algorithm in the Appendix involves solving a non-linear optimization problem. It is of interest to study the consequence of using various non-linear optimization techniques for solving the sub-problem of **Step 2** on the overall solution. In this paper, we present the solutions obtained using the method of steepest descent and the method of dynamic tunneling discussed in the next section.

III. The Dynamic Tunneling Technique

As already mentioned above, good non-linear optimization technique is necessary to handle **Step 2** in the algorithm described in the Appendix. For this step we have experimented with the dynamic tunneling optimization technique for training MLPs. The technique is based on TRUST (Terminal Repeller Unconstrained Subenergy Tunneling) proposed by Cetin *et. al.* [6]. The method was further generalized to lower semi-continuous functions by Barhen *et. al.* [1]. This technique was adapted for training MLPs by Pinaki *et. al.* in [14]. The computational scheme described in [14] and used by us in our experiments is described below.

The scheme comprises two phases. In the first phase the well known and commonly used error backpropagation algorithm is used to obtain a minimum which is guaranteed only to be a local minimum [15], [10]. In the second phase, to detrap the system from the point of local minimum the dynamic tunneling technique is employed. Since the process of error backpropagation is well understood we only give details of Phase 2 of the computational scheme. For this consider the dynamical system given below:

$$\frac{dx}{dt} = g(x). \quad (8)$$

An equilibrium point $\mathbf{x}_{eq}$ is termed as an attractor (repeller) if no (at least one) eigenvalue of the matrix

$$A = \frac{\partial g(\mathbf{x}_{eq})}{\partial \mathbf{x}} \quad (9)$$

has a positive real part [1]. Dynamical systems that obey the Lipschitz condition

$$\left\| \frac{\partial g(\mathbf{x}_{eq})}{\partial \mathbf{x}} \right\| < \infty \quad (10)$$

are guaranteed that a unique solution exists for each initial point $\mathbf{x}_0$. Typically, such systems have an infinite relaxation time to an attractor and an infinite escape time from a repeller. When the Lipschitz condition is violated singular solutions are imposed in such a way that each solution approaches an attractor or escapes from a repeller in *finite time*. The core concept behind the dynamic tunneling is the violation of the Lipschitz condition at an equilibrium point of the system. If any particle is placed at a small perturbation point from an equilibrium point that violates the Lipschitz condition and is a repeller, it will move away from this point to another point within a finite amount of time. To understand this, consider the dynamical system given by the differential equation

$$\frac{dx}{dt} = x^{\frac{1}{3}}. \quad (11)$$

This system has an equilibrium point at $x = 0$ which violates the Lipschitz condition at $x = 0$. This is because,

$$\left| \frac{d}{dx}\left(\frac{dx}{dt}\right) \right| = \left| \frac{1}{3} x^{-\frac{2}{3}} \right| \rightarrow \infty \text{ as } x \rightarrow 0$$

The system has a repelling equilibrium point at $x = 0$. This can be verified taking into cognisance equation 10 above and the related discussion. This implies that any initial point that lies infinitesimally close to the repeller point $x = 0$ will escape the repeller and reach a new point y in finite time given by

$$t = \int x^{\frac{1}{3}} dx = \frac{3}{4} y^{\frac{4}{3}}$$

.

Note that in Step2 of the algorithm given in Section we need to find a descent direction d_j^k and a stepsize α_j^k along this direction so that as given in equation 18,

$$\phi_j(V_j^k + \alpha_j^k d_j^k, \lambda_j^{k+1}, U^k; c_j^k) \leq \phi_j(V_j^k, \lambda_j^{k+1}, U^k; c_j^k).$$

We need to minimize the objective function given by Φ in equation 3 and the relevant expressions for the partial derivatives of Φ with respect to the various variables are:

$$\bigtriangledown_U \Phi = -\sum_{j=1}^{N} \left(\lambda_j + 2(c_j + \tau \parallel \lambda_j \parallel^2)(V_j - U) \right), \quad (12)$$

$$\begin{aligned} \bigtriangledown_{\lambda_j} \Phi &= \bigtriangledown_{\lambda_j} \phi_j \\ &= V_j - U + 2\tau\lambda_j \parallel V_j - U \parallel^2 \\ &= 2\eta(\bigtriangledown F_j(V_j) + \lambda_j), \end{aligned} \quad (13)$$

$$\begin{aligned} \bigtriangledown_{V_j} \Phi &= \bigtriangledown_{V_j} \phi_j \\ &= \bigtriangledown F_j(V_j) + \lambda_j + \\ &\quad 2(c_j + \tau \parallel \lambda_j \parallel^2)(V_j - U) + \\ &\quad 2\eta \bigtriangledown^2 F_j(V_j)(\bigtriangledown F_j(V_j) + \lambda_j). \end{aligned} \quad (14)$$

Here $\bigtriangledown F_j$ represents the gradient of F_j which is nothing but the error function of the j^{th} copy of the neural network. $\bigtriangledown^2 F_j$ is the Hessian. To an approximation, for a sufficiently small step t,

$$\bigtriangledown^2 F_j(V_j)(\bigtriangledown F_j(V_j + \lambda_j) \simeq \frac{1}{t}(\bigtriangledown F_j(V_j + tz_j) - \bigtriangledown F_j(V_j)), \quad (15)$$

where $z_j = \bigtriangledown F_j(V_j) + \lambda_j$. This is the approximation we use in our code implementation.

We now proceed to discuss how dynamic tunneling is utilized to reduce the objective function Φ. Note that the update with respect to λ has already been given as an expression in equation 17. After computing $\phi_j(V_j^k, \lambda_j^{k+1}, U^k; c_j^k)$ we now work with the weights V_j^k to obtain the desired reduction in ϕ_j. The error backpropagation algorithm is used so that the weights in each block j are updated to take a small, fixed step in the negative direction of the gradient of ϕ_j. This is Phase 1 of the computational scheme as already mentioned. At the end of this phase we have a set of weights V_j^* representing the solution point. The solution $\phi_j(V_j^*, \lambda_j^{k+1}, U^k; c_j^k)$ is likely to be a local minimum. To detrap the solution from the local minimum we perform dynamic tunneling so that the system moves to a new point V_j in the weight space from where Phase 1 of the computational scheme can be initiated all over again. For convenience, let v_{pq}^r be a component of V_j representing the weight on the arc connecting node p in layer r and node q, where q is a node in a layer above layer r in the MLP. The following differential equation is used to implement the tunneling:

$$\frac{dv_{pq}^r}{dt} = \rho(v_{pq}^r - v_{pq}^{r*})^{\frac{1}{3}}. \quad (16)$$

This expression is similar to expression 11 discussed above. Here v_{pq}^{r*} is a component of V_j^*. Tunneling involves perturbing a single weight by a small amount ϵ_{pq}^r where $|\epsilon_{pq}^r| \ll 1$. The equation 16 is integrated for a fixed amount of time t with a small time-step $\triangle t$. After every time-step, the value of $F_j(V_j)$ is computed where V_j is the set of weights obtained by replacing v_{pq}^{r*} in V_j^* with the current value of v_{pq}^r. Tunneling comes to a halt when $F_j(V_j) \leq F_j(V_j^*)$. Consequent to this, the system re-enters Phase 1 to re-start the process of error backpropagation with the new set of weights as a starting point wherein only a single value v_{pq}^r has been perturbed by an appropriate amount, the rest of the weights remaining the same as in V_j^*. If the condition, $F_j(V_j) \leq F_j(V_j^*)$ is never satisfied a new weight is considered for tunneling. This process is repeated till all the components v_{pq}^{r*} of V_j^* have been considered. If for no v_{pq}^{r*} the tunneling leads to a point where $F_j(V_j) \leq F_j(V_j^*)$, V_j^* is retained. In this way, by a repeated application of gradient descent using error backpropagation and tunneling in the weight space the system may be led to a good solution.

IV. Parallel Implementation on a Linux Cluster

Applying neural network classifiers for classifying a large volume of high dimensional data is a difficult task as the training process is computationally expensive. A parallel implementation of the neural network training paradigms offers a feasible solution to the problem. Linux clusters are fast becoming popular platforms for the development of parallel and portable programs. Establishing a Linux cluster involves connecting computers running the Linux operating system with an appropriate network switch and then installing the requisite parallel libraries on each of them. This method is basically a way of establishing a loosely coupled parallel computing environment in which different processes communicate with each other by means of messages. Message passing is a paradigm widely used on parallel machines since it can be efficiently and portably implemented. This way of developing parallel programs has caught the attention of many application developers as it offers a cost effective solution.

A cluster of 17 personal computers working with the Linux (Debian GNU/Linux) operating system was established to carry out the implementation. Connectivity between the computers was achieved via a 3-Com switch and the Ethernet protocol. As already mentioned Message Passing Interface (MPI) is a paradigm that provides the facility to develop

parallel and portable algorithms. An MPI program consists of autonomous processes, executing their own code, in an MIMD style, as described in [8]. The codes executed by each process need not be identical. The processes communicate via calls to MPI communication primitives. Typically, each process executes in its own address space, although shared-memory implementations of MPI are possible. LAM stands for Local Area Multi-computer and is an implementation of the MPI standard. It is a parallel processing environment and development system, described in [5], for a network of independent computers. It features the MPI programming standard for developing parallel programs. The LAM MPI parallel libraries were installed on each computer in the cluster to implement parallel constructs based on MPI.

One of these computers was designated as the *master*. The master monitors the overall execution of the application program. The rest of the computers were designated as *slave nodes*. Essentially, a setup consisting of a master-slave environment with 16 slave nodes was established.

V. Implementing the Method of Steepest Descent

In training an MLP one models the problem as a nonlinear optimization problem. The general strategy in solving such a problem requires ascertaining a direction and taking a step in this direction. The acceptability of the choice of direction and stepsize is constrained by the requirement that the process should ensure a "sufficient decrease" in the objective function and a "sufficient displacement" from the current point [11], [7]. In **Step 2** of the algorithm in the Appendix we need to perform a nonlinear optimization for each objective function $\phi_j(V_j^k, \lambda_j^{k+1}, U^k; c_j^k)$. In the method of steepest descent, the direction is chosen to be the negative of the gradient of the objective function *i.e.* in our case we have,

$$- \bigtriangledown_{v_j} \phi_j(V_j^k, \lambda_j^{k+1}, U^k; c_j^k) = d_k^j.$$

Here d_k^j is the direction chosen at the k^{th} iteration of the j^th sub-problem. Linesearch is the formal method of choosing a step-size α_j^k along the direction d_j^k. Choice of α_j^k that ensures convergence is defined by what are called as the Armijo-Goldstein-Wolfe conditions [7], [11]. These conditions have been employed in [9] to obtain a Linesearch procedure reproduced below:

Linesearch Procedure Data: $\rho > 0, \gamma \in (0,1)$, $\delta \in (0,1)$.

1) choose a positive number
$$s_j^k \geq \rho | \bigtriangledown_{v_j} \phi_j^{kT} d_j^k | / \| d_j^k \|^2$$
2) compute the first non-negative integer i such that
$$\phi_j(V_j^k + (\delta)^i s_j^k d_j^k, \lambda_j^{k+1}, U^k; c_j^k) \leq \phi_j^k + \gamma(\delta)^i s_j^k \bigtriangledown_{v_j} \phi_j^{kT} d_j^k$$
and set $\alpha_j^k = (\delta)^i s_j^k$.

The method of steepest descent combined with the Linesearch procedure detailed above has been implemented. We use this implementation as a baseline system for experimental comparisons.

VI. Experimental Results and Conclusions

To test our implementations we ran it on two popular benchmarks "Breast Cancer" and "Satellite" from the UCI machine learning repository. The Breast Cancer benchmark consists of a total of 569 patterns with 32 attributes. The first attribute is a patient identification and the second is either of the labels M or B where M stands for malignant and B stands for benign. The rest of the 30 values are real numbers. The data was randomly split into a training set of 414 patterns and a test set consisting of the remaining patterns. Several runs were made. The output of the system for a few representative runs is reproduced below. Note that the performance of the system with dynamic tunneling is significantly superior to that with steepest descent as can be seen in Table I. Significant speed ups are observed as the number of nodes that participate in the computation increases. It is apparent that the random split into training and test data has led to a Test subset that is easy to generalize on. As a result, for the Breast Cancer data set, the results are consistently better for the test set. The emphasis of this paper is, however, to study the performance of the implementation and how much the use of different methods for solving the sub-problems influences the overall solution.

TABLE I
Results of block training algorithm with steepest descent and dynamic tunneling using 4 nodes on Breast Cancer data set.

	Steepest Descent - 4 nodes		
Run No.	% Accuracy		MSE
	Training	Testing	
1	57.52	78.14	0.00675
2	61.65	80.79	0.0048
3	59.9	29.8	0.0185
4	50.2	79.47	0.0049
	Block Training - 4 nodes		
Run No.	% Accuracy		MSE
	Training	Testing	
1	86.407	94.702	0.00081
2	91.504	96.026	0.0023
3	93.446	96.688	0.0009
4	92.407	94.702	0.005

The satellite data set is the other popular data set obtained from the machine learning repository at UCI. It consists of a training set with 4435 patterns and a test set with 2000 patterns. The attributes are multi-spectral values of pixels in 3×3 neighbourhoods in a satellite image. Since 4 spectral bands are considered the total number of attributes is $36 = 9 \times 4$. Table IV presents the results of some runs of the implementation made on this data set. To the best of our knowledge the best reported results give a % Accuracy of 86.02% on the test set.

To conclude, this paper offers an empirical study of a divide and conquer strategy implemented on a Linux cluster. Preliminary results show that combined with recent advances

TABLE II

RESULTS OF BLOCK TRAINING ALGORITHM WITH DYNAMIC TUNNELING USING 2 NODES AND 6 NODES ON BREAST CANCER DATA SET.

	Block Training - 2 nodes		
Run No.	% Accuracy		MSE
	Training	Testing	
1	89.077	92.053	0.00055
2	86.165	94.702	0.0018
3	88.349	93.377	0.00199
4	86.165	94.039	0.00032

	Block Training - 6 nodes		
Run No.	% Accuracy		MSE
	Training	Testing	
1	87.378	94.039	0.00517
2	87.621	94.039	0.0076
3	87.621	94.702	0.00087
4	87.1359	96.6887	0.00262

TABLE III

AVERAGE TIME TAKEN FOR CONVERGENCE ON BREAST CANCER DATA SET.

Ser. No.	Algorithm	Avg. Time over 5 runs
1	Dynamic Tunneling on a single node (no block training)	82 min
2	Block Training with Dynamic Tunneling using 2 nodes	7.2 min
3	Block Training with Dynamic Tunneling using 4 nodes	4.1 min
4	Block Training with Dynamic Tunneling using 6 nodes	2.76 min

in the theory of non-linear optimization techniques and Linux-based parallel computing technologies, these methods may offer cheap and powerful solutions for handling data exploration.

APPENDIX

ESTABLISHING A CONVERGENT TRAINING ALGORITHM

In [9] results have been presented that prove that both the stationary points and global minimizers of the original error function E can be located by searching for stationary points and global minimizers of the augmented Lagrangian described in equation 3, provided the penalty parameters given by the N dimensional vector $\mathbf{c}$ are sufficiently large. An iterative algorithm to solve Φ outlined in [9] can be described as a sequence of epochs as follows:

Step 1 Initialize the neural network to small random weights denoted by W^0. Set $U^0 = W^0$ and $V_j^0 = W^0,\ \forall\ j = 1, ..., N$. Choose $\lambda_j^0 \in \Re^n,\ j = 1, \ldots, N$ and $c^0 \in \Re^N$. Set $k = 0$. The current estimate for the problem variables is $(V_j^k, \lambda_j^k, U^k; c^k)$ where $j = 1, \ldots, N$.

Step 2 Holding U^k fixed, perform suitable updates on $\lambda_j^k, V_j^k,\ j = 1, ..., N$. These updates can be performed in parallel. To compute these updates the objective function for each block is considered. This is $\phi_j(V_j^k, \lambda_j^k, U^k; c_j^k)$ and is as given in equation 6. For a given $V_j = V_j^k$, ϕ_j is a strictly convex quadratic function of λ_j. Therefore one can analytically solve for obtaining the minima using

$$\bigtriangledown_{\lambda_j} \phi_j(V_j^k, \lambda_j^k, U^k; c_j^k) = 0.$$

This gives

$$\lambda_j^{k+1} = -\frac{V_j^k - U^k + 2\eta \bigtriangledown F_j(V_j^k)}{2 * (\tau \parallel V_j^k - U^k \parallel^2 + \eta)}). \tag{17}$$

Having obtained λ_j^{k+1} as above we now need to obtain V_j^{k+1} and for this we consider

$$\phi_j(V_j^k, \lambda_j^{k+1}, U^k; c_j^k).$$

If $\bigtriangledown_{V_j} \phi_j^k = 0$ then set $V_j^{k+1} = V_j^k$ else use a suitable non-linear optimization method to obtain a descent direction d_j^k and a stepsize α_j^k along this direction so that

$$\phi_j(V_j^k + \alpha_j^k d_j^k, \lambda_j^{k+1}, U^k; c_j^k) \leq \phi_j(V_j^k, \lambda_j^{k+1}, U^k; c_j^k). \tag{18}$$

Set $V_j^{k+1} = V_j^k + \alpha_j^k d_j^k$.

Step 3 After computing λ_j^{k+1} and V_j^{k+1} we now obtain U^{k+1} using the following expression:

$$U^{k+1} = \frac{1}{2\mu^k} \sum_{j=1}^{N} \left[\lambda_j^{k+1} + 2\left(c_j^k + \tau \parallel \lambda_j^{k+1} \parallel^2\right) V_j^{k+1}\right], \tag{19}$$

$$\text{where } \mu^k = \sum_{j=1}^{N} \left(c_j^k + \tau \parallel \lambda_j^{k+1} \parallel^2\right). \tag{20}$$

Step 4 We now update the penalty coefficients, $c_j^k,\ j = 1, ..., N$. Here $0 < \rho < \eta$ and $\theta > 1$. η is as in equation 4.

If, $\bigtriangledown_{V_j} \phi_j \left(V_j^k, \lambda_j^k, U^k; c_j^k\right)^T \left(V_j^k - U^k\right) +$
$\bigtriangledown_{\lambda_j}\ \phi_j \left(V_j^k, \lambda_j^k, U^k; c_j^k\right)^T \left(\bigtriangledown F_j \left(V_j^k\right) + \lambda_j^k\right)$
$\geq \rho \left(\parallel V_j^k - U^k \parallel^2 + \parallel \bigtriangledown F_j \left(V_j^k\right) + \lambda_j^k \parallel^2\right)$
then set $c_j^{k+1} = c_j^k$ else set $c_j^{k+1} = \theta c_j^k$.

Step 5 If $\Phi(V^{k+1}, \lambda^{k+1}, U^{k+1}; \mathbf{c}^{k+1})$
$> \Phi(V^k, \lambda^k, U^k; \mathbf{c}^k)$
set $V^{k+1} = V^0$, $\lambda^{k+1} = \lambda^0$ and $U^{k+1} = U^0$.

In the limit, all different copies of the neural networks represented by the weight vectors V_j^k will converge to the network described by the parameter vector U^k. The updates made to U^k are so designed that it averages the various solutions across blocks.

TABLE IV

RESULTS OF BLOCK TRAINING ALGORITHM WITH DYNAMIC TUNNELING ON SATELLITE DATA SET.

Number of nodes	% Accuracy		Average Time taken over 3 runs
	Training	Testing	
1	81.23	77.74	18 hrs 2 min
10	83.82	83.73	44 min

REFERENCES

[1] J. Barhen Vladimir Protopopescu, and David Reister. TRUST: A deterministic algorithm for global optimization. *Science*, 276:1094–1097, May 1997.

[2] D P Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. New York: Academic, 1982.

[3] D P Bertsekas and J Tsitsiklis. *Parallel and Distributed Computation*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[4] D P Bertsekas and J Tsitsiklis. *Neuro-Dynamic Programming*. Boston, MA: Athena, 1996.

[5] Ohio Supercomputer Centre. MPI Primer: Developing with LAM. Technical Report Version 1.0, The Ohio State University, November 1996.

[6] B. C. Cetin J. Barhen, and J. W. Burdlick. Terminal repeller unconstrained subenergy tunneling (TRUST) for fast global optimization. *J. Optimization Theory and Applications*, 77(1):97–126, 1993.

[7] R Fletcher. *Practical Methods of Optimisation*. Wiley, 1987.

[8] Message Passing Interface Forum. MPI:A Message-Passing Interface Standard. Technical Report Version 1.0, University of Tennessee, Knoxville, Tennessee, June 1995.

[9] L. Grippo˙ Convergent on-line algorithms for supervised learning in neural networks. *IEEE Transcations on Neural Networks*, 11(6):1284–1299, November 2000.

[10] S. Haykin. *Neural Networks - A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1999.

[11] D G Luenberger. *Linear and Non-linear Programming*. Addison Wesley, 1984.

[12] G. D. Pillo and L. Grippo. A new class of augmented Lagrangians in nonlinear programming. *SIAM J. Contr. Optim.*, 17(5):618–628, 1979.

[13] G. D. Pillo and S. Lucidi. On exact augmented Lagrangian functions for nonlinear programming problems. *Nonlinear Optimization and Applications*, pages 85–100, 1996.

[14] P. Roychowdhury Y. P. Singh, and R. A. Chansarkar. Dynamic tunneling technique for efficient training of multilayer perceptrons. *IEEE Transactions on Neural Networks*, 10(1):48–56, Jan 1999.

[15] B. Yegnanarayana. *Artificial Neural Networks*. Prentice Hall of India, 1999.