Military Technical College
Kobry El-Kobbah
Cairo, Egypt

10th International Conference
On Aerospace Sciences&
Aviation Technology

APPLYING NEWTON ALGORITHMS WITHIN A SUPERVISED FEED FORWARD NEURAL NETWORK ARCHITECTURE TO FORECAST A MISSILE TRAJECTORY

TAREK A. TUTUNJI *

ABSTRACT

A Neural Network is trained to forecast a moving trajectory. The neural network training is formulated as a nonlinear programming problem and a Newton method is used to find the optimal weights. The learning Algorithm is derived using a Recursive Prediction Error Method that approximates the inverse of the Hessian. Furthermore, box Constraints are added to the network weights to avoid network paralysis and a constraint nonlinear programming problem is formulated. Logarithmic Barrier methods which are a class of Interior Point Methods are presented. Interior point methods have good convergence properties because the weights move on a center path in the interior of the feasible weight. The logarithmic barrier method is combined with the Newton method to form a Newton-Barrier method.

The moving missile trajectory is simulated using differential equations and the proposed algorithm is used to train the network in order to forecast the missile position at any given time.

KEY WORDS

Neural Networks, Nonlinear Programming, Newton Algorithm, Logarithmic Barrier Method, Interior Point Methods, Recursive Prediction Error Method, Ballistic Missile Trajectory

* Assistant Professor, Department of Mechatronics Engineering, University of Philadelphia University, Jordan

1. INTRODUCTION

Artificial Neural networks are mathematical models that mimic the biological neuron connections. They consist of processing units called neurons, weights, and directed connections. Each unit receives input signals from weighting incoming connections and responds by sending its signal to other connected units. By comparing the input and the output patterns, a neural network updates its weights to give the desired output.

Neural networks have been used successfully in many areas such as pattern recognition, function approximation, system identification, control, and speech recognition.

A well-known learning algorithm for neural networks is back propagation, Werbos [12], that is based on the classical gradient descent method. Unfortunately, the slow convergence of the back propagation algorithm often results in training times exceeding hours of computer time

Newton-type based minimization techniques have been quite successful in speeding up the traditional back propagation scheme, Tutunji [10] and Watrous [11].

Barrier methods form a class of Interior point methods that are used for solving linear and nonlinear constrained optimization problems. Breakthroughs in interior point methods for solving large-scale optimization problems, Lustig et al [6], Gonzaga [4], Nash and Sofer [7], has gained much attention.

Supervised neural network training can be modeled as a nonlinear optimization problem. Specifically, minimizing a nonlinear error function with respect to the parameter weights. A wide variety of techniques from nonlinear optimization, Hertz et al [5], have been used for solving the learning optimization problem in Artificial Neural Networks. If weight constraints are added, the network can be also modeled as a nonlinear constraint optimization problem and logarithmic barrier functions can be combined with Newton methods to train the network.

In previous work, we introduced deterministic and stochastic Newton-Barrier methods and showed their performance on pattern recognition and function approximation problems, Tutunji [10], Trafalis and Tutunji [9]. The results proved to be superior over backpropagation.

In this paper, we represent two methods: Newton and Newton-Barrier with results for training neural networks to forecast trajectories. The trajectory considered is a simple ballistic missile. The Newton method was sufficient to forecast the trajectory in little time with insignificant error. Furthermore, the

Newton-Barrier Method was used to forecast a missile trajectory with thruster forces.

2. NEWTON-TYPE METHODS

Multi-layer neural networks are usually trained to perform a particular task by the backpropagation algorithm which is a version of the gradient algorithm. This type of method uses the negative gradient as the search direction. It is well-known however that this type of algorithm suffers from the drawback of slow convergence. The gradient descent algorithms make slow progress as they get closer to the minimum. The inefficiency of gradient descent is due to the fact that it moves in perpendicular steps, Bazaraa and Shetty [1]. Researchers have used Newton-type methods to update the network weights.

Given a function *f(w)*, general Newton method can be used to find the optimum variable, w, to minimize the function *f(w)*. The variable can be updated iteratively using the following formula

$$w^{k+1} = w^k - \frac{f'(w^k)}{f''(w^k)} \tag{1}$$

where w^k is the variable at iteration *k*. If more variables are involved in the function, then a vector of variables, their gradient, and Hessian is used

$$w^{k+1} = w^k + d^k \tag{2}$$

$$d^k = -H(w^k)^{-1} g(w^k) \tag{3}$$

where *w* now is a vector, *d* is the search direction, *g* is the gradient and *H* is the Hessian

The major step in those methods is calculating the inverse of the Hessian, H^{-1} . On large problems, calculating this inverse is considered to be too expensive. Another two major problems in calculating the inverse of the Hessian is ill-conditioning and positive definiteness.

In order to avoid the direct computation of the second derivatives (i.e Hessian), Quasi-Newton methods were developed, Fletcher [3]. In this class of methods, the inverse Hessian matrix is approximated iteratively using only information about the gradient. Also, for good approximations, the positive definiteness property is maintained.

The Quasi-Newton methods differ in the update strategy for the approximate inverse Hessian. The general approach is to consider various update families that meet the Quasi-Newton variable metric condition

$$P^{k+1} y^k = s^k \tag{4}$$

where P is the approximate inverse Hessian, $y^k = g^{k+1} - g^k$, and $s^k = w^{k+1} - w^k$. By g^k we denote the gradient of the error at the point w^k. Thus, the inverse Hessian maps a change in gradient to a change in position.

Recursive Prediction Error Method (RPEM) is a Newton-type algorithm. It approximates the Hessian iteratively using the following formula, Soderstrom [8]

$$H^{k+1} = \lambda \; H^k + g^{k+1} (g^{k+1})^T \tag{5}$$

where λ is the forgetting factor. By using the above approximation, we guarantee the Hessian to be positive definite and we avoid singularity problems. Then, the Sherman-Morrisson-Woodbury formula, Soderstrom [8] and Bazaraa et al. [1], is used to compute an approximation to the inverse of the Hessian which gives

$$P^{k+1} = \{ P^k - P^k g^{k+1} \left((g^{k+1})^T P^k g^{k+1} + \lambda \; I \right)^{-1} g^{k+1} P^k \} / \lambda \tag{6}$$

2.1 Newton Method for Neural Network Training

Consider a 3-layer neural network which receives an input signal X, processes it to the hidden layer which gives an output Y. Then, to the output layer that gives an output Z. Let

$X_p = (x_{p1}, x_{p2}, \ldots, x_{pq})$,
$Y_p = (y_{p1}, y_{p2}, \ldots, y_{pm})$,
$Z_p = (z_{p1}, z_{p2}, \ldots, z_{pn})$

denote the input vector, the hidden output vector, and the output vector respectively for a certain pattern p, where q, m and n are the number of nodes at these layers. Let V and W be the weight matrices between the input hidden and the hidden output layers respectively. The outputs of the hidden and output nodes can be computed as follows

$$y_j = f\left(\sum_{i=1}^{q} v_{ij} x_i\right) \qquad j=1,\ldots,m \tag{7}$$

$$z_h = f(\sum_{j=1}^{m} w_{jh} y_j) \qquad h=1,...,n \qquad (8)$$

where $f(x) = \frac{1}{1+e^{-x}}$

Although we did not use a bias weight, this term can be easily incorporated in the equations. Let $Dp = (d_{p1}, d_{p2}, \ldots, d_{pn})$ be the target vector representing the desired output of the network. The learning objective is to determine the W and V values that minimize the difference between the desired output Dp and the computed output Zp for all the patterns. Let the error criterion be defined as follows

$$E(v,w) = \frac{1}{2}\sum_{p=1}^{P}\sum_{h=1}^{n}(z_{ph} - d_{ph})^2 \qquad (9)$$

We will refer to this error as $E(w)$ since w is the variable that is updated using the gradient error information. The gradient of the error will have the following entrees

$$\frac{\partial E}{\partial w_{jh}} = (z_h - d_h) z_h (1 - z_h) y_j \qquad (10)$$

The hidden error is calculated using backpropagation and referred to as E_h. Specifically,

$$E_{hi} = y_i(1 - y_i)\sum_{j=1}^{n} E_j w_{ij} \qquad \text{where i is the i}^{\text{th}} \text{ hidden neuron} \qquad (11)$$

The hidden gradient

$$\frac{\partial E}{\partial v_{ij}} = (y_j - y_j^2) x_i \sum_{h=1}^{n} (z_h - d_h) z_h (1 - z_h) w_{jh} \qquad (12)$$

Now, consider the following constrained optimization learning problem for the case of an input-output feed-forward neural network.

min $E(w)$

The weights will be updated using the RPEM update equations (for convenience, we represent the matrix W as a column vector w).

$$w^{k+1} = w^k + \alpha^k d^k \qquad (13)$$

$$d_w^k = -P_w^k(\nabla E_w^k) \tag{14}$$

$$P_w^k = \{P_w^{k-1} - P_w^{k-1}\nabla E_w^k\left((\nabla E_w^k)^T P_w^{k-1}\nabla E_w^k + \lambda I\right)^{-1}(\nabla E_w^k)^T P_w^k\}/\lambda \tag{15}$$

the above equations are repeated pattern by pattern (i.e. batch mode). Here, α is the step size, ∇E_w^k is the gradient of the error for a single pattern and P_w^k is a positive definite matrix approximation to the inverse of the Hessian.

The weights at the hidden layer are updated using similar equations with ∇E_v instead of ∇E_w and v instead of w. The equations used for updating the hidden weights are the same as equations (13)-(15) and are repeated for completion

$$v^{k+1} = v^k + \alpha_v^k d_v^k \tag{16}$$

$$d_v^k = -P_v^k(\nabla E_v^k) \tag{17}$$

$$P_v^k = \{P_v^{k-1} - P_v^{k-1}\nabla E_v^k\left((\nabla E_v^k)^T P_v^{k-1}\nabla E_v^k + \lambda I\right)^{-1}(\nabla E_v^k)^T P_v^k\}/\lambda \tag{18}$$

2.2 Newton Algorithm Description

step 0	set eps = accuracy, p=1 (1st pattern), k=0 (1st iteration), W=W^0, V=V^0, H=cI, c is a large number and I is the identity matrix.
step 1	Evaluate the error using equation (9) If E < eps then stop. Else go to step 2
step 2	Given X_p, calculate Y_p and Z_p using equations (7) and (8)
step 3	Compute ∇E using equation (10)
step 4	Compute P^k using (15)
step 5	Update W using equation (13) for all of the output neurons
step 6	Compute ∇E_h using equation (12)
step 7	Compute $(P_h)^k$ using (18)
step 8	Update V for all the hidden neurons in a similar fashion using equation (16)
step 9	If p=P (last pattern) set p=1, k=k+1 and go to step 1 else p=p+1 and go to step 2.

3. BARRIER METHODS

Barrier functions are used to transform a constrained problem into an unconstrained problem or into a sequence of unconstrained problems. If the optimal solution occurs at the boundary of the feasible region, the procedure moves from the interior of the feasible region to the boundary. Consider the following constrained optimization problem

$$\begin{aligned} &\min E(w) \\ &\text{s.t. } c(w) \le 0 \end{aligned} \tag{20}$$

where w is a vector and c is a vector function whose components are c_1 , c_2 , ... , c_m . Now, we can form the barrier problem as

$$\begin{aligned} &\min \beta(w, \mu) = E(w) + \mu B(w) \\ &\text{s.t. } \mu \ge 0 \end{aligned} \tag{21}$$

where μ is the barrier parameter and B is the barrier function that is nonnegative and continuous over the region { w: $c(w) < 0$ }, and approaches infinity as the boundary of the region { w: $c(w) \le 0$ } is approached from the interior. More specifically, the barrier function B is defined by

$$B(w) = \sum_{i=1}^{m} \Phi[c_i(w)] \tag{22}$$

where Φ is a function of one variable that is continuous over { y: $y < 0$ } and satisfies

$$\Phi(y) \ge 0 \quad if \quad y < 0$$

$$\lim_{y \to 0^-} \Phi(y) = \infty$$

One barrier function is known Logarithmic Barrier Function, Bazaraa et al. [1],

$$B(w) = -\sum_{i=1}^{m} log[-c_i(w)] \tag{23}$$

Ideally, we would like the function B to take value of zero on the region { w: $c(w) < 0$ } and value infinity on its boundary. This would guarantee that we would not leave the region {w: $c(w) \le 0$}, provided that the minimization problem started at an interior point. However, this discontinuity poses serious difficulties for any computational procedure. Therefore, this ideal construction

of B is replaced by the more realistic requirement that B is nonnegative and continuous over the region { w: c(w) < 0 } and that it approaches infinity as the boundary is approached from the interior. Note that μB approaches the ideal barrier function described above as μ approaches zero.

3.1 General Barrier Algorithm

Initialization Step
Let $\varepsilon > 0$ be a termination scalar, and choose a point w^1 with $c(w^1) < 0$.
Let $0.1 > \mu^1 > 0$, $\mu step \in (1,1000)$, let $k=1$, go to main step
Main Step

1. Starting with w^k, solve the following problem:

$$\min \beta\ (w, \mu^k) = f(w) - \mu^k \sum_{i=1}^{m} log(-c_i(w))$$

Let w^{k+1} be an optimal solution, and go to step 2.

2. If μ^k, $\beta(w^{k+1}, \mu^k) < \varepsilon$, stop.
Else $\mu^{k+1} = \mu^k / \mu step$, $k=k+1$ and repeat step 1.

3.2 Newton-Barrier Method for Neural Network Training

Consider the following constrained optimization learning problem for the case of an input-output feed-forward neural network

$$\begin{aligned} &\min E(v,w) \\ &\text{s.t.} \quad -M < w_{jh} < M, \qquad -M < v_{ij} < M \end{aligned} \tag{24}$$

Heuristics are used for choosing those box constraints, M is a constant number. Then, the logarithmic barrier method can be used to transform it into an unconstrained problem as follows

$$\min \beta(v,w,\mu) = \frac{1}{2}\sum_{p}\sum_{h}\left(d_{ph} - z_{ph}\right)^2 - \mu\sum_{j}\sum_{h} log\left(M^2 - w_{jh}^2\right) - \mu\sum_{i}\sum_{j} log\left(M^2 - v_{ij}^2\right) \tag{25}$$

The derivatives of the above logarithmic error function with respect to the weights are (for convenience we used only one pattern)

$$\frac{\partial \beta}{\partial w_{jh}} = (z_h - d_h) z_h (1 - z_h) y_j + \mu \frac{2w_{jh}}{M^2 - w_{jh}^2} \tag{26}$$

$$\frac{\partial \beta}{\partial v_{ij}} = (y_j - y_j^2) x_i \sum_{h=1}^{n} (z_h - d_h) z_h (1 - z_h) w_{jh} + \mu \frac{2v_{ij}}{M^2 - v_{ij}^2} \tag{27}$$

The above equations can then be used for updating the neurons weights by using the new error β defined in equation (25) to replace the original error

defined in equation (9). This will result in a combination between the Newton algorithm described in section 2.2 and the general barrier algorithm described in 3.1. The resulted algorithm is referred to as the Newton-Barrier Algorithm. Previous work, Trafalis and Tutunji [9], has shown that the Newton-Barrier Algorithm has better convergence properties over the Newton Algorithm for complicated problems.

4. TRAJECTORY MODEL

A ballistic missile, like a bullet or an artillery shell, has no internal propulsion system. Once it is launched, it is has a projectile, following the trajectory dictated by Eqs. (28) and (29) below. Those parametric equations are derived from elementary differential equations.

$$x = (v_0 \cos\alpha)t \tag{28}$$

$$y = (v_o \sin\alpha)t - 0.5*g*t^2 \tag{29}$$

Where (x,y) are the two dimensional vector space coordinates. Both are functions of time. v_0 is the initial velocity, α is the firing angle, and g is the gravity force. For simplicity, the above equations assume that the only force acting on the projectile during the flight is the constant force of gravity.

A one-dimensional thrust force is added to equation (29) to establish a second model

$$y = (v_o \sin\alpha)t - 0.5*g*t^2 + T(t) \quad \text{where T is the Thruster force added} \tag{30}$$

Future work will test the described algorithms on cruise missile trajectories, which are rockets with their own guidance and propulsion systems. Other variables such as wind and pressure will also be added to the environment.

5. SIMULATION

The missile trajectory governed by equations (28) and (29) was simulated with an initial velocity of 600 Km/minute and a firing angle of 60 degrees. The trajectory over 100 minutes is shown in figure 1.

The proposed Newton algorithm described was used within a feed-forward neural network with the following architecture: 8 input neurons, 5 hidden neurons, and 2 output neurons. Each pattern used had 4 inputs for the x position and 4 inputs for the y position (i.e. 8 input neurons). Each output pattern had two outputs: the x and y position. As an example for each

pattern, the values at x_i and y_i where forecasted using x_{i-4}, x_{i-3}, x_{i-2}, x_{i-1}, y_{i-1}, y_{i-2}, y_{i-3}, and y_{i-4}. This resulted in a total of 96 patterns.

Several runs were tried with different initial weights. In all cases, the described Newton algorithm was successful in approximating the trajectory function and forecasting the missile position. Figure 2 shows the trajectory approximation while figure 3 shows error function convergence to a value of 0.00001 in only 8 iterations.

Another missile simulation similar to the previous model was run. This time however, thruster forces (in a weighted lookup table) were added, equation (30), to the ballistic trajectory in order to change its simple ballistic path as shown in figure 4. Here, the Newton-Barrier algorithm was used for forecasting as shown in figure 5.

6. CONCLUSIONS AND FUTURE RESEARCH

Two algorithms: Newton and Newton-Barrier were used within a feed-forward neural network to approximate a ballistic missile trajectory. The neural network problem was formulated as a nonlinear optimization problem and a detailed description of both algorithms was provided. In both algorithms a recursive prediction error method was used to approximate the inverse of the Hessian iteratively using only first derivative information. Simulation results show that the algorithms were successful in forecasting the missile position in only several iterations.

The described algorithms can be used in a fully integrated missile interceptor system as shown in figure 6. The radar will track the missile position at different times, the algorithm will then use this on-line information to forecast the missile trajectory at future times. The forecasted position of the missile will then be supplied to an interceptor launcher that will be fired in the direction of the forecasted missile position.

Future research will use the described algorithms to approximate more involved trajectories such as guided missiles within complex environment.

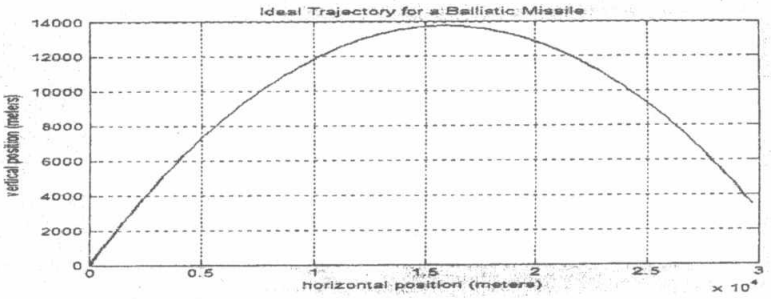

Figure 1. Ballistic Missile Trajectory Simulation

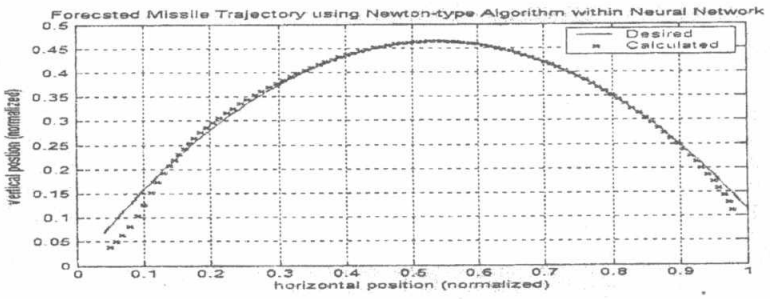

Figure 2. Forecasted Missile Trajectory using the Newton Algorithm

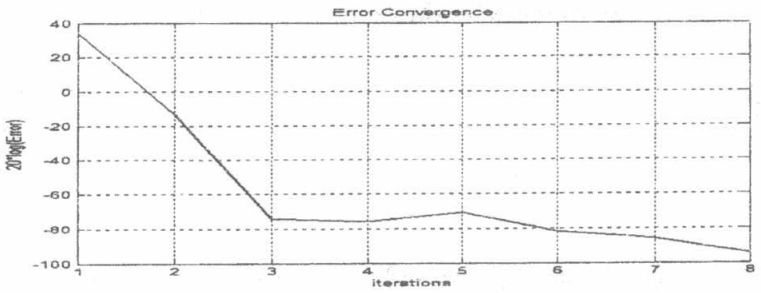

Figure 3. Error Convergence

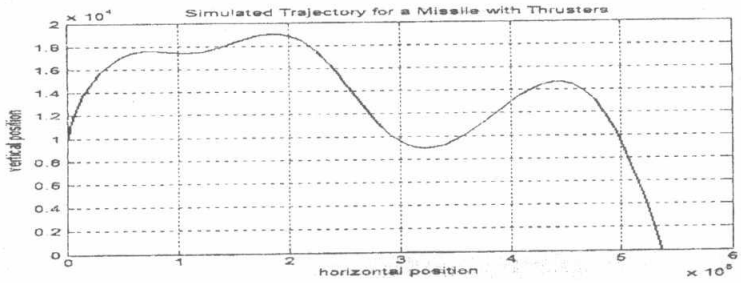

Figure 4. Missile Trajectory with Thruster force Simulation

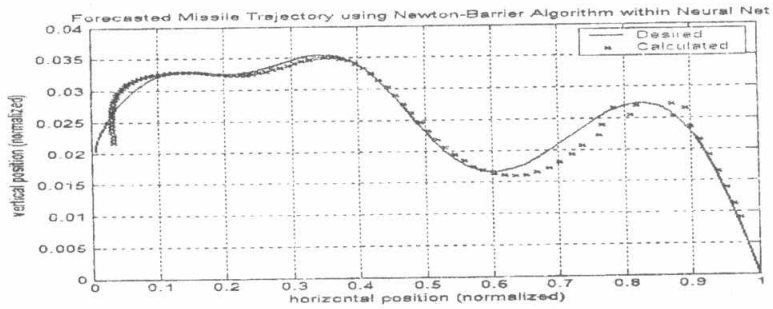

Figure 5. Forecasted Missile Trajectory using Newton-Barrier Algorithm

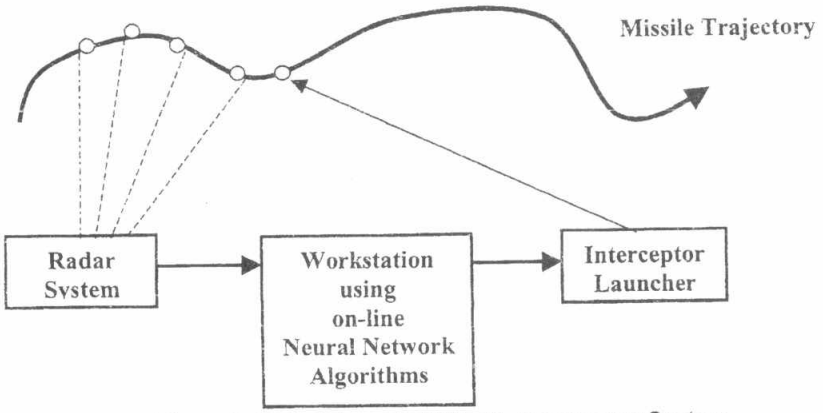

Figure 6. Fully Integrated Missile Interceptor System

REFERENCES

[1] Bazaraa, M.S., Sherali, H.D., and Shetty, C.M., (1993), *Nonlinear Programming Theory and Algorithms* , Wiley, NY.

[2] Davidon, W. C. (1976), "New Least-Square Algorithms", *Journal of Optimization Theory and Applications*, Vol. 18, no. 2, pp.187-197, Feb. 1976.

[3] Fletcher, R. (1990), *Practical Methods of optimization* , 2nd edition, Wiley.

[4] Gonzaga, C. C. (1991), "Large Step Path-following Methods for Linear Programming, Parts 1 & 2", *SIAM Journal of Optimization* 1 ,268-280.

[5] Hertz, J., Krogh, A., and Palmer, R. G. (1991), *Introduction to the Theory of Neural Computation* , CA: Addison-Wesley, Redwood City .

[6] Lustig, I. J., Marsten, R. E., and Shanno, D. F (1994), "Interior Point Methods for Linear Programming: Computational State of the Art", *ORSA Journal on Computing*, 6 (1),1-14.

[7] Nash, S. G. and Sofer, A. (1993),"A Barrier Method for Large-scale Constrained Optimization", *ORSA Journal on Computing* , vol. 5, No. 1, pp. 40-53.

[8] Soderstrom, T. and Stoica, P. (1989), *System Identification* , Prentice Hall International (UK), Englewood Cliffs, NJ

[9] Trafalis, T. B. and Tutunji, T. A. (1995), "Deterministic and Stochastic Logarithmic Barrier Function Methods for Neural Network Training", Parallel Computing in Optimization, (A. Migdalas, P.M. Pardalos and S. Storoy, Eds) Kluwer Academic Publishers, Chapter 13, pp. 519-574, 1997

[10] Tutunji, Tarek A. (1996), *Logarithmic Barrier Functions and Newton-type Methods with Applications to Neural Network Training*, Dissertation , School of Industrial Engineering , University of Oklahoma.

[11] Watrous, R. L. (1987), "Learning Algorithm for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization", *IEEE First International Conference on Neural Networks*, San Diego, CA, 619-628.

[12] Werbos, P. (1974), "Beyond Regression: New tools for Prediction and Analysis in the Behavioral Sciences", *Ph.D. Thesis*, Committee on Applied Mathematics, Harvard University, Cambridge, MA, reprinted by Wiley, "The Roots of Back Propagation", 1994.

Author background:

Dr. Tarek A. Tutunji is a full-time assistant professor at Philadelphia University in Jordan. He received his PhD (Industrial Engineering) in 1996 and his MS (Electrical Engineering) in 1993. Both degrees were received from the University of Oklahoma, USA. He has six years work experience prior to joining Philadelphia University. He worked as a manufacturing engineer (2 years) and as an optimization algorithm developer (2 years) for Halliburton Energy Services in Texas. He later worked as a design engineer (2 years) for Seagate Tech in Oklahoma City.