Neural Networks for Solving Constrained Optimization Problems

VALERI M. MLADENOV*, NICHOLAS G. MARATOS**

(*) Department of Theoretical Electrotechnics, Faculty of Automatics,
Technical University of Sofia, 1756 Sofia, BULGARIA.

(**) Dept. of Electrical and Computer Engineering, Division of Electroscience,
National Technical University of Athens, 9 Iroon Polytechniou St., 15773 Athens, GREECE.

Abstract: - In this paper we consider several Neural Network architectures for solving constrained optimization problems with inequality constrains. We present a new architecture based on the exact penalty function approach. Simulation results based on SIMULINK® models are given and compared.

Key-Words: - neural networks, constrained optimization, penalty functions, exact penalty functions

1 Introduction

Artificial Neural Networks (ANN) have already been used to obtain solution of constrained optimization problems [1]. In 1984 Chua and Lin [2] developed the canonical non-linear programming circuit, using the Kuhn-Tucker conditions from mathematical programming theory. Later, Tank and Hopfield [3] developed their optimization network for solving linear programming problems. Some practical design problems of the Tank and Hopfield network along with its stability properties are discussed in [4].

An extension of the results of Tank and Hopfield to more general non-linear programming problems is presented in [5]. The authors note that the network introduced by Tank and Hopfield can be considered to be a special case of the canonical non-linear programming network proposed in [2], with capacitors added to account for the dynamic behavior of the circuit. The above discussed approach implicitly utilizes the penalty function method [1], [6], [10] whereby a constrained optimization problem is approximated by an unconstrained optimization problem. In [7] an exact penalty function approach is used and a neural optimization network is presented for solving constrained optimization problems. The proposed architecture can be viewed as a continuous Neural Network (NN) model, and, in [8], use is made of SIMULINK® for modeling and simulations of its behavior. In this paper we synthesize a new NN architecture based on a different exact penalty function. The proposed optimization network is capable of solving a general class of constrained optimization problems. Simulation results based on MATLAB® and SIMULINK® are given.

The paper is outlined as follows. In the next section we give a brief overview of some important NN models based on penalty functions (both differentiable and exact) for solving constrained optimization problems with inequality constraints. Based on the max exact penalty function we present a new NN architecture. Section 3 contains simulation results and section 4 some concluding

2 Neural Networks for Solving Constrained Optimization Problems Based on the Penalty Function Approach

We begin this section with a brief overview of NN architectures proposed in the literature for solving constrained optimization problems with inequality constraints. The general form of the problem to be solved is

$$\min_{x} f(x)$$
$$\text{subject to:} \quad g_i(x) \geq 0,\ i=1,2,\ldots,m \tag{1}$$

where $x=(x_1, x_2,\ldots, x_n)^T$. In (1) $f(x)$ is the objective function which has to be minimized and $g_i(x)$, $i=1,2,\ldots,m$ are inequality constraints. It should be noted that any equality constraint $h_i(x)$, could be transformed into two inequality constraints $h_i(x) \geq 0, -h_i(x) \geq 0$, i.e. the problem (1) is

sufficiently general. All functions involved are assumed to be continuously differentiable.

The penalty function method consists of transforming the constrained optimization problem (1) into an unconstrained one based on a penalty function defined as follows [10]:

$$E(x,k) = f(x) + \sum_{i=1}^{m} k_i P(g_i(x)) \tag{2}$$

where the penalty term $P(g_i(x))$ should satisfy

$$\begin{aligned} P(g_i(x)) &= 0, \quad \text{if} \quad g_i(x) \geq 0, \\ P(g_i(x)) &> 0, \quad \text{if} \quad g_i(x) < 0. \end{aligned} \tag{3}$$

In the NN literature the penalty function $E(x,k)$ is often viewed as an energy function for the corresponding network [1]. Assuming differentiability, a local minimum of the penalty function $E(x,k)$ is found by using the following dynamic gradient scheme [1]:

$$\begin{aligned} &\frac{dx}{dt} = -m\nabla_x E(x,k), \qquad x(0) = x^{(0)} \\ &m = diag(m_1, m_2, ..., m_n) \end{aligned} \tag{4}$$

or, in extended form,

$$\left.\begin{aligned} \frac{dx_1}{dt} &= -m_1\left(\frac{\partial f(x)}{\partial x_1} + \sum_{i=1}^{m} k_i \frac{\partial P}{\partial g_i}\frac{\partial g_i(x)}{\partial x_1}\right), \qquad x_1(0) = x_1^{(0)} \\ \frac{dx_2}{dt} &= -m_2\left(\frac{\partial f(x)}{\partial x_2} + \sum_{i=1}^{m} k_i \frac{\partial P}{\partial g_i}\frac{\partial g_i(x)}{\partial x_2}\right), \qquad x_2(0) = x_2^{(0)} \\ &\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ \frac{dx_n}{dt} &= -m_n\left(\frac{\partial f(x)}{\partial x_n} + \sum_{i=1}^{m} k_i \frac{\partial P}{\partial g_i}\frac{\partial g_i(x)}{\partial x_n}\right), \qquad x_n(0) = x_n^{(0)} \end{aligned}\right\} \tag{5}$$

where $m_j > 0$, $k_i > 0$. Usually one takes $m_j = m = 1/t$, $j=1,2,\ldots,m$, where t is the NN time constant, and $k_i = k$, $i=1,2,\ldots,n$.

Normally, two types of penalty terms are used

$$P(g_i(x)) = [\min\{0, g_i(x)\}]^2 \tag{6}$$

$$P(g_i(x)) = -\min\{0, g_i(x)\} \tag{7}$$

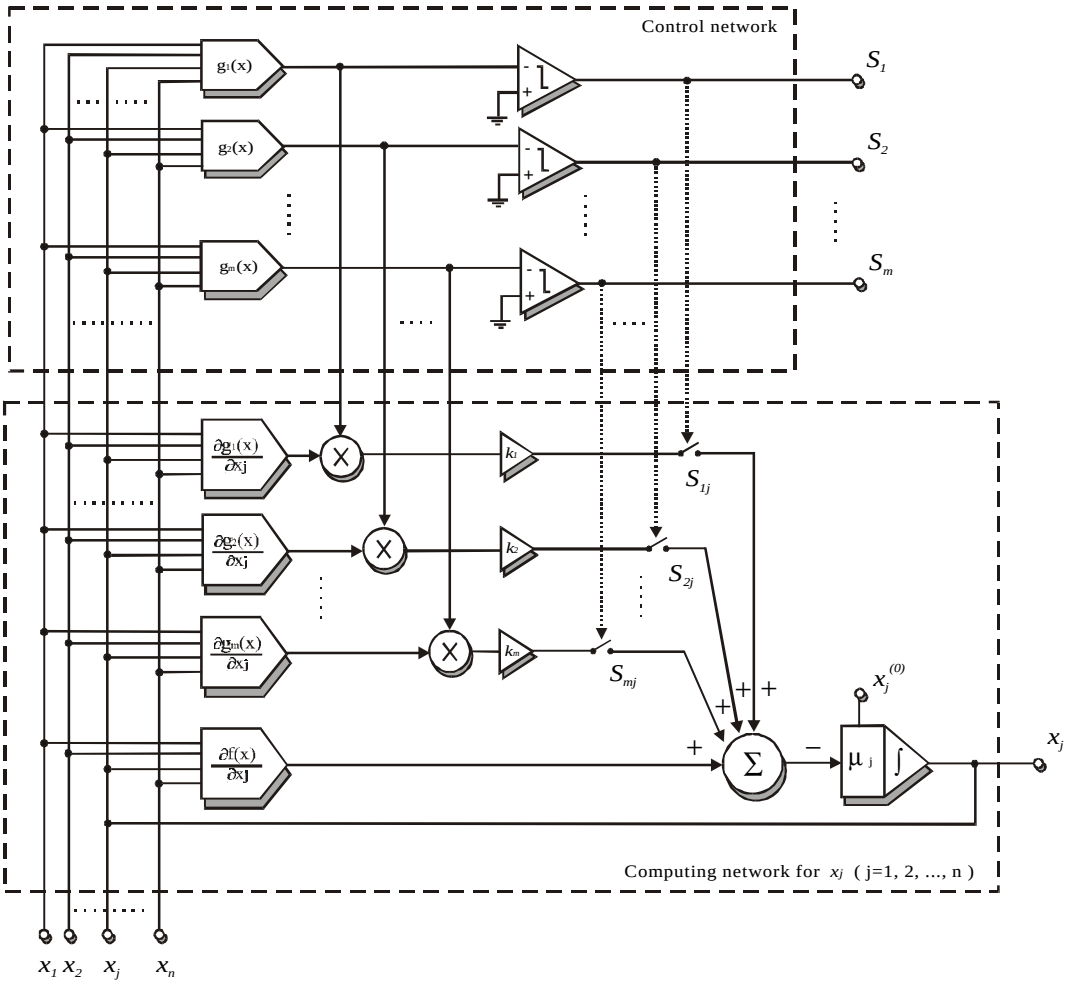

Figure 1. Neural Network for solving the constrained optimization problem based on the penalty terms (6)

Based on the penalty terms (6), the energy function obtained is

$$E(x,k) = f(x) + \frac{1}{2}\sum_{i=1}^{m} k_i [\ min\{0, g_i(x)\}]^2\ , \quad (8)$$

Assuming that $g_i(x)$, $i=1,2,\ldots,m$ have continuous first derivatives, it is easy to show that the same is true for $P(g_i(x))$ as defined by (6), therefore $E(x,k)$ in (8) is continuously differentiable. The gradient of $P(g_i(x))$ in (6) is:

$$\nabla[min\{0, g_i(x)\}]^2 = min\{0, g_i(x)\}\nabla g_i(x) \quad (9)$$

By using the gradient strategy for minimizing $E(x,k)$, the following system of ordinary differential equations [1], [8] is obtained

$$\frac{dx_j}{dt} = -m_j\left(\frac{\partial f(x)}{\partial x_j} + \sum_{i=1}^{m} k_i S_i g_i(x)\frac{\partial g_i(x)}{\partial x_j}\right), \quad (10)$$
$$j = 1, 2, \ldots, n$$

where,

$$S_i = \begin{cases} 1, & if \quad g_i(x) \le 0, \\ 0, & if \quad g_i(x) > 0. \end{cases} \quad (11)$$

and, obviously,

$$S_i g_i(x) = min\{0, g_i(x)\}.$$

The functional scheme for simulating the above equations is given in figure 1. This scheme could be considered as a Neural Network, where the integrators represent the neurons (basic units) and functional nonlinear generators build up the connections between them.

This approach, described in [1], [5], [6], essentially replaces the constrained problem (1) by an unconstrained minimization of the differentiable penalty function (8). However, theoretical results on the penalty function method [10], show that equivalence of the problems (1) and $\min_x\{E(x,k)\}$ is only obtained in the limit, as $\min_{i=1,\ldots,m}\{k_i\} \to \infty$.

Since the values of the parameters k_i, $i=1,\ldots,m$ used by the NN (10) of fig.1 are finite, it follows that the unconstrained minima of (8) obtained by the NN will only be approximations to the true solutions of the constrained problem (1). The quality of the approximation improves if larger values are chosen for the k_i's, however, the true solution of (1) will not be obtained unless $\min_{i=1,\ldots,m}\{k_i\} \to \infty$.

Let us now consider the penalty terms (7). Using these in (2) gives the following penalty function:

$$E(x,k) = f(x) - \sum_{i=1}^{m} k_i\ min\{0, g_i(x)\} \quad (12)$$

Due to the non-differentiability of the penalty term (7), this penalty function is non-differentiable, even though the functions $g_i(x)$ are assumed differentiable. The penalty function (12) is termed exact because it can be proven, [10], that any unconstrained minimum of (12) is also a solution of the constrained problem (1), provided that $\min_{i=1,\ldots,m}\{k_i\}$ is sufficiently large. Thus, in contrast to the previous situation, replacing (1) by an unconstrained minimization of (12) will yield the true solution of (1) if the k_i's are sufficiently large but finite.

When the penalty terms (7) are used, the corresponding system of ordinary differential equations is [1]:

$$\frac{dx_j}{dt} = -m_j\left(\frac{\partial f(x)}{\partial x_j} - \sum_{i=1}^{m} k_i S_i \frac{\partial g_i(x)}{\partial x_j}\right), \quad (13)$$
$$j = 1, 2, \ldots, n$$

where $m_j > 0$ and

$$S_i = \begin{cases} 1, & if \quad g_i(x) \le 0, \\ 0, & if \quad g_i(x) > 0. \end{cases} \quad (14)$$

In this case the functional scheme is given by figure 2. This is a simpler NN than those of figure 1. It should be noted that this Neural Network realization uses simple switches instead of the expensive analog multipliers. However because of the non-smooth nature of the penalty function, the first derivative discontinuities could influence into parasitic effects and slow up the solution speed.

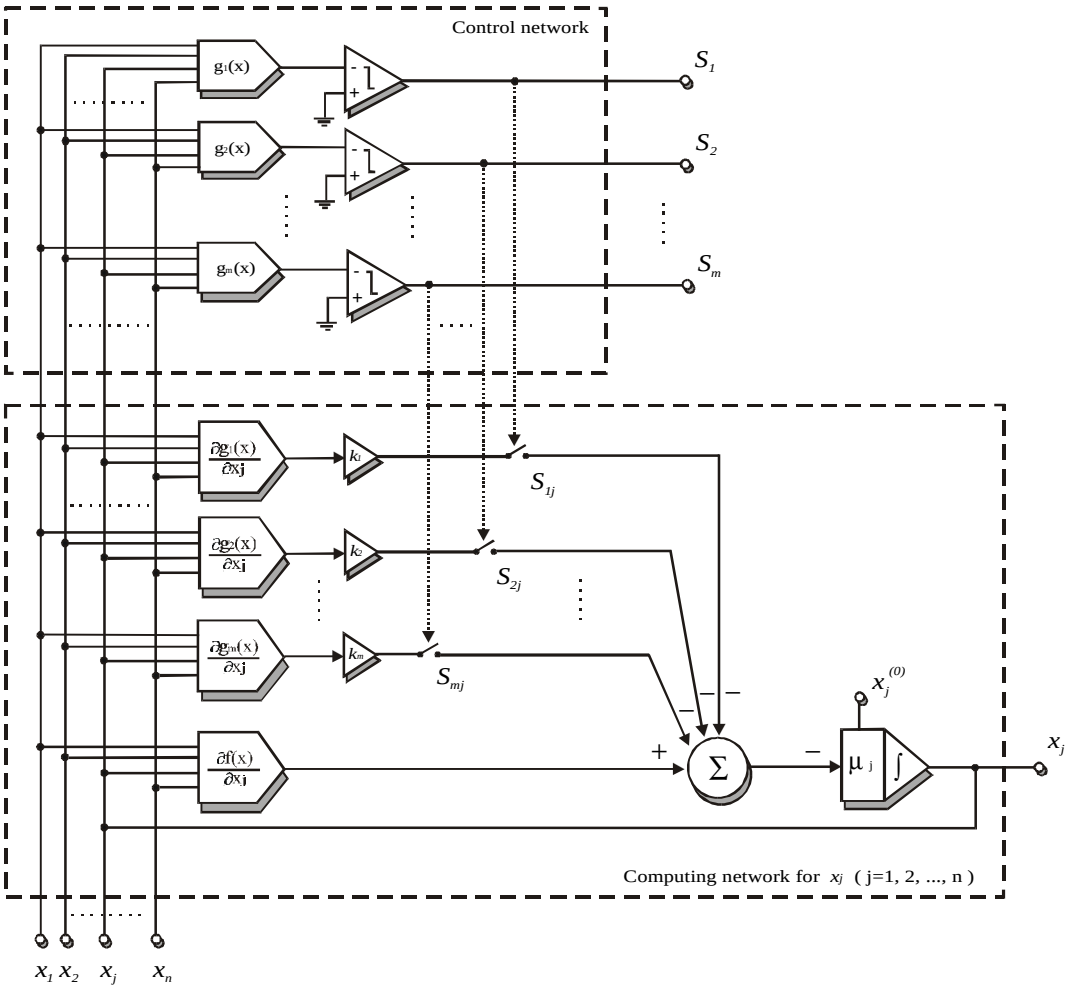

Figure 2. Neural Network for solving the constrained optimization problem based on the penalty terms (7).

In this paper we propose to use an alternative penalty function. Instead of adding together the penalty terms $-k_i \, min\{0, g_i(x)\}$, we propose to take the maximum of these terms. Thus the following penalty (energy) function is obtained:

$$\begin{aligned} E(x,k) &= f(x) + \max_{i=1,\ldots,m} \{k_i P(g_i(x))\} = \\ & f(x) + \max_{i=1,\ldots,m} \{\max\{0, -k_i g_i(x)\}\} = \\ & f(x) + \max\{0, -k_1 g_1(x), -k_2 g_2(x), \ldots, -k_m g_m(x)\} \end{aligned} \tag{15}$$

In this penalty function, only the most violated inequality constraint is taken into account. The penalty function (15) has been analyzed in the optimization literature, and it has been shown, [11], that it is an exact penalty function, i.e. that replacing (1) by an unconstrained minimization of (15) will yield the true solution of (1) if the k_i's are sufficiently large but finite. Obviously, (15) is non-differentiable.

In order to apply the gradient strategy for minimizing (15), we select

$$\frac{\partial x}{\partial t} \in -m \partial E(x,k) \tag{16}$$

where $\partial E(x,k)$ is the generalized gradient of (15). This may be done by using the same system of ordinary differential equations (13) as before, where now the control signals S_i are defined as follows:

$$S_i = \begin{cases} 1, & \text{if} \quad \max\{0, -g_i(x)\} = \max\limits_{j=1,\ldots,m} \{\max\{0, -g_i(x)\}\}, \\ 0, & \text{if} \quad \max\{0, -g_i(x)\} < \max\limits_{j=1,\ldots,m} \{\max\{0, -g_i(x)\}\}. \end{cases} \tag{17}$$

The corresponding functional scheme, which in fact is the proposed NN architecture, is shown in figure 3.

The difference between the architecture of figure 2 and the proposed NN architecture (fig. 3) consists in a "winner-take-all" block included into the Control Network. This block is used to determine

dynamically the index i^* of the most violated inequality and to generate the control signal $S_{i^*}=1$ corresponding to this inequality, while all other control signals (corresponding to other inequalities) are equal to 0. Details of the operation of the "winner-take-all" block are given in [1]. Because of the non-smooth character of the exact penalty function (15), the first derivative discontinuities can lead sometimes again into parasitic effects and slow up the solution speed

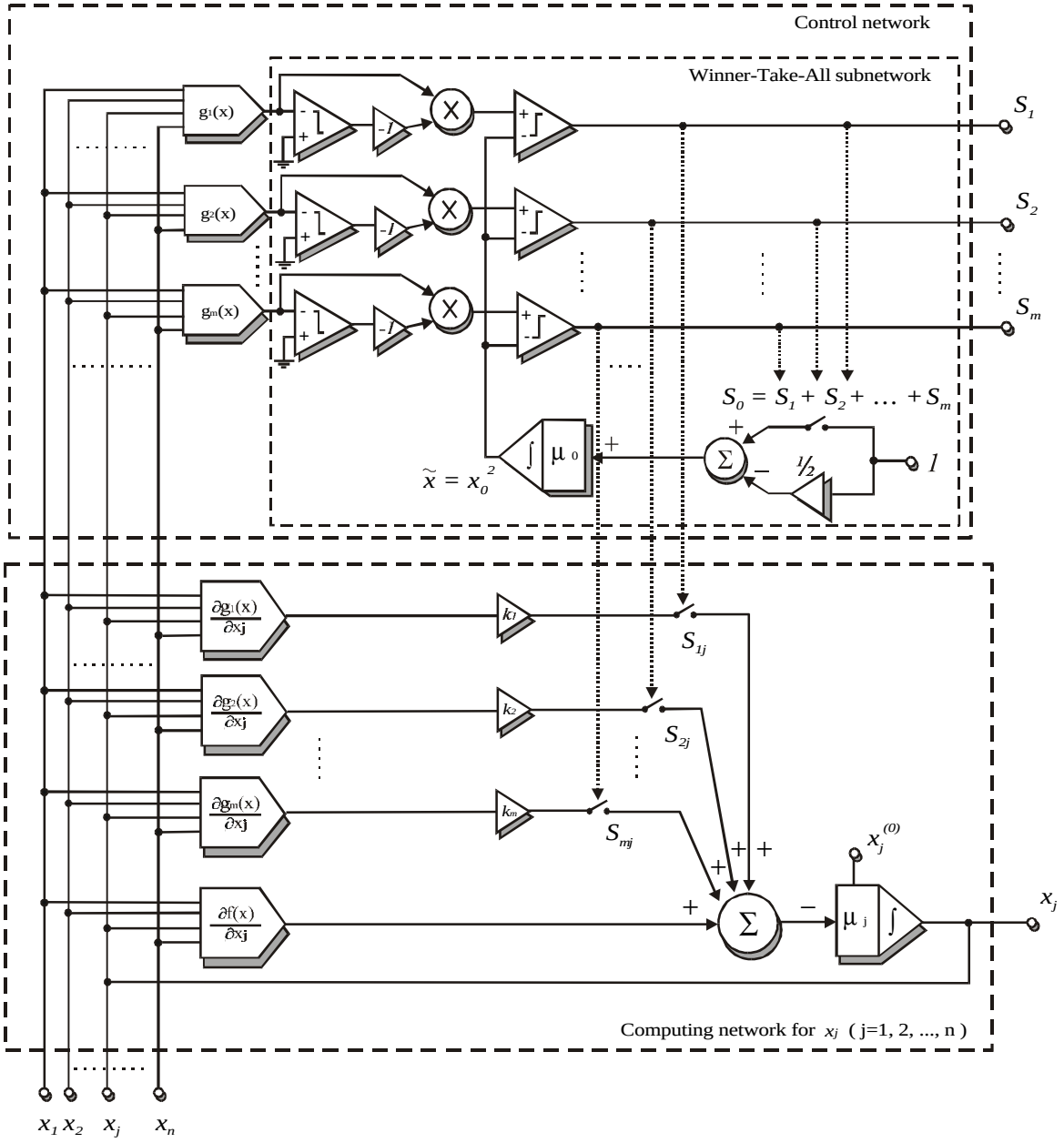

Figure 3. Proposed Neural Network for solving the constrained optimization problem based on the max exact penalty function

3 Simulation Example for testing the Neural Networks

In this section we present simulation results for the Neural Network architectures considered. We use the following simple example

$$\underset{x_1,x_2}{minimize} \quad f(x) = 2x_1^2 + x_2^2$$

subject to: (18)

$$\begin{cases} g_1(x) = -x_1 + x_2 - 2 \geq 0 \\ g_2(x) = x_1 + x_2 - 2 \geq 0 \end{cases}$$

The solution of the above problem is the point x* = (x_1*, x_2*) = (0, 2) as can be easily verified analytically. For this example we have *m=2, n=2.* We have made SIMULINK® models of the Neural Networks of figures 1, 2 and 3, and have simulated their behavior. The results are given for the following parameter values: $k_1 = 20$, $k_2 = 20$, $m_1 = 10$, $m_2 = 10$.

The SIMULINK® model of the NN of figure 1 is given in figure 4.

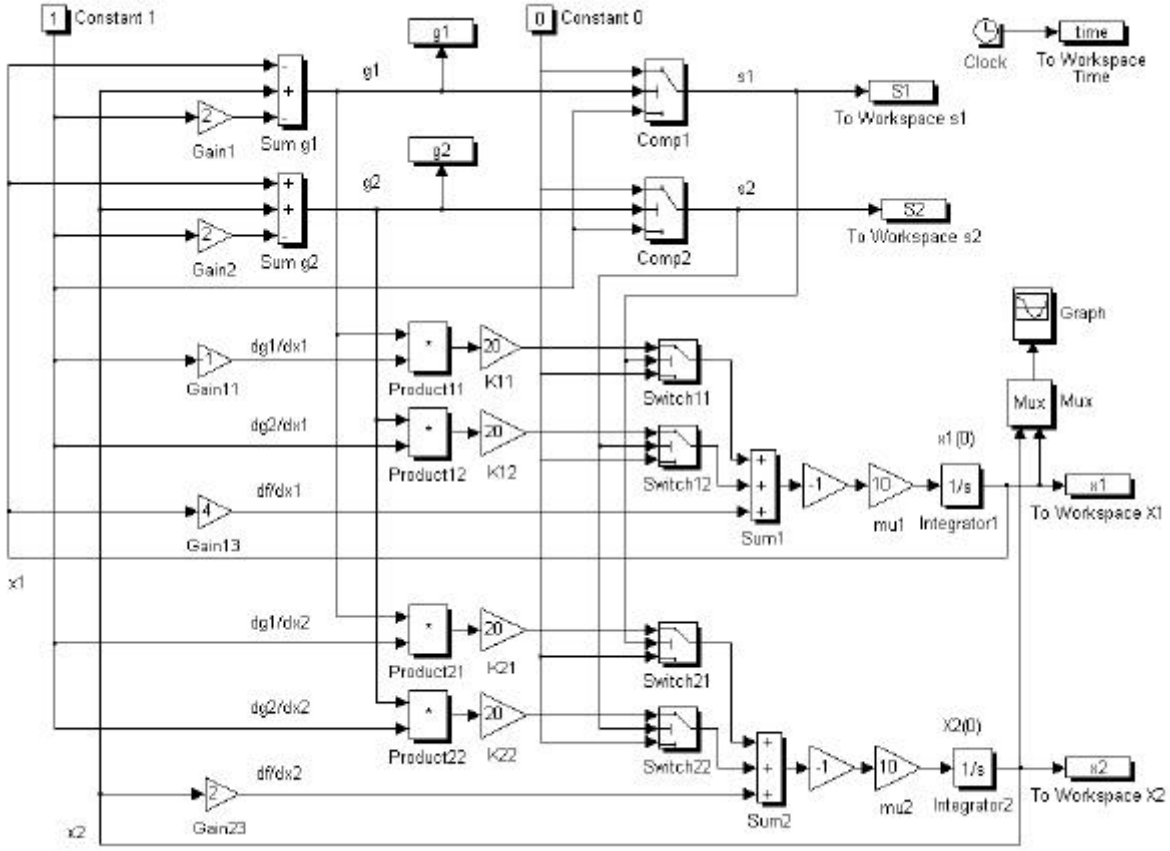

Figure 4. SIMULINK® model for the NN of figure 1, for the solution of problem (18)

The simulation results for initial conditions $x^{(0)} =(x_1^{(0)},x_2^{(0)})=(1,-1)$ and $x^{(0)} =(x_1^{(0)},x_2^{(0)})=(-1,1)$ are given in figure 5a and figure 5b respectively.

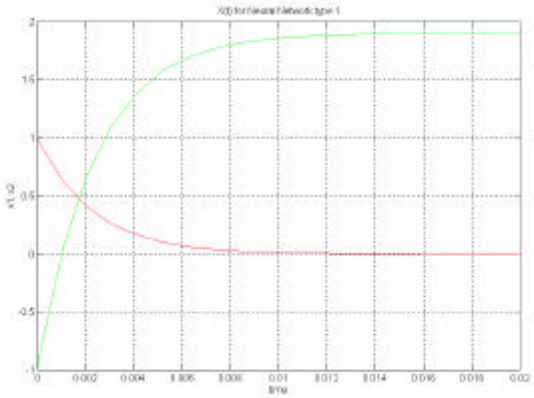

Figure 5a. Simulation results for initial conditions $x^{(0)} =(x_1^{(0)},x_2^{(0)})=(1,-1)$ for the NN model of figure 4

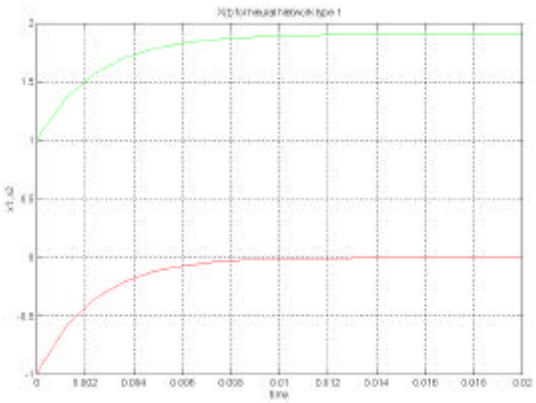

Figure 5b. Simulation results for initial conditions $x^{(0)} =(x_1^{(0)},x_2^{(0)})=(-1,1)$ for the NN model of figure 4

The model of the NN of figure 2 is given in figure 6.

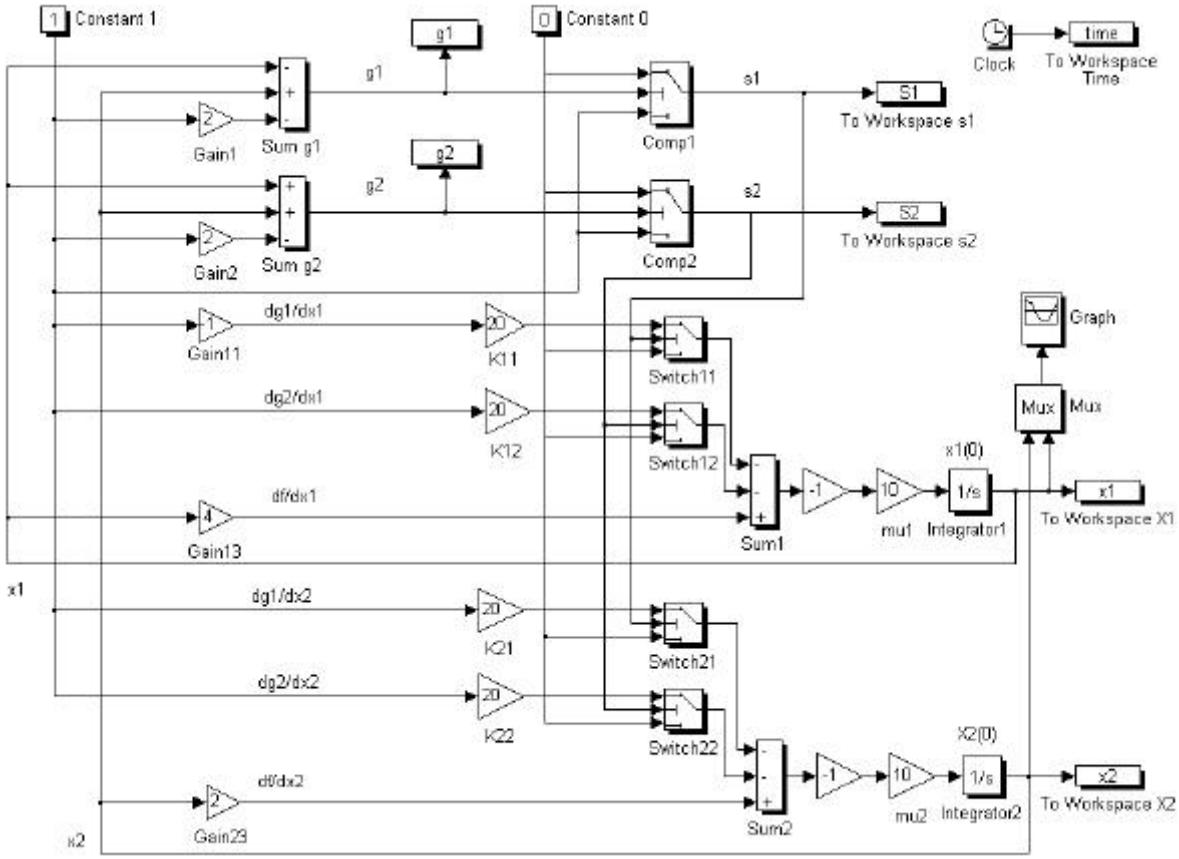

Figure 6. SIMULINK® model for the NN of figure 2, for the solution of problem (18)

The corresponding simulation results for initial conditions $x^{(0)} =(x_1^{(0)},x_2^{(0)})=(1,-1)$ and $x^{(0)} =(x_1^{(0)},x_2^{(0)})=(-1,1)$ are given in figure 7a and figure 7b respectively.

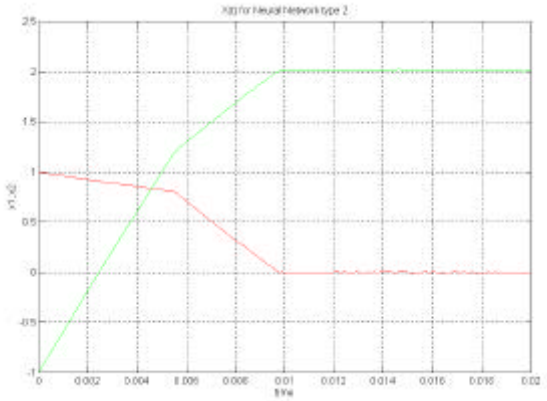

Figure 7a. Simulation results for initial conditions $x^{(0)} =(x_1^{(0)},x_2^{(0)})=(1,-1)$ for the NN model of figure 6

Finally, we have modeled the proposed Neural Network of figure 3. The corresponding SIMULINK® model is given in figure 8.

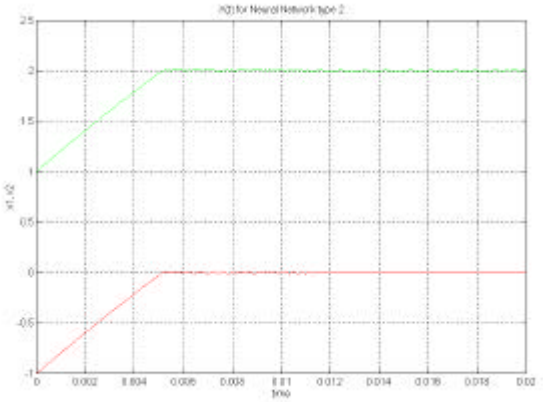

Figure 7b. Simulation results for initial conditions $x^{(0)} =(x_1^{(0)},x_2^{(0)})=(-1,1)$ for the NN model of figure 6

The simulation results of the behavior of this model - for initial conditions $x^{(0)} =(x_1^{(0)},x_2^{(0)})=(1,-1)$ and $x^{(0)} =(x_1^{(0)},x_2^{(0)})=(-1,1)$ - are given in figure 9a and figure 9b respectively.

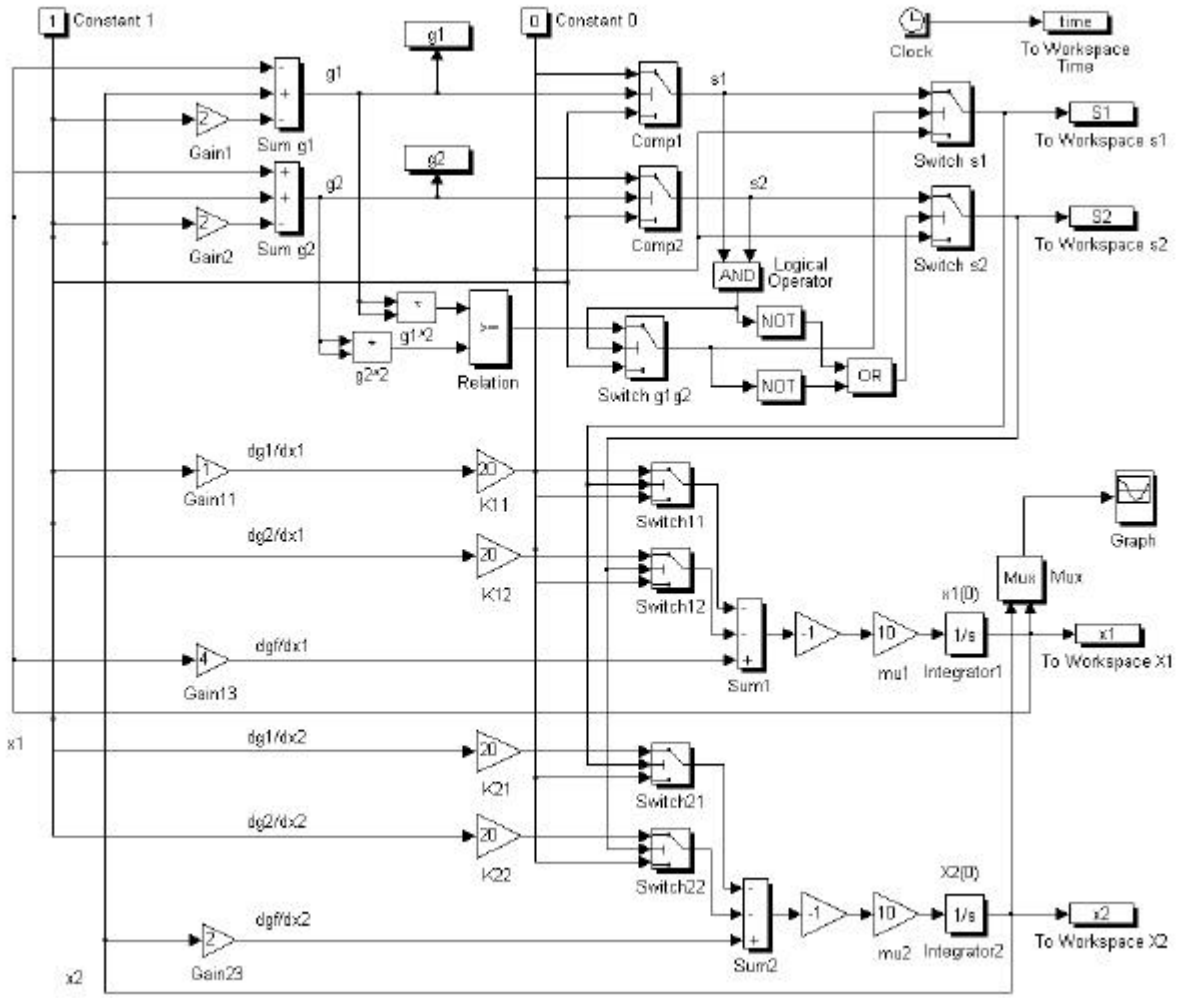

Figure 8. SIMULINK® model for the proposed NN of figure 3, for the solution of problem (18)

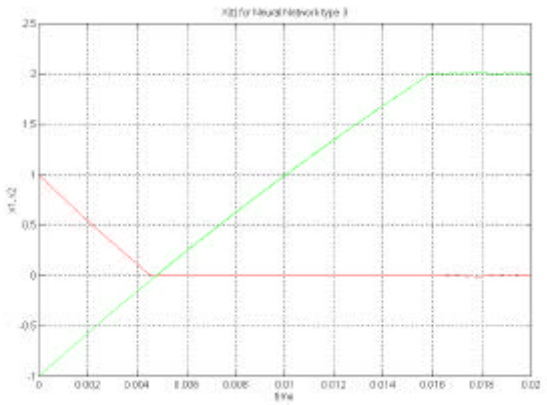

Figure 9a. Simulation results for initial conditions $x^{(0)} = (x_1^{(0)}, x_2^{(0)}) = (1,-1)$ for the NN model of figure 8

In relation with the results of figures 5, 7 and 9, the following observations should be made. The asymptotic value of the variable x_2 in figures 5a and 5b is close to the optimal value $x_2^* = 2$ of (18), however it is not equal to x_2^*. This is due to the non-exact character of the penalty function (8) minimized by the NN of figure 1 (model of fig. 4).

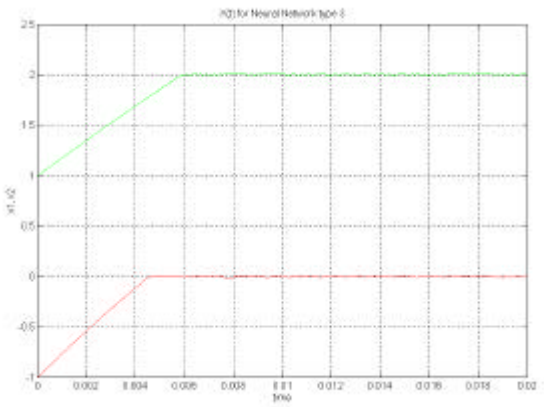

Figure 9b. Simulation results for initial conditions $x^{(0)} = (x_1^{(0)}, x_2^{(0)}) = (-1-1)$ for the NN model of figure 8

By contrast, the Neural Networks of figures 2 and 3 (models of fig. 6 and 8), which are based on the exact penalty functions (12) and (15) respectively,

do obtain the true optimal point $x^* = (x_1^*, x_2^*) = (0, 2)$ eventually. This is clearly depicted in figures 7a, 7b and 9a, 9b, respectively. This situation has been confirmed by extensive experimentation (with different parameter values) for the simple problem considered here. It should also be observed that for the NN of figure 1, the final steady state is reached exponentially, theoretically for $t \to \infty$. For the NN of figure 2, the steady state is reached faster (in finite time) and, as mentioned above, coincides with the analytical solution of problem (18). The proposed NN (figure 3) behaves in a similar manner as the NN from figure 2. The only practical problems encountered in these cases were related to modeling of the discontinuities (for $g(x)=0$) with SIMULINK®. The parameters k_1 and k_2 were chosen after some experimentation with the models. For smaller values of these parameters, some of the constraints could be violated, while the larger values of k_1 and k_2 could influence into different effects in the dynamic behavior of Neural Networks due to ill conditioning etc. The parameters m_1 and m_2 are directly connected with the duration of the transient. For larger m_1 and m_2 the transient is faster but this is connected with difficulties during the simulation procedure.

4 Conclusion

In this paper we consider several Neural Network architectures for solving constrained optimization problems with inequality constrains. We present a new architecture based on the max penalty function. SIMULINK® models of the considered NN are presented and simulation results are given.

Acknowledgement:
This work was supported by NATO Collaborative Linkage Grant PST.CLG 975709.

References:

[1] A. Cichocki, R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, John Wiley & Sons, Chichester-New York-Brisbane-Toronto-Singapore, 1993.

[2] L. O. Chua and G. N. Lin. “Non-linear programming without computation”, *IEEE Trans. Circuits and Systems*, CAS-31, pp.182-186, 1984.

[3] D. W. Tank and J. J. Hopfield. “Simple “neural” optimization networks: an A/D converter, signal decision circuit, and a linear programming circuit”, *IEEE Trans. Circuits and Systems*, CAS-33, pp.533-541, 1986.

[4] M. J. Smith *and* C. L. Portmann. “Practical design and analysis of a simple “neural” optimization circuit”, *IEEE Trans. Circuit and Systems*, Vol.36, pp.42-50, 1989.

[5] M. P. Kennedy and L. O. Chua. “Neural networks for non-linear programming”, *IEEE Trans. Circuit and* Systems, Vol. 35, pp.554-562, 1988.

[6] W. E. Lillo, M. H. Loh, S. Hui and S. H. Žak. “On solving constrained optimization problems with neural networks : a penalty method approach”, *Technical Report TR EE 91-43*, School of EE, Purdue University, West Lafayette, IN, 1991

[7] W. E. Lillo, S. Hui, S. Hui and S. H. Žak. “Neural network for constrained optimization problems”, *International Journal of Circuit Theory and Applications*, vol. 21, pp.385-399, 1993.

[8] V.M. Mladenov, P.N. Proshkov, "Modelling and Simulation of Continuous Neural Networks for Constrained Optimization Problems", *2nd IMACS International Conference on: Circuits, Systems and* Computers, *(CSC'98)*, Greece, published in "Recent Advances in Information Science and Technology", World Scientific, ISBN 981-02-3657-3, 386-393, 1998.

[9] Mladenov V.M., N. Mastorakis, "Design of 2-Dimensional Recursive Filters by using Neural Networks", CSCC’99, Athens, Greece, "Computational Intelligence and Applications", (N.Mastorakis Editor), WSES Press, Athens 1999, pp. 12-20.

[10] D.G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, 1984.

[11] E. Polak, *Optimization: Algorithms and Consistent Approximations*, Springer, 1997.