



Numerical solutions for constrained quadratic problems using high-performance neural networks

A. Malek *, H.G. Oskoei

Department of Mathematics, Tarbiat Modarres University, P.O. Box 14115-175 Tehran, Iran

Abstract

Two new classes of neural networks for solving constrained quadratic programming problems are presented. The main advantage of these networks is the requirement to use economic analog multipliers for variables. The numerical simulations demonstrate that in the new neural networks not only the cost of the hardware implementation is not relatively expensive, but also accuracy of the solution is greatly good. The network dynamic behaviors are discussed. The numerical simulations are shown that, an optimal solution of the quadratic problems is an equilibrium point of the neural dynamics, and vice versa. We show that these networks find the solution of both primal and dual problems, and converge to the corresponding exact solutions globally. The proposed new neural networks models are fully stable.

© 2005 Published by Elsevier Inc.

Keywords: Quadratic programming; Neural networks; Dynamical systems

* Corresponding author.

E-mail address: mala@modares.ac.ir (A. Malek).

1. Introduction

Numerical computation in many disciplines, such as physics, applied mathematics, robotics, satellite guidance economics, etc. [1,2], has received a great deal of attention recently as a practical technique to understand complex phenomena that are almost impossible to treat analytically. In the last 50 years, researchers have proposed various dynamic solvers for solving linear, quadratic, and convex programming problems [3–6]. Dynamic solvers or analog computer, was first proposed by Dennis [7], and later studied by Rybashov [4,5], Karpinskaya [6], and others [1,2,8–12].

An online optimizer is desirable for many real time applications with time-dependent cost functions. One possible approach to solve this problem is to employ neural network on the basis of an analog circuit [8,13,14].

Most of the optimization problems with nonlinear objective functions are usually approximated by a second-order systems and solved numerically by a quadratic programming technique [15–17]. These traditional quadratic programming methods typically involve an iterative process, but long computational time limits their usage. The most important advantages of the neural networks are massively parallel processing and fast convergence. Thus algorithms of the neural networks have many computational advantages over the traditional algorithms.

Most of the existed neural networks, proposed for the solution of quadratic programming problems, contain some penalty parameters, the stable equilibrium points of which are the solutions of optimization problems only when the penalty parameters are infinite. This is almost impossible when we solve the problem numerically. Hence, we seek for some new neural networks with no network parameters.

Among those existing neural networks to the solution of quadratic problems, there are five widespread networks, proposed by Kennedy and Chua [1], Maa and Shanblatt [18], Chen and Fang [19], Wu et al. [20] and Xia [21]. All of these networks are depended to the network parameters or use very expensive analog multipliers.

In this paper we will propose two new classes of high-performance networks with economic analog multipliers (for variables) that solve the quadratic problems efficiently. Using economic analog multipliers is necessary, because too many complex network models, not only cause the high cost of hardware implementation, but also greatly affect the accuracy of solution. Our networks are designed in a way that one can easily use it to solve constrained linear programming problems, since in these proposed models, standard linear programming problem is considered as a specific simpler quadratic programming problem with no second order term. Our models give efficient solution for both primal and dual variables, since it uses the primal and dual properties of the problem simultaneously. For the linear primal-dual solution see Malek's

models [22]. Here in this article we introduce the new models that need not network parameters.

The following section lays the basic problems and mathematical formulation. Section 3 contains the two classes of new models. Section 4 contains a comparative study of the models. Section 5 contains simulation and numerical results.

2. The basic problem formulation

We are concerned with quadratic programming problems of the following general form:

$$\begin{aligned} \text{Min } Q(x) &= \frac{1}{2}x^T Ax + c^T x \\ \text{s.t. } Dx &= b, \\ x &\geq 0, \end{aligned} \tag{1}$$

where $A_{m \times m}$ as a symmetric positive semi-definite matrix, D is an $n \times m$ matrix, $x = (x_1, x_2, \dots, x_m)^T$ and $c = (c_1, c_2, \dots, c_m)^T$. In the following, we denote $\|\cdot\|_2$ as the Euclidean norm. We define the dual problem as follows:

$$\begin{aligned} \text{Max } \hat{Q}(x) &= b^T y - \frac{1}{2}x^T Ax \\ \text{s.t. } D^T y &\leq \nabla Q(x), \end{aligned} \tag{2}$$

where $\nabla Q(x) = Ax + c$, $b \in \mathfrak{R}^n$ and $y \in \mathfrak{R}^n$ is free in sign. Assume that the solution set

$$\Omega = \{(x, y) | y \in \mathfrak{R}^n, x \in \mathfrak{R}^m, x \geq 0\}$$

for the quadratic problems (1) and (2) is nonempty. For convenience, define vector

$$(x)^+ = [(x_1)^+, \dots, (x_m)^+]^T,$$

where $(x_i)^+ = \max\{0, x_i\}$ for $i = 1, \dots, m$.

Quadratic form in problem (1) is in a general form, for example if A is a positive semi-definite matrix, but it is not symmetric, problem (1) can easily reformulated [23], by choosing $\hat{A} = \frac{1}{2}(A + A^T)$ instead of A .

Wu et al. [20] in 1996 proposed the following neural network model to solve problems (1) and (2)

$$\frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = - \left\{ \begin{array}{l} \beta(-D^T y + Ax + c) + \beta A[x - (x + D^T y - Ax - c)^+] \\ + D^T(Dx - b) \\ \beta\{Dx - b + D[(x + D^T y - Ax - c)^+ - x]\} \end{array} \right\}, \tag{3}$$

where $\beta = \|x - (x + D^T y - Ax - c)^+\|_2^2$.

Xia [21] considered the adjusted form of model (3) as follows:

$$\frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = - \left\{ \begin{array}{l} (I + A)[x - (x + D^T y - Ax - c)^+] + D^T(Dx - b) \\ -D[x - (x + D^T y - Ax - c)^+] + Dx - b \end{array} \right\}, \quad (4)$$

where I is the identity matrix. For simplicity assume that

$$r = (x + D^T y - Ax - c)^+ \quad \text{and} \quad \lambda = D^T(Dx - b),$$

thus models (3) and (4) are in the following forms respectively:

$$\frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} -\beta(D^T y - Ax - c) + \beta A(r - x) - \lambda \\ -\beta(Dr - b) \end{bmatrix}, \quad (5)$$

$$\frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} (I + A)(r - x) - \lambda \\ -Dr + b \end{bmatrix}. \quad (6)$$

Comparing models (5) and (6), it is found that network in (5) uses more expensive analog multipliers for variables than model (5) and therefore it needs relatively expensive hardware instruments.

3. Class of new models

In this section we introduce a class of first new models containing two new neural networks. New1 neural network is given in the following form:

$$\frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = - \left\{ \begin{array}{l} -D^T y + Ax + c + A[x - (x + D^T y - Ax - c)^+] \\ + D^T(Dx - b) \\ D(x + D^T y - Ax - c)^+ - b \end{array} \right\}. \quad (7)$$

It is a simplified model of Wu et al. (see (3)). Here we have been concerned of obtaining better accuracy for the final solutions, while we do not use expensive analog multipliers of Wu et al. and therefore our network model is simpler in the manipulation of hardware tools. In the Tables 3–6 we show that for the example problem in Section 4. New1 model converges to the exact solution with 13 exact decimal points. While in the same conditions the solutions for neural network proposed by Wu only agrees up to 3 decimal points with the corresponding exact solution.

It is still possible to simplify New1 model. The New2 model has the advantage of serious simplification and good accuracy in the same time. New2 neural network model is in the form

$$\frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = - \left\{ \begin{array}{l} (I + A)[x - (x + D^T y - Ax - c)^+] \\ D(x + D^T y - Ax - c)^+ - b \end{array} \right\}. \quad (8)$$

The network circuit implementation for solving (1) and (2) whose dynamics are governed by (8) are given in Fig. 1. The circuit consists of adders (summing amplifiers) and integrators. In Fig. 1, vectors c and b are external input vectors, while x and y are the network outputs. In this diagram dynamical process of vector r is the same as what is given in [21]. A simplified block diagram of r is illustrated in Fig. 2.

We present another model New3 in (9) which appears to be more efficient than the New1 and New2 models when we investigate the complexity, complexity of individual neurons, stability, and accuracy of the solutions, (see Tables 3–6).

$$\frac{d}{dt} \begin{pmatrix} x \\ y \end{pmatrix} = - \begin{Bmatrix} x - (x + D^T y - Ax - c)^+ \\ D[(x + D^T y - Ax - c)^+] - b \end{Bmatrix}. \tag{9}$$

This neural network model does not need to use λ and therefore in practice needs relatively less computational efforts. Moreover, this model is globally convergent to the solution set of the programming problem (1) and (2), i.e. it will converge to the corresponding exact solution independent of where and how to choose the starting input initial values. New3 model not only has the same global convergence property as the model (6), but also has some more advantages, plus simplicity. To prove the globally convergent properties of model (9), consider the following theorem.

Theorem 1. *The neural network (9) is globally convergent to the solution set of the primal and dual quadratic programming problems (1) and (2).*

Proof. Let in the proposed model of Tao et al. [24], general projection operator to be the identity operator. Then the proof is similar to Tao’s proof. \square

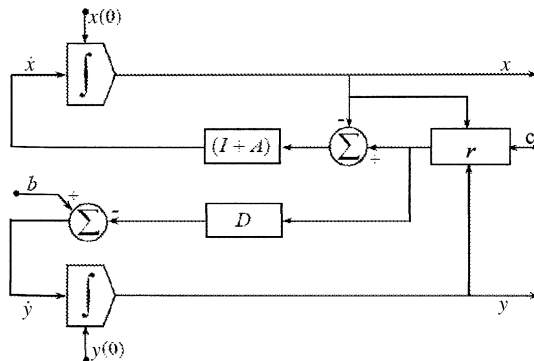


Fig. 1. A simplified neural network diagram for New2 model.

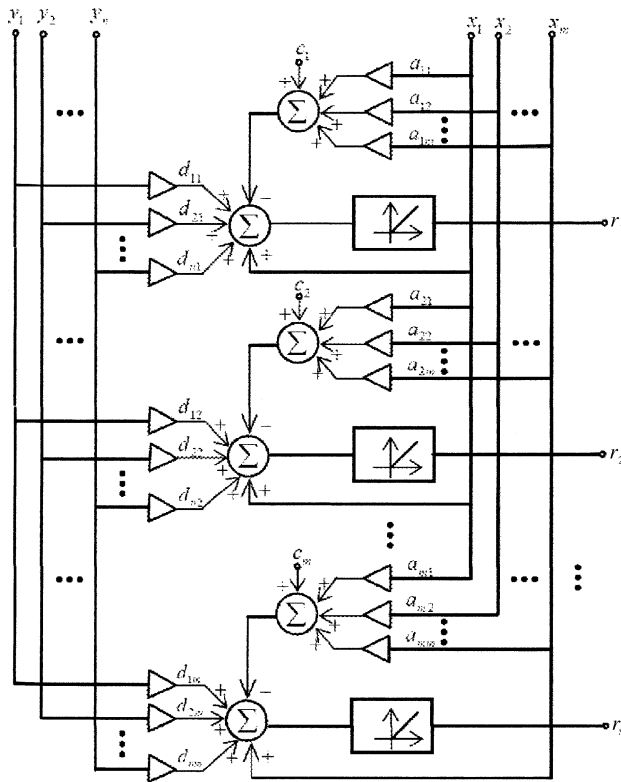


Fig. 2. A simplified block diagram for r , where $A = (a_{ij})$ and $D = (d_{ij})$.

New3 model do not use any projection operator in practice thus it is different and simpler from the model proposed by Tao et al. Here in model (9), unlike the Tao’s model we do not use any extension of Newton’s optimal descent flow equation to solve the problem.

4. A comparative study

Note that we do not include the Tank and Hopfield, Chua and Lin, Kennedy and Chua, Maa and Shanblatt and Chen and Fang network models in our study. The reason is as follows: The Tank and Hopfield neural network model [8] is a specific implementation of the dynamical canonical nonlinear programming circuit of Chua and Lin [13]. Therefore, the Tank and Hopfield neural network model can be derived from the Kennedy and Chua [1] neural model. On the other hand, the dynamics of the network of Kennedy and Chua

converges to only approximate solutions, and has the difficulty produced by choosing penalty parameters.

Phase-1 of Maa and Shanblatt network is the same as Kennedy and Chua model, but phase-2 of Maa and Shanblatt has difficulty choosing parameters, and can not guarantee to converge globally. Chen and Fang's model, [19] is a generalization of Kennedy and Chua's model, uses only simple hardware and no analog multipliers are required. Yet the network has difficulty choosing penalty parameters.

We now compare the networks (5) and (6) with our proposed networks in (7)–(9) for solving problems (1) and (2). The network proposed by Xia (6) is stable to exact solution and there are no parameters to set, but the main disadvantage of it is that too many expensive analog multipliers (like r and λ) are required for large scale quadratic programming problems, thus the set of hardware implementation is expensive, and therefore greatly affect the accuracy of solutions. Neural network model (7) of us is compared with (5), in Section 3, (see also Tables 3–6). In (8) one does not need to compute all those calculations required in (6) connected to the computational activities for λ (see Fig. 1). Thus (8) is easier to use and has the same accuracy as (6), (see Tables 3–6). Neural network (9) proposed by us is better than our network (8) in the sense of complexity, i.e. usage of analog multipliers and hardware implementations (see Tables 3–6). Thus, all together, we recommend neural network (9) since in globally convergence, accuracy and complexity has better performance over all other mentioned models. Simulation and numerical results are discussed in the next section.

5. Simulation and numerical results

In this section, we illustrate the performance of the five described quadratic problem solvers on a numerical example. We use our written software package, with the fourth-order Runge–Kutta algorithm. The step size is 1/10. The example problem is adapted from Wu et al. [20], and stated below:

Example 1. Consider the following constrained primal quadratic programming problem (QP) and its dual (DQP):

$$\begin{aligned}
 \text{(QP)} \quad & \text{Min} \quad x_1^2 + x_2^2 + x_1x_2 - 30x_1 - 30x_2 \\
 & \text{s.t.} \quad \frac{5}{12}x_1 - x_2 + x_3 = \frac{35}{12}, \\
 & \quad \quad \frac{5}{2}x_1 + x_2 + x_4 = \frac{35}{2},
 \end{aligned}$$

$$\begin{aligned}
 &x_5 - x_1 = 5, \\
 &x_2 + x_6 = 5, \\
 &x_i \geq 0 \quad (i = 1, 2, \dots, 6), \\
 \text{(DQP) Max} & \quad \frac{35}{12}y_1 + \frac{35}{2}y_2 + 5y_3 + 5y_4 - x_1^2 - x_2^2 - x_1x_2 \\
 \text{s.t.} & \quad \frac{5}{12}y_1 + \frac{5}{2}y_2 - y_3 - 2x_1 - x_2 \leq -30, \\
 & \quad -y_1 + y_2 + y_4 - x_1 - 2x_2 \leq -30, \\
 & \quad y_1 \leq 0, \\
 & \quad y_4 \leq 0.
 \end{aligned}$$

Table 1 shows the optimal values for both problems (QP) and (DQP).

Our models are designed to solve problems (QP) and (DQP) simultaneously, we therefore are going to show the effect of choosing feasible or infeasible starting points on the convergence, stability and exactness of the solution. The starting points are chosen to be in the both feasible and infeasible region of the primal or dual problem. Four cases for the starting points are considered and they are shown in Table 2.

Table 1
The exact optimal solutions

Exact primal variables x^*	Exact dual variables y^*
5	
5	0
5.833333	-6
0	0
10	9
0	
Exact primal value (objective function)	Exact dual value (objective function)
-225	-225

Table 2
Four ways of choosing initial values for primal and dual variables

Table	Figure	Feasible or infeasible (primal)	Feasible or infeasible (dual)	Initial values (primal)	Initial values (dual)
3	3	Feasible	Feasible	$(-5, -5, 0, 35, 0, 10)^T$	$(0, -12, 24, -35)^T$
4	4	Feasible	Infeasible	$(-5, -5, 0, 35, 0, 10)^T$	$(0, -1, 0, -2)^T$
5	5	Infeasible	Feasible	$(3, 0, 7, 5, -4, 1)^T$	$(0, -12, 24, -35)^T$
6	6	Infeasible	Infeasible	$(3, 0, 7, 5, -4, 1)^T$	$(0, -1, 0, -2)^T$

Figs. 3–6 show the various state trajectories from different initial points for both primal and dual variables of model (9). We see that New3 neural network model (9) converges quickly ($t = 10$) to the exact solutions, moreover it is fully

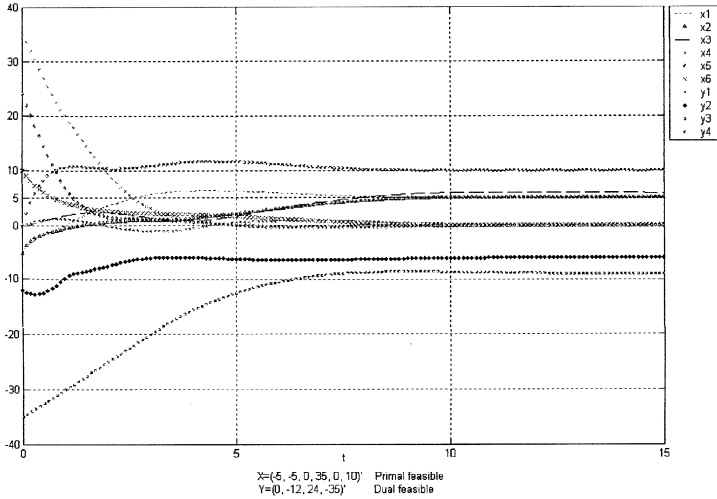


Fig. 3. Trajectories of example problem (QP) and (DQP) for the given X and Y initial vectors ($t = 15$).

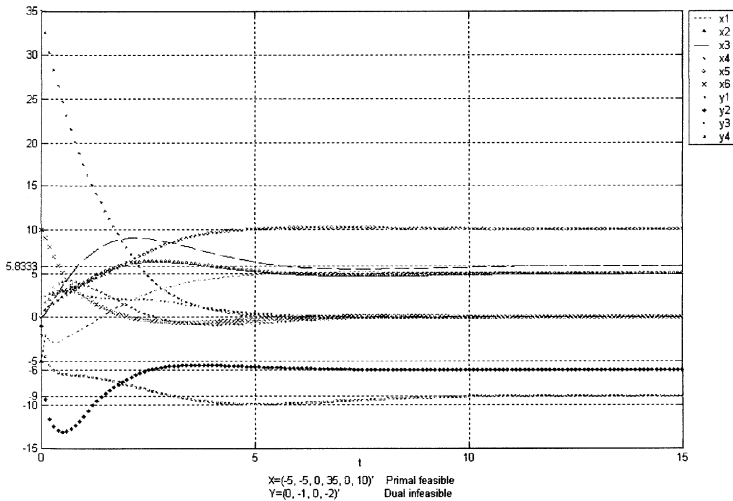


Fig. 4. Trajectories of example problem (QP) and (DQP) for the given X and Y initial vectors ($t = 15$).

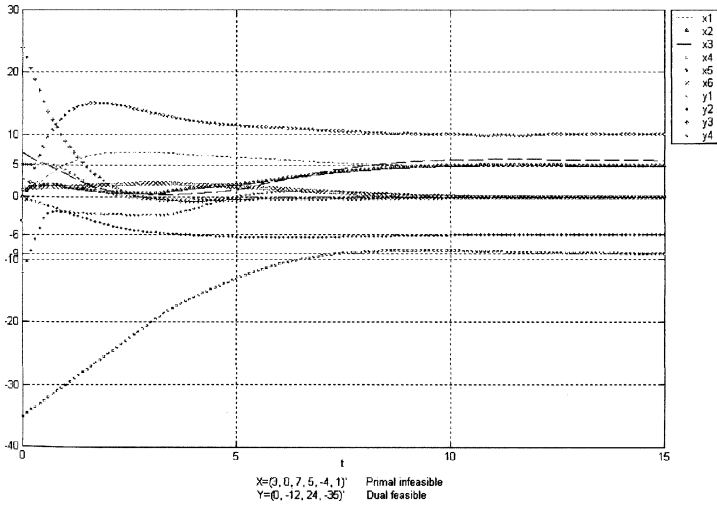


Fig. 5. Trajectories of example problem (QP) and (DQP) for the given X and Y initial vectors ($t = 15$).

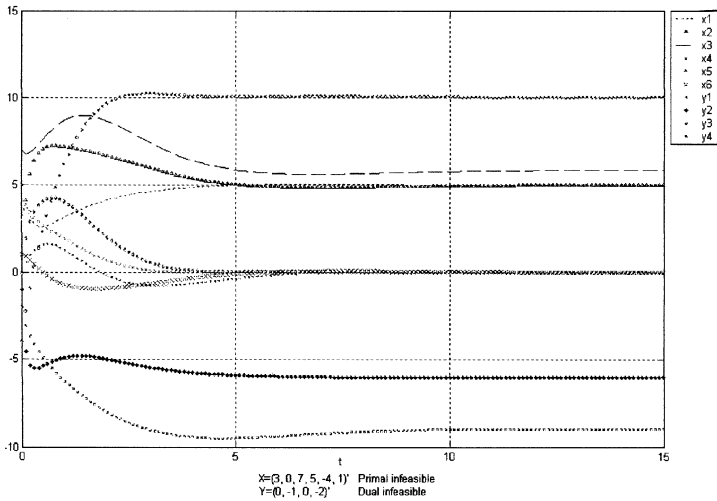


Fig. 6. Trajectories of example problem (QP) and (DQP) for the given X and Y initial vectors ($t = 15$).

stable. Figs. 7 and 8 show that for model (8) the state trajectories always converges to the exact solution independent of how to choose the initial points, feasible or infeasible. Fig. 7 shows five arbitrary feasible initial starting values

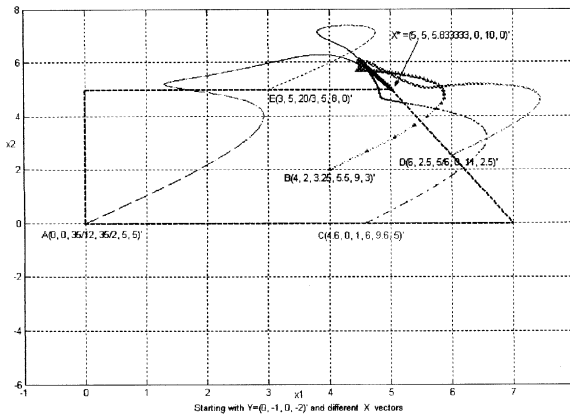


Fig. 7. Numerical results for New2 model using feasible initial points. Five trajectories of example problem (QP) and (DQP) starting with arbitrary fixed vector $Y = (0, -1, 0, -2)^T$ and various X vectors.

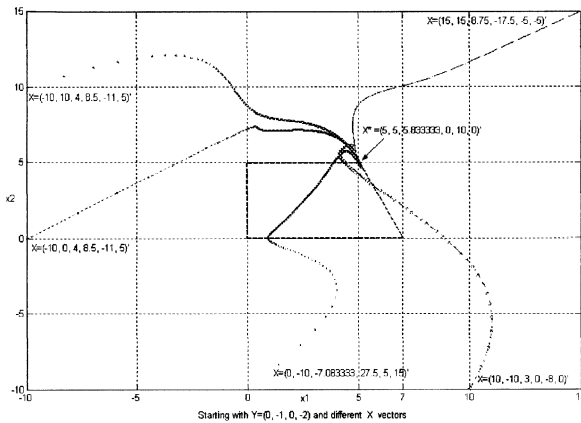


Fig. 8. Numerical results for New2 model using infeasible initial points. Five trajectories of example problem (QP) and (DQP) starting with arbitrary fixed vector $Y = (0, -1, 0, -2)^T$ and various X vectors.

for the example problem converging to the correct optimal solution, while Fig. 8, illustrate five infeasible arbitrary initial starting values converges to the exact solution.

To demonstrate and compare the behavior and properties of the proposed neural networks in this article, many problems solved. Finally numerical solutions for example problem (QP) and (DQP) using models (5)–(9) are given in Tables 3–6.

Table 3

Numerical results for (QP) and (DQP) problems at $t = 60$ for different neural network models where initial values are $x^0 = (-5, -5, 0, 35, 0, 10)^T$, $y^0 = (0, -12, 24, -35)^T$

NN	Optimal values for primal problem (x^*)	Optimal values for dual problem (y^*)	Optimal objective value (primal) Absolute error	Optimal objective value (dual) Absolute error	Complexity
Wu et al. model (5)	5.0001180876 4.9997428710 5.8329386074 0 10.0001529128 0	-0.0082511461 -5.996383688 0.0032528421 -9.0171838065	-224.9979143306 0.0020856693	-225.0283496493 0.0283496493	5
New1 model (7)	5.0000000000 4.9999999999 5.8333333333 0 10.0000000000 0	-4.39068402e-012 -5.9999999999 1.81121400e-012 -9.0000000000	-225 1.13686837e-013	-225.0000000000 8.41282599e-012	4
Xia model (6)	4.9999999999 5.0000000000 5.8333333333 1.30263732e-012 10.0000000000 -2.01427941e-011	-6.14541363e-011 -5.9999999999 2.63102380e-011 -9.0000000000	-225.00000000111 1.10958353e-010	-225.0000000001 1.21303855e-010	3

New2 model (8)	4.999999999		-225.000000003	-225.000000001	2
	5.000000000	-6.95673123e-011			
	5.833333334	-5.999999999			
	1. 85089119e-033	2.73464745e-011	3.16106252e-010	1.56887836e-010	
	9.999999999	-9.000000000			
	1.80498218e-034				
New3 model (9)	5.000000000		-224.999999999	-225.000000001	1
	4.999999999	-1.47815477e-011			
	5.833333332	-5.9999999998			
	1.64400253e-033	1. 82228911e-012	9.28821464e-011	1.31393562e-010	
	10.000000000	-9.000000000			
	1.80498218e-034				

Table 4

Numerical results for (QP) and (DQP) problems at $t = 60$ for different neural network models where initial values are $x^0 = (-5, -5, 0, 35, 0, 10)^T$, $y^0 = (0, -1, 0, -2)^T$

NN	Optimal values for primal problem (x^*)	Optimal values for dual problem (y^*)	Optimal objective value (primal) Absolute error	Optimal objective value (dual) Absolute error	Complexity
Wu et al. model (5)	5.000090751 4.9998034595 5.8330321225 0 10.0001197675 0	-0.0071409798 -5.9967327285 0.0031041751 -9.0149982375	-224.9984131401 0.0015868598	-225.0215341173 0.0215341173	5
New1 model (7)	4.9999999999 5.0000000000 5.8333333333 0 9.9999999999 0	-5.88453718e-013 -5.9999999999 2.89460588e-013 -8.9999999999	-225.0000000000 1.22213350e-011	-225 3.97903932e-013	4
Xia model (6)	4.9999999999 5.0000000000 5.8333333333 3.990592539e-012 9.9999999999 -1.72532936e-011	-1.18676920e-011 -5.9999999999 5.99806465e-012 -8.9999999999	-225.0000000002 2.23280949e-010	-225.0000000000 1.46656020e-011	3

New2 model (8)	4.999999999		−225.000000000	−224.999999999	2
	5.000000000	1.84478664e−011			
	5.833333333	−6.000000000			
	2.10404664e−033	−5.16284130e−012	9.19442300e−011	6.30109298e−011	
	9.999999999	−8.999999999			
	2.34913816e−034				
New3 model (9)	4.999999999		−225.000000003	−225.000000000	1
	5.000000000	−4.20303163e−011			
	5.833333333	−5.999999999			
	1.56947942e−033	1.93772650e−011	3.31738192e−010	8.37871994e−011	
	9.999999999	−9.000000000			
	2.11029084e−034				

Table 5

Numerical results for (QP) and (DQP) problems at $t = 60$ for different neural network models where initial values are $x^0 = (3, 0, 7, 5, -4, 1)^T$, $y^0 = (0, -12, 24, -35)^T$

NN	Optimal values for primal problem (x^*)	Optimal values for dual problem (y^*)	Optimal objective value (primal) Absolute error	Optimal objective value (dual) Absolute error	Complexity
Wu et al. model (5)	5.0001192449 4.9997403342 5.8329347049 0 10.0001543758 0	-0.0082922819 -5.9963675610 0.0032650795 -9.0172693436	-224.9978936368 0.0021063631	-225.0285331995 0.0285331995	5
New1 model (7)	5.0000000000 5.0000000000 5.8333333333 0 10.0000000000 0	-5.12802229e-012 -5.9999999999 2.12851633e-012 -9.0000000000	-225.0000000000 3.92219590e-012	-225.0000000000 9.60653778e-012	4
Xia model (6)	4.9999999999 5.0000000000 5.8333333333 2.15215012e-012 10 -2.51939960e-011	-6.94480294e-011 -5.9999999999 2.98985971e-011 -9.0000000000	-225.0000000001 1.62174274e-010	-225.0000000001 1.35571553e-010	3

New2 model (8)	4.999999999		−225.0000000004	−225.0000000001	2
	5.000000000	−8.92239124e−011			
	5.833333334	−5.999999999			
	9.02491092e−035	3.56874380e−011	4.56850557e−010	1.94916083e−010	
	9.999999999	−9.000000000			
	1.80498218e−035				
New3 model (9)	5.000000000		−224.999999999	−225.00000000216	1
	4.999999999	−3.13753239e−011			
	5.833333332	−5.999999999			
	9.02844396e−035	6.94596776e−012	6.77857769e−011	2.1 6004991e−010	
	10.000000000	−9.000000001			
	1.80498218e−035				

Table 6

Numerical results for (QP) and (DQP) problems at $t = 60$ for different neural network models where initial values are $x^0 = (3, 0, 7, 5, -4, 1)^T$, $y^0 = (0, -1, 0, -2)^T$

NN	Optimal values for primal problem (x^*)	Optimal values for dual problem (y^*)	Optimal objective value (primal) Absolute error	Optimal objective value (dual) Absolute error	Complexity
Wu et al. model (5)	5.0001254936 4.9997269386 5.8329142368 0 10.0001629143 0	-0.0084856449 -5.9962597130 0.0033900306 -9.0177143440	-224.9977864274 0.0022135725	-225.0287028834 0.0287028834	5
New1 model (7)	4.9999999999 5.0000000000 5.8333333333 0 9.9999999999 0	-2.78927526e-013 -5.9999999999 1.36681625e-013 -8.9999999999	-225.0000000000 5.7411853049e-012	-225 2.2737367544e-013	4
Xia model (6)	4.9999999999 5.0000000000 5.8333333333 1.69404522e-012 9.9999999999 -7.80332135e-012	-6.93493967e-012 -5.9999999999 3.34716313e-012 -8.9999999999	-225.0000000000 9.6065377874e-011	-225.0000000000 1.0061285138e-011	3

New2 model (8)	4.9999999999		–225.0000000001	–225.0000000000	2
	5.0000000000	–1.01132467e–011			
	5.8333333333	–5.9999999999			
	9.41733623e–035	5.01038519e–012	1.3304202184e–010	1.2192913345e–011	
	9.9999999999	–8.9999999999			
	1.80498218e–035				
New3 model (9)	4.9999999999		–225.0000000001	–225.0000000000	1
	5.0000000000	–2.43092294e–011			
	5.8333333333	–5.9999999999			
	9.31312930e–035	9.28243240e–012	1.1110046216e–010	8.7737817011e–011	
	10.0000000000	–9.0000000000			
	1.80498218e–035				

The results of computer simulations indicate that the proposed neural networks of us are efficient techniques (see [Tables 3–6](#)), and their convergence to the exact solution is not dependent to the initial values of the starting points. This is due to the fact that we considered both primal and dual problems in our analysis, and thus our models are based on the primal-dual solution of the general quadratic problem.

The model complexity is ranked between 1 and 5, according to the usage of analog multipliers and hardware implementations, i.e. as much as the number is greater, the model uses more analog multipliers and therefore it needs relatively more expensive hardware implementations. From [Tables 3–6](#) we see that our New3 neural network has the better performance among all of the other models. It gives solution for both primal and dual problem independent of the initial starting point. It is simple to use, not only converges more quickly but also results in higher accuracy. It is fully stable and converges to the exact solution globally. It is less complex model and therefore it is ranked 1 when complexity is concerned. This is due to the fact that we do not process λ in each of the iterations, and this will save lots of computational efforts where we are in need of thousands of iterations to solve the problem.

References

- [1] M.P. Kennedy, L.O. Chua, Neural networks for nonlinear programming, *IEEE Trans. Circ. Syst.* 35 (1988) 554–562.
- [2] A. Rodriguez-Vazquez, Dominguer-Castro, A. Rueda, J.L. Huertas, E. Sanchez-Sinenico, Nonlinear switched-capacitor “neural” networks for optimization problems, *IEEE Trans. Circ. Syst.* 37 (1990) 384–397.
- [3] I.B. Pyne, Linear programming on an electronic analogue computer, *Trans. Amer. Inst. Elec. Eng.* 75 (1956) 139–143.
- [4] M.V. Rybashov, The gradient method for solving convex programming problems on electronic analog computers, *Automat. Remote Contr.* 26 (11) (1965) 1886–1898.
- [5] M.V. Rybashov, Gradient method for solving convex programming problems on electronic analog computers, *Automat. Remote Contr.* 26 (12) (1965) 2079–2089.
- [6] N.N. Karpinskaya, Method of “Penalty” functions and the foundations of Pyne’s method, *Automat. Remote Contr.* 28 (1) (1967) 124–129.
- [7] J.B. Dennis, *Mathematical Programming and Electrical Networks*, Chapman and Hall, London, 1959.
- [8] D.W. Tank, J.J. Hopfield, Simple neural optimization networks: an A/D converter, signal decision network, and linear programming circuit, *IEEE Trans. Circ. Syst. CAS* 33 (1986) 533–541.
- [9] W.E. Lillo, M.H. Loh, S. Hui, S.H. Zak, On solving constrained optimization problems with neural networks: a penalty function method approach, *Tech. Rep. TREE 91-43*, Purdue Univ., W. Lafayette. IN, October 1991.
- [10] A. Cichocki, R. Unbehauen, Neural networks for solving systems of linear equations and related problems, *IEEE Trans. Circ. Syst.—I: Fund. Theory App.* 39 (1992) 124–138.

- [11] S.H. Zak, V. Upatising, W.E. Lillo, S. Hui, A dynamical systems approach to solving linear programming problems, in: K.D. Elworthy, W.N. Everitt, E.B. Lee (Eds.), *Differential Equations, Dynamical Systems, and Control Science*, Marcel Dekker, New York, 1994.
- [12] S.H. Zak, V. Upatising, S. Hui, Solving linear programming problems with neural networks: a comparative study, *IEEE Trans. Neural Networks* 6 (1) (1995) 96–104.
- [13] L.O. Chua, G.N. Lin, Nonlinear programming without computation, *IEEE Trans. Circ. Syst. CAS* 31 (1984) 182–188.
- [14] G. Wilson, Quadratic programming analogs, *IEEE Trans. Circuits Syst. CAS* 33 (9) (1986) 907–911.
- [15] M.S. Bazaraa, C.M. Shetty, *Nonlinear Programming, Theory and Algorithms*, John Wiley and Sons, New York, 1990.
- [16] D.G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA, 1973, Chapter 12.
- [17] S.S. Rao, *Optimization, Theory and Applications*, Wiley Eastern Limited, New Delhi, 1984.
- [18] C.Y. Maa, M.A. Shablatt, A two-phase optimization neural network, *IEEE Trans. Neural Network* 3 (6) (1992).
- [19] Y.-H. Chen, S.-C. Fang, Solving convex programming problems with equality constraints by neural networks, *Comp. Math. Appl.* 36 (7) (1998) 41–68.
- [20] X.Y. Wu, Y.S. Xia, J. Li, W.K. Chen, A high performance neural network for solving linear and quadratic programming problems, *IEEE Trans. Neural Networks* 7 (3) (1996) 643–651.
- [21] Y. Xia, A new neural network for solving linear and quadratic programming problems, *IEEE Trans. Neural Networks* 7 (6) (1996) 1544–1547.
- [22] A. Malek, A. Yari, Primal–dual solution for the linear programming problems using neural networks, *Appl. Math. Comput.*, in press.
- [23] A. Bouzerdoum, T.R. Pattison, Neural network for quadratic optimization with bound constraint, *IEEE Trans. Neural Networks* 4 (1993).
- [24] Q. Tao, J. Cao, D. Sun, A simple and high performance neural network for quadratic programming problems, *Appl. Math. Comput.* 124 (2001) 251–260.