Design of Recurrent Neural Networks for Solving Constrained Least Absolute Deviation Problems

Xiaolin Hu, *Member, IEEE,* Changyin Sun, *Member, IEEE,* and Bo Zhang

***Abstract*—Recurrent neural networks for solving constrained least absolute deviation (LAD) problems or L_1-norm optimization problems have attracted much interest in recent years. But so far most neural networks can only deal with some special linear constraints efficiently. In this paper, two neural networks are proposed for solving LAD problems with various linear constraints including equality, two-sided inequality and bound constraints. When tailored to solve some special cases of LAD problems in which not all types of constraints are present, the two networks can yield simpler architectures than most existing ones in the literature. In particular, for solving problems with both equality and one-sided inequality constraints, another network is invented. All of the networks proposed in this paper are rigorously shown to be capable of solving the corresponding problems. The different networks designed for solving the same types of problems possess the same structural complexity, which is due to the fact these architectures share the same computing blocks and only differ in connections between some blocks. By this means, some flexibility for circuits realization is provided. Numerical simulations are carried out to illustrate the theoretical results and compare the convergence rates of the networks.**

***Index Terms*—L_1-norm optimization, least absolute deviation (LAD), minimax optimization, recurrent neural network (RNN), stability analysis.**

I. Introduction

CONSIDER solving the following minimization problem:

$$\begin{aligned} &\min \ \|Ax-b\|_1 \\ &\text{subject to} \ \ Cx \in \Omega, x \in X \end{aligned} \tag{1}$$

where $x \in \Re^n$ is the unknown variable, $A \in \Re^{m\times n}, b \in \Re^m, C \in \Re^{r\times n}$ are given parameters, and X, Ω are two nonempty box sets defined as

$$X = \{x \in \Re^n | \rho \le x \le \varrho\}$$

$$\Omega = \{y \in \Re^r | l \le y \le h\}$$

where ρ, ϱ, l, h are vectors whose components are all constants. In above equations, the inequality signs between two vectors are understood componentwise. Note that some components of l and ρ can be $-\infty$ and some components of h and ϱ can be $+\infty$. So, the two-sided inequality constraints $Cx \in \Omega$, or $l \le Cx \le h$ can include both one-sided inequality constraints (l or h is infinity) and equality constraints ($l = h$) as special cases. Clearly $x \in X$ is also a special case of $Cx \in \Omega$, but as it can be handled more efficiently in many algorithms, it is separated from the inequality constraints and given a name *bound constraint*.

Equation (1) represents a very general L_1-norm optimization problem often encountered in signal processing. One of its most important applications is the estimation of model parameters in the presence of background noise. In that case, x stands for the vector of parameters to be estimated, A stands for the model matrix, b stands for the vector of observation and $e \triangleq b - Ax$ stands for the noise and errors contained in b. The goal is to estimate x in the constrained set $\{x \in X | Cx \in \Omega\}$ conforming to a certain criterion such as least square (LS), i.e., minimal $\|e\|_2$, or least absolute deviation (LAD), i.e., minimal $\|e\|_1$. In this sense, (1) is often called an LAD problem. Due to the smoothness of $\|e\|_2$ and nonsmoothness of $\|e\|_1$, the LS solution is easier to obtain than the LAD solution. However, it is well known that the LAD method performs much better than the LS method when the noise distribution is nonGaussian such as Laplace or Cauchy distribution, which is often the case in practice [2], [7], [14]. Moreover, the LAD method is more robust as it is less susceptible to outliers (bad data points) than the LS method.

Because of their excellent properties, LAD problems have been extensively studied and many numerical algorithms have been proposed for solving them (see [2], [24], [29], [30], and references therein). Recently, there is a surge of interest in designing recurrent neural networks (RNNs) for solving them. For example, Wang *et al.* [33] proposed an RNN for solving unconstrained LAD problems (all constraints are absent). For another example, Xia [35] and Kamel [36] proposed two RNNs for solving LAD problems with bound constraint only. The most powerful network so far might be the one devised in [34] as it is claimed to be capable of handling both equality constraints and inequality constraints (but this claim

Manuscript received April 10, 2009; revised April 3, 2010. Date of publication June 17, 2010; date of current version July 8, 2010. This work was supported in part by the National Natural Science Foundation of China, under Grants 60 805 023, 60 621 062, and 60 605 003, by the National Key Foundation Research and Development Project, under Grants 2003CB317 007, 2004CB318 108, and 2007CB311 003, by the China Post-Doctoral Science Foundation, under Grants 20 080 430 032 and 200 801 072, and by the Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList).

X. Hu and B. Zhang are with the State Key Laboratory of Intelligent Technology and Systems, TNList, and the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: xiaolin.hu@gmail.com; dcszb@tsinghua.edu.cn).

C. Sun is with the School of Automation, Southeast University, Nanjing 210096, China (e-mail: cysun@seu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2010.2048123

needs further investigation; see Section III-B). A distinguished property shared by these RNNs is their dynamic nature in the process of searching for solutions, which makes them possible to be implemented physically by designated hardware such as application-specific integrated circuits, where the processing is truly done in a parallel manner. This *hardware solver* concept has stimulated the emergence of many RNNs for solving various optimization-related problems (see [1], [4]–[6], [8]–[13], [16]–[19], [25], [28], [32], and [37]–[39]) not limited to LAD problems since the mid 1980s when Hopfield and Tank published some milestone articles [15], [31]. A variety of RNNs have been proposed for solving linear and nonlinear programming problems [1], [6], [8], [10], [11], [13], [19], [25], [37], [38], combinatorial optimization problems [9], [28], [32], variational inequalities [4], [12], [17], [18], algebraic problems [5], [39], and so on (see also references therein).

For solving the LAD problems considered in [33], [35], [36], the RNNs proposed in these articles have many excellent properties. For instance, all of them can guarantee the global convergence of the output to exact solutions without any control circuits for tuning parameters during the dynamic evolving process. However, these models may fail to produce efficient circuits when a generally constrained LAD problem like (1) is encountered. In fact, none of the models in [33], [35], [36] can solve an LAD problem with all types of constraints including bound, inequality and equality constraints. Though the model proposed in [34] may solve such problems, its dimension must be very high because it has to convert the bound and two-sided inequality constraints into one-sided inequality constraints, which is undesired for circuits realization. Even worse, its ability for handling inequality constraints is questionable. In Section III-B, a simple example will be used to illustrate this point. On the other hand, even for solving the special problems considered in these references, the proposed networks are not simplest and there leaves much space for improvement (see Section V for simpler architectures).

In this paper, we aim at answering the following two questions: 1) Is there an RNN capable of solving the most general LAD problem (1) with low structural complexity? 2) Are there simpler RNNs compared with existing ones [33], [35], [36] for solving the same special LAD problems?

Meanwhile, we are not satisfied with designing just one model for solving one problem; instead, we will endeavor to find alternatives for solving every problem that we are interested in. This is quite useful for circuits practitioners to choose appropriate models for implementation, as in practice some physical limitations may make a particular implementation inefficient. The idea for designing alternatives to a model that has been proved to be feasible is to change some connections in the feasible model without, or with least, insertion of additional blocks. This idea has gained successes in designing alternative RNNs for linear and quadratic programming recently [20]–[22].

The rest of this paper is organized as follows. In Section II, the optimality conditions of the general LAD problem (1) are presented, which lay a base for RNN design in subsequent sections. In Section III, two efficient RNNs are presented for solving (1) with the discussion of their structural complexities. In Section IV, their performances are analyzed rigorously. In Section V, some important special cases of problem (1), as well as the corresponding RNNs, are discussed. Section VI presents the results of numerical simulations and Section VII concludes this paper.

Throughout this paper, it is always assumed that there exists at least one finite solution to the problem (1).

II. Problem Formulation

First of all we convert the LAD problem (1) into a minimax optimization problem. This is a regular strategy used in [33]–[36].

Lemma 1: The problem (1) is equivalent to the following optimization problem:

$$\begin{aligned} &\min \ \max_{y\in Y} y^T(Ax-b) \\ &\text{subject to} \ \ Cx\in\Omega, x\in X \end{aligned} \tag{2}$$

where $Y=\{y\in\Re^m|-1\le y_i\le 1, i=1,\ldots,m\}$.

Proof: The results can be established by using the same techniques as in [33]–[36]. ■

Then we define three piecewise linear functions $g_X(\cdot), g_Y(\cdot), g_\Omega(\cdot)$, which will be used extensively in what follows. $g_X(x)$ is defined as $(g_{X_1}(x_1),\ldots,g_{X_n}(x_n))^T$ where

$$g_{X_i}(x_i)=\begin{cases}\rho_i & x_i<\rho_i\\ x_i & \rho_i\le x_i\le\varrho_i\\ \varrho_i & x_i>\varrho_i.\end{cases} \tag{3}$$

The other two functions can be defined similarly. These functions are often called *projection operators* in the neural network context [4], [10]–[12], [17]–[19], [37], [38].

Theorem 1: A point $x^*\in\Re^n$ is a solution of problem (1) if and only if there exists $y^*\in\Re^m$ and $z^*\in\Re^r$ such that

$$\begin{cases} x^*=g_X(x^*-A^Ty^*+C^Tz^*)\\ y^*=g_Y(y^*+Ax^*-b)\\ Cx^*=g_\Omega(Cx^*-z^*).\end{cases} \tag{4}$$

Proof: According to Lemma 1, it is only needed to show that the results hold for the minimax problem (2). Note that $Cx\in\Omega$ can be written as $Cx=s, s\in\Omega$ by introducing a new variable s. Then the Lagrangian function associated with the minimax problem can be defined as

$$L(x,y,s,z)=y^T(Ax-b)-z^T(Cx-s).$$

According to the well-known saddle point theorem, a point $x^*\in\Re^n$ is a solution of problem (2) if and only if there exists $y^*\in\Re^m$ and $s^*, z^*\in\Re^r$ such that

$$L(x^*,y,s^*,z)\le L(x^*,y^*,s^*,z^*)\le L(x,y^*,s,z^*)$$

for $x\in X, y\in Y, s\in\Omega, z\in\Re^r$. Let $\phi(y,z)=-L(x^*,y,s^*,z)$, which is a linear function with respect to y and z. Clearly

$$\phi(y^*, z^*) = \min \phi(y, z) \quad \forall y \in Y,\ z \in \Re^r$$

which implies $\frac{\partial \phi(y,z)}{\partial z}|_{z=z^*} = Cx^* - s^* = 0$ and $(y - y^*)^T \frac{\partial \phi(y,z)}{\partial y}|_{y=y^*} \geq 0, \forall y \in Y$. The latter equation is known as a *variational inequality*, and a point y^* satisfies it if and only if $y^* = g_Y(y^* - \frac{\partial \phi(y,z)}{\partial y}|_{y=y^*}) = g_Y(y^* + Ax^* - b)$ [23]. Define another function $\psi(x, s) = L(x, y^*, s, z^*)$, which is linear in x and s. Clearly

$$\psi(x^*, s^*) = \min \psi(x, s) \quad \forall x \in X,\ s \in \Omega.$$

This is equivalent to $(x - x^*)^T \frac{\partial \psi(x,s)}{\partial x}|_{x=x^*} \geq 0, \forall x \in X$ and $(s - s^*)^T \frac{\partial \psi(x,s)}{\partial s}|_{s=s^*} \geq 0, \forall s \in \Omega$, or

$$(x - x^*)^T (A^T y^* - C^T z^*) \geq 0 \quad \forall x \in X$$
$$(s - s^*)^T z^* \geq 0, \quad \forall s \in \Omega.$$

The two equations imply $x^* = g_X(x^* - A^T y^* + C^T z^*)$ and $s^* = g_\Omega(s^* - z^*)$ [23]. Because $s^* = Cx^*$, the latter equality can be rewritten as $Cx^* = g_\Omega(Cx^* - z^*)$. The proof is completed. ■

III. Two Neural Network Models

A. Models

In this section, we present two RNNs for solving the LAD problem (1). The first one is governed by the following dynamic equations:

$$\textbf{NN-I:} \qquad \frac{d}{dt}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = -\lambda \begin{pmatrix} x - \bar{x} \\ 2(y - \bar{y}) \\ 2(C\bar{x} - \bar{z}) \end{pmatrix}$$

where

$$\bar{x} = g_X(x - A^T y + C^T z)$$
$$\bar{y} = g_Y(y + A\bar{x} - b)$$
$$\bar{z} = g_\Omega(C\bar{x} - z)$$

and $\lambda > 0$ is a constant for scaling the convergence rate. The output of the network is x. The second one is governed by the following dynamic equations:

$$\textbf{NN-II:} \qquad \frac{d}{dt}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = -\lambda \begin{pmatrix} 2(x - \tilde{x}) \\ y - \tilde{y} \\ Cx - \tilde{z} \end{pmatrix}$$

where

$$\tilde{x} = g_X(x - A^T \tilde{y} + C^T(z - Cx + \tilde{z}))$$
$$\tilde{y} = g_Y(y + Ax - b)$$
$$\tilde{z} = g_\Omega(Cx - z)$$

and $\lambda > 0$. The output of the network is also x. For convenience, in what follows the two networks, as well as their dynamic equations, are referred to as NN-I and NN-II, respectively.

The two sets of dynamic equations share much similarity, which can be understood as follows. If we let $\tilde{x}$ in NN-II be equal to $g_X(x - A^T \tilde{y} + C^T \tilde{z})$, then one iteration of NN-I and NN-II requires the same number of algebraic operations such as summations, multiplications and nonlinear transformations by appropriately arranging the computing orders (actually, in NN-I, $\bar{x}$ should be computed before $\bar{y}$ and $\bar{z}$; while in NN-II, $\tilde{x}$ should be computed after $\tilde{y}$ and $\tilde{z}$). In other words, NN-II entails just one more operation than NN-I, that is, adding $z - Cx$, which is available from the third line, to $\tilde{z}$ to compute $\tilde{x}$. This strong similarity stimulate us to ask whether some blocks of one architecture can be used by the other. The answer is yes.

Fig. 1 plots the architectures of the networks by block diagrams. In the figure, $\{a_{ji}\}_{m\times n} = A$, $\{c_{ki}\}_{r\times n} = C$, $\{b_j\}_{m\times 1} = b$. From the diagram, implementation of any model requires n amplifiers for realizing the activation function $g_X(\cdot)$, m amplifiers for the activation function $g_Y(\cdot)$ and r amplifiers for the activation function $g_\Omega(\cdot)$. (One should note that though the term $\bar{x}$ in NN-I appears also in the expressions of $\bar{y}$ and $\bar{z}$, it does not mean that the g_X term has to be implemented three times in circuits as it can be built as a common block. Likewise, $\tilde{y}$ and $\tilde{z}$ in NN-II are needed to be implemented once only.) The rest elements required are some conventionaly units for realizing weighted summations and integrations. Now a days very-large-scale integration technologies make this implementation to be an easy task. What we would like to emphasize here is that in circuits realization the two networks entail nearly the same number of elements including the amplifiers, capacitors, connections weights and so on. In fact, from the block diagrams, switching between the two schemes can be easily achieved by changing some connections and gain factors without adding or removing any major computing units. The only difference is that for implementing NN-I, the summator just before the "OUT6" port [see Fig. 1(a)] is redundant, whereas it is a must for NN-II. But in general this difference is negligible and we can claim that the two networks possess the same structural complexity.

B. Comparison With Existing Models

As pointed out in Section I, there exist several neural networks for solving different types of LAD problem, e.g., [33]–[36], and the first three can not solve the most general problem (1). Let us take a look at the last model, which was assumed to be capable of solving the LAD problems in the following form:

$$\begin{aligned} &\min \ \|Ax - b\|_1 \\ &\text{subject to} \ \ Ex \leq f \end{aligned} \tag{5}$$

where E and f are, respectively, a matrix and a vector with appropriate dimensions. Obviously, the problem (1) can be rewritten in the above form. The dynamics equations of the network proposed in [34] for solving (5) are

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ w \end{pmatrix} = -\lambda \begin{pmatrix} E^T((Ex - f)^+ - w) + A^T \tilde{y} \\ y - \tilde{y} - AQ \\ (Ex - f)^+ + EQ \end{pmatrix}$$

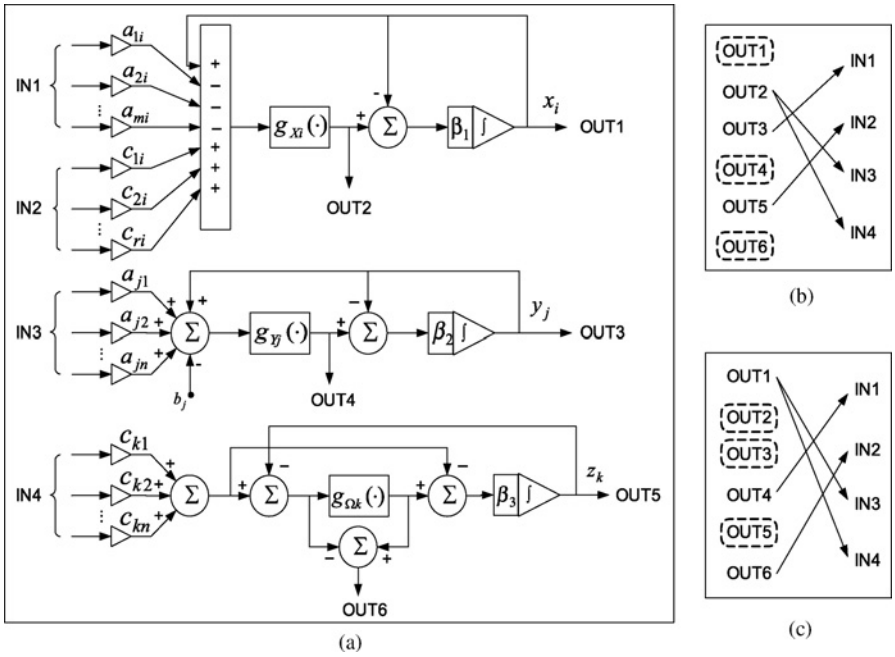


Fig. 1. Block diagrams of the neural networks NN-I and NN-II. (a) Basic blocks with their input and output ports, where $i = 1, \cdots, n$, $j = 1, \cdots, m$ and $k = 1, \cdots, r$. (b) and (c) Illustrate how the blocks should be connected to constitute NN-I and NN-II. The output ports surrounded by dashed rectangles do not need to be connected with other ports in the corresponding configuration. The gain factors in (a) for NN-I should be set as $\beta_1 = \lambda$, $\beta_2 = \beta_3 = 2\lambda$, while for NN-II should be set as $\beta_1 = 2\lambda$, $\beta_2 = \beta_3 = \lambda$. Clearly, transforming NN-I to NN-II, or the converse, entails just alternating a few gain factors and the starting points of some connections.

where $Q = E^T w - A^T y$, $\lambda > 0$ and $\tilde{y}$ is as in NN-II. It is easy to show that the equilibrium point set of the network may not correspond to the solution set of the problem. Consider a very simple problem in which x is a scalar and $A = E = 1, b = f = 0$. Clearly, the minimizer is $x = 0$. However, it is easy to check that $x = y = w = -1$ is also an equilibrium point of the network. So the performance of this network is questionable.

From Section II, it is seen that solving (1) is equivalent to solving the following minimax problem:

$$\min_{x\in X, s\in\Omega} \max_{y\in Y, z\in\Re^r} L(x, y, s, z) = y^T(Ax - b) - z^T(Cx - s)$$
$$= \begin{pmatrix} x \\ s \end{pmatrix}^T \begin{pmatrix} A^T & -C^T \\ 0 & I \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} - \begin{pmatrix} b \\ 0 \end{pmatrix}^T \begin{pmatrix} y \\ z \end{pmatrix}.$$

Then the two neural networks invented, respectively, in [13] and [11] can also solve the problem. Specifically, their dynamic equations become

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \\ s \end{pmatrix} = -\lambda \begin{pmatrix} x - \bar{x} \\ y - \tilde{y} \\ Cx - s \\ s - g_\Omega(s - z) \end{pmatrix} \tag{6}$$

and

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \\ s \end{pmatrix} = -\lambda \begin{pmatrix} 2(x - v) \\ y - \tilde{y} \\ Cx - s \\ 2(s - \tilde{z}) \end{pmatrix} \tag{7}$$

where $v = g_X(x - A^T\tilde{y} + C^T(z - Cx + s))$, and the other notations are the same as in NN-I and NN-II. Clearly, both of the networks are more complex than NN-I and NN-II. At least, they have one more state variable which entails more elements in hardware implementation. In addition, the first one can not guarantee the convergence to the solutions of the problem. For example, consider a very simple and special problem

$$\min \ \|x\|_1, \quad \text{subject to} \ \ x \in X = \Re.$$

Then (6) becomes

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = -\lambda \begin{pmatrix} y \\ y - g_{[-1,1]}(y + x) \end{pmatrix}.$$

It is easy to verify that when $x_0 = y_0 = 1/2$, the quantity $y + x$ will never exceed $[-1, 1]$ and the solution is

$$\begin{cases} x(t) = (\cos\lambda t - \sin\lambda t)/2 \\ y(t) = (\sin\lambda t + \cos\lambda t)/2. \end{cases}$$

Obviously, the state trajectory $x(t)$ will never converge to the desired value 0.

IV. Stability Analysis

In this section we will show that both of the neural network NN-I and NN-II are stable and globally convergent to a solution of problem (1). First, some preliminaries are introduced.

Lemma 2: Consider the function $g_X(\cdot)$ defined in (3).

1) For any $x \in \Re^n$ and any $v \in X$

$$(g_X(x) - x)^T(v - g_X(x)) \geq 0.$$

For any $x, y \in \Re^n$

$$\|g_X(x) - g_X(y)\|_2 \leq \|x - y\|_2.$$

2) The function $\psi(x) = \|x - g_X(x)\|_2^2$ is convex and continuously differentiable on $\Re^n$, whose gradient is given by $\nabla\psi(x) = 2(x - g_X(x))$.

The same properties hold for the functions $g_Y(\cdot)$ and $g_\Omega(\cdot)$.

Proof: 1) See [23]. 2) See Theorem 2.2 in [26]. ■

Note that (4) can be equivalently put into the following variational inequality form [23]:

$$\begin{cases} (x - x^*)^T(A^T y^* - C^T z^*) \geq 0 \quad \forall x \in X \\ (y - y^*)^T(-Ax^* + b) \geq 0 \quad \forall y \in Y \\ (z - Cx^*)^T z^* \geq 0 \quad \forall z \in \Omega. \end{cases} \tag{8}$$

Theorem 2: A point $x^* \in \Re^n$ is a solution of problem (1) if and only if there exists $y^* \in \Re^m$ and $z^* \in \Re^r$ such that they constitute an equilibrium point of NN-I and NN-II.

Note that there exists at least one finite point in the equilibrium sets of NN-I and NN-II since it is assumed that there exists at least one finite solution to the problem (1).

In the rest of this section, for convenience the triple $(x^T, y^T, z^T)^T$ is often denoted by u, $((x^*)^T, (y^*)^T, (z^*)^T)^T$ is often denoted by u^*, and so on. According to Theorem 1, it is trivial to show one of the significant properties of the two networks NN-I and NN-II.

Lemma 3: For any initial point $u_0 \in \Re^{n+m+r}$, there exists a unique continuous solution $u(t)$ for both NN-I and NN-II for $t \in [t_0, \tau)$ with $u(t_0) = u_0$.

Proof: By using Lemma 2, it is easy to prove that the right-hand-sides of both NN-I and NN-II are Lipschitz continuous in $\Re^{n+m+r}$. Then the results follow from the existence theory of ordinary differential equations [27]. ■

A. Neural Network I

Lemma 4: The following function is convex and continuously differentiable on $\Re^{n+m+r}$:

$$\phi(u) = \frac{1}{2}(\|x - A^T y + C^T z\|_2^2 - \|x - A^T y + C^T z - \bar{x}\|_2^2).$$

Proof: In view of Lemma 2 it is trivial to show that $\phi(u)$ is continuously differentiable on $\Re^{n+m+r}$ whose gradient is given by

$$\nabla\phi(u) = \begin{pmatrix} \bar{x} \\ -A\bar{x} \\ C\bar{x} \end{pmatrix}.$$

For any $u, u' \in \Re^{n+m+r}$

$$\begin{aligned} &(u - u')^T(\nabla\phi(u) - \nabla\phi(u')) \\ =&(x - x')^T(\bar{x} - \bar{x}') + (y - y')^T(-A\bar{x} + A\bar{x}') \\ &+ (z - z')^T(C\bar{x} - C\bar{x}') \\ =&(\bar{x} - \bar{x}')^T((x - A^T y + C^T z - \bar{x}) \\ &- (x' - A^T y' + C^T z' - \bar{x}')) + \|\bar{x} - \bar{x}'\|_2^2 \\ \geq&(\bar{x} - \bar{x}')^T(x - A^T y + C^T z - \bar{x}) \\ &+ (\bar{x}' - \bar{x})^T(x' - A^T y' + C^T z' - \bar{x}') \\ \geq&0 \end{aligned}$$

where $\bar{x}' = g_X(x' - A^T y' + C^T z')$ and the last inequality follows from Lemma 2. Therefore $\phi(u)$ is convex on $\Re^{n+m+r}$ [3]. ■

Theorem 3: With any initial point $u_0 \in \Re^{n+m+r}$, the neural network NN-I is stable in the sense of Lyapunov and converges to a solution of (1).

Proof: From Lemma 3, there exists a unique continuous solution $u(t)$ to the neural network NN-I for $t \in [t_0, \tau)$ with $u(t_0) = u_0$. Define the following Lyapunov function with respect to $u(t)$:

$$V_1(u(t)) = \frac{1}{2}\|u - u^*\|_2^2 + \phi(u) - \phi(u^*) - (u - u^*)^T\nabla\phi(u^*)$$

where u^* stands for a finite equilibrium point of NN-I and $\phi(u(t))$ is defined in Lemma 4. The convexity of $\phi(u)$ implies that $V_1(u) \geq \frac{1}{2}\|u - u^*\|_2^2$. By Lemma 2 we have

$$\begin{gathered} (\bar{x} - x^*)^T(x - A^T y + C^T z - \bar{x}) \geq 0 \\ (\bar{y} - y^*)^T(y + A\bar{x} - b - \bar{y}) \geq 0 \\ (\bar{z} - Cx^*)^T(C\bar{x} - z - \bar{z}) \geq 0. \end{gathered}$$

By (8) we have

$$\begin{gathered} (\bar{x} - x^*)^T(A^T y^* - C^T z^*) \geq 0 \\ (\bar{y} - y^*)^T(-Ax^* + b) \geq 0 \\ (\bar{z} - Cx^*)^T z^* \geq 0. \end{gathered}$$

Adding the two sets of equations gives

$$\begin{gathered} (\bar{x} - x^*)^T(x - A^T y + C^T z - \bar{x} + A^T y^* - C^T z^*) \geq 0 \\ (\bar{y} - y^*)^T(y + A\bar{x} - \bar{y} - Ax^*) \geq 0 \\ (\bar{z} - Cx^*)^T(C\bar{x} - z - \bar{z} + z^*) \geq 0. \end{gathered}$$

Making use of these inequalities we can calculate the time derivative of $V_1(u)/\lambda$ along the trajectory of neural network NN-I as follows:

$$
\begin{aligned}
&\frac{1}{\lambda}\frac{dV_1(u(t))}{dt} = \frac{1}{\lambda}\nabla V_1(u)^T\frac{du}{dt}\\
&= -(x-\bar{x})^T(x-2x^*+\bar{x}) - 2(y-\bar{y})^T(y-y^*\\
&\quad - A\bar{x}+Ax^*) - 2(C\bar{x}-\bar{z})^T(z-z^*+C(\bar{x}-x^*))\\
&= -\|x-\bar{x}\|_2^2 - (x-\bar{x})^T(2\bar{x}-2x^*)\\
&\quad - 2\|y-\bar{y}\|_2^2 - 2(y-\bar{y})^T(\bar{y}-y^*-A\bar{x}+Ax^*)\\
&\quad - 2\|C\bar{x}-\bar{z}\|_2^2 - 2(C\bar{x}-Cx^*)^T(z-z^*+\bar{z}-Cx^*)\\
&\quad - 2(\bar{z}-Cx^*)^T(-z+z^*-\bar{z}+Cx^*)\\
&= -\|x-\bar{x}\|_2^2 - 2\|y-\bar{y}\|_2^2 - 2\|C\bar{x}-\bar{z}\|_2^2\\
&\quad - 2(\bar{x}-x^*)^T(x-\bar{x}) - 2(\bar{y}-y^*)^T(y-\bar{y})\\
&\quad - 2(y-y^*)^T(-A\bar{x}+Ax^*) - 2(\bar{y}-y^*)^T(A\bar{x}-Ax^*)\\
&\quad - 2(\bar{x}-x^*)^T(C^Tz-C^Tz^*) - 2(\bar{z}-Cx^*)^T(C\bar{x}-Cx^*)\\
&\quad - 2(\bar{z}-Cx^*)^T(-z+z^*-\bar{z}+Cx^*)\\
&= -\|x-\bar{x}\|_2^2 - 2\|y-\bar{y}\|_2^2 - 2\|C\bar{x}-\bar{z}\|_2^2\\
&\quad - 2(\bar{x}-x^*)^T(x-\bar{x}-A^Ty+A^Ty^*+C^Tz-C^Tz^*)\\
&\quad - 2(\bar{y}-y^*)^T(y-\bar{y}+A\bar{x}-Ax^*)\\
&\quad - 2(\bar{z}-Cx^*)^T(C\bar{x}-z+z^*-\bar{z})\\
&\le -\|x-\bar{x}\|_2^2 - 2\|y-\bar{y}\|_2^2 - 2\|C\bar{x}-\bar{z}\|_2^2 \le 0.
\end{aligned}
$$

Hence, the neural network NN-I is stable in the sense of Lyapunov. Clearly, $dV_1/dt = 0$ if and only if u is an equilibrium point of neural network NN-I.

Since $V_1(u) \ge \|u-u^*\|_2^2/2$, any level set of $V_1(u)$, for example, $\mathcal{L} = \{u \in \Re^{n+m+r} | V_1(u) \le V_1(u_0)\}$ is bounded. Then $u(t) \in \mathcal{L}$ is bounded for all $t \ge t_0$. Therefore $\tau = +\infty$. It also follows that for any initial point u_0, there exists a convergent subsequence $\{u(t_k)\}$ with $t_0 < t_1 < t_2 < \cdots < t_k < t_{k+1} < \cdots$ such that

$$\lim_{k\to\infty} u(t_k) = u^\dagger$$

where $u^\dagger$ is an equilibrium point of NN-I. Finally, define another Lyapunov function

$$V_1^\dagger(u) = \frac{1}{2}\|u-u^\dagger\|_2^2 + \phi(u) - \phi(u^\dagger) - (u-u^\dagger)^T\nabla\phi(u^\dagger).$$

It is easy to see that $V_1^\dagger(u)$ decreases along the trajectory of NN-I and satisfies $V_1^\dagger(u^\dagger) = 0$. Therefore, for any $\varepsilon > 0$, there exists $q > 0$ such that, for all $t \ge t_q$

$$\|u(t)-u^\dagger\|_2^2/2 \le V_1^\dagger(u(t)) \le V_1^\dagger(u(t_q)) < \varepsilon.$$

Therefore, $\lim_{t\to\infty} u(t) = u^\dagger$. It follows that the neural network is globally convergent to an equilibrium point, and hence, a solution of problem (1). ■

B. Neural Network II

In view of Lemma 2 and similar to Lemma 4 we can prove the following results.

Lemma 5: The following function is convex and continuously differentiable on $\Re^{n+m+r}$:

$$\psi(u) = \frac{1}{2}(\|y+Ax-b\|_2^2 - \|y+Ax-b-\tilde{y}\|_2^2 + \|Cx-z-\tilde{z}\|_2^2)$$

whose gradient is given by

$$\nabla\psi(u) = \begin{pmatrix} A^T\tilde{y} + C^T(Cx-z-\tilde{z}) \\ \tilde{y} \\ -Cx+z+\tilde{z} \end{pmatrix}.$$

Theorem 4: With any initial point $u_0 \in \Re^{n+m+r}$, the neural network NN-II is stable in the sense of Lyapunov and converges to a solution of (1).

Proof: From Lemma 3, there exists a unique continuous solution $u(t)$ of NN-II for $t \in [t_0, \tau)$ with $u(t_0) = u_0$. Define the following Lyapunov function with respect to $u(t)$:

$$V_2(u(t)) = \frac{1}{2}\|u-u^*\|_2^2 + \psi(u) - \psi(u^*) - (u-u^*)^T\nabla\psi(u^*)$$

where u^* stands for a finite equilibrium point of NN-II and $\psi(u(t))$ is defined in Lemma (5). The convexity of $\psi(u)$ implies that $V_2(u) \ge \frac{1}{2}\|u-u^*\|_2^2$. By Lemma 2, we have

$$
\begin{aligned}
(\tilde{x}-x^*)^T(x-A^T\tilde{y}+C^T(z-Cx+\tilde{z})-\tilde{x}) \ge 0\\
(\tilde{y}-y^*)^T(y+Ax-b-\tilde{y}) \ge 0\\
(\tilde{z}-Cx^*)^T(Cx-z-\tilde{z}) \ge 0.
\end{aligned}
$$

By (8) we have

$$
\begin{aligned}
(\tilde{x}-x^*)^T(A^Ty^*-C^Tz^*) \ge 0\\
(\tilde{y}-y^*)^T(-Ax^*+b) \ge 0\\
(\tilde{z}-Cx^*)^Tz^* \ge 0.
\end{aligned}
$$

Adding the two sets of equations gives

$$
\begin{aligned}
(\tilde{x}-x^*)^T(x-A^T\tilde{y}+C^T(z-Cx+\tilde{z})\\
-\tilde{x}+A^Ty^*-C^Tz^*) \ge 0\\
(\tilde{y}-y^*)^T(y+Ax-\tilde{y}-Ax^*) \ge 0\\
(\tilde{z}-Cx^*)^T(Cx-z-\tilde{z}+z^*) \ge 0.
\end{aligned}
$$

Making use of these inequalities we can calculate the time derivative of $V_2(u)/\lambda$ along the trajectory of neural network NN-II as follows:

$$
\begin{aligned}
&\frac{1}{\lambda}\frac{dV_2(u(t))}{dt} = \frac{1}{\lambda}\nabla V_2(u)^T\frac{du}{dt}\\
&= -2(x-\tilde{x})^T(x-x^*+A^T\tilde{y}-A^Ty^*\\
&\quad + C^T(Cx-z-\tilde{z}+z^*)) - (y-\tilde{y})^T(y-2y^*+\tilde{y})\\
&\quad - (Cx-\tilde{z})^T(2z-2z^*-Cx+\tilde{z})\\
&= -2\|x-\tilde{x}\|_2^2 - 2(\tilde{x}-x^*)^T(-\tilde{x}+x^*-A^T\tilde{y}+A^Ty^*\\
&\quad + C^T(-Cx+z+\tilde{z}-z^*)) - 2(x-x^*)^T(\tilde{x}-x^*\\
&\quad + A^T\tilde{y}-A^Ty^*+C^T(Cx-z-\tilde{z}+z^*))\\
&\quad - \|y-\tilde{y}\|_2^2 - (y-\tilde{y})^T(2\tilde{y}-2y^*)\\
&\quad - \|Cx-\tilde{z}\|_2^2 - 2(Cx-Cx^*)^T(z-z^*+\tilde{z}-Cx)\\
&\quad - 2(\tilde{z}-Cx^*)^T(-z+z^*-\tilde{z}+Cx)\\
&= -2\|x-\tilde{x}\|_2^2 - \|y-\tilde{y}\|_2^2 - \|Cx-\tilde{z}\|_2^2\\
&\quad - 2(\tilde{x}-x^*)^T(x-\tilde{x}-A^T\tilde{y}+A^Ty^*\\
&\quad + C^T(-Cx+z+\tilde{z}-z^*))\\
&\quad - 2(\tilde{y}-y^*)^T(Ax-Ax^*+y-\tilde{y})\\
&\quad - 2(\tilde{z}-Cx^*)^T(-z+z^*-\tilde{z}+Cx)\\
&\le -2\|x-\tilde{x}\|_2^2 - \|y-\tilde{y}\|_2^2 - \|Cx-\tilde{z}\|_2^2 \le 0.
\end{aligned}
$$

The rest of the proof is similar to the proof of Theorem 3, and is omitted here for brevity. ■

V. Special Cases

In this section, we consider solving some LAD problems with different combinations of linear constraints, which have been studied from the viewpoint of RNN in the literature. The purpose is to compare the two new RNNs with existing ones for solving the same problems. Since most of the RNNs to be compared are theoretically guaranteed to be globally convergent to correct solutions, the comparison criterion is mainly the structural complexity (some preliminary comparisons for convergence rates based on numerical simulations are presented in Section VI). It will be seen that the two new RNNs are simpler than most existing ones for solving variously constrained LAD problems, even for solving the unconstrained problem. Moreover, for solving a particular problem, another network will be derived from NN-I and NN-II by changing some connections in them.

A. Equality + One-Sided Inequality + Bound Constraints

First, we are concerned with the following LAD problem:

$$
\begin{aligned}
&\min \ \|Ax-b\|_1\\
&\text{subject to} \ \ Cx=d,\ Ex\le f,\ x\in X
\end{aligned}
\tag{9}
$$

where $E\in\Re^{p\times n}$, $f\in\Re^p$ and the other notations are the same as in (1).

By noticing that $Cx=d$, $Ex\le f$ can be written as

$$
\begin{pmatrix} d \\ -\infty \end{pmatrix} \le \begin{pmatrix} C \\ E \end{pmatrix} x \le \begin{pmatrix} d \\ f \end{pmatrix}
$$

the two neural networks developed in previous sections can well solve the problem. By introducing a new state variable $s\in\Re^p$ associated with the inequality constraint $Ex\le f$, we can derive the following neural network from NN-I:

$$
\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \\ s \end{pmatrix} = -\lambda \begin{pmatrix} x-\bar{x} \\ 2(y-\bar{y}) \\ 2(C\bar{x}-\bar{z}) \\ 2(E\bar{x}-\bar{s}) \end{pmatrix}
$$

where

$$
\begin{aligned}
\bar{x} &= g_X(x-A^Ty+C^Tz+E^Ts)\\
\bar{y} &= g_Y(y+A\bar{x}-b)\\
\bar{z} &= g_{[d,d]}(C\bar{x}-z)\\
\bar{s} &= g_{(-\infty,f]}(E\bar{x}-s).
\end{aligned}
$$

In view of $g_{[d,d]}(\cdot)=d$ and $g_{(-\infty,f]}(E\bar{x}-s)=E\bar{x}-s-(-s+E\bar{x}-f)^+$, where $(\cdot)^+=g_{\Re^p_+}(\cdot)$ with $\Re^p_+$ standing for the nonnegative quadrant of $\Re^p$, by letting $s=-w$ we have

NN-a:
$$
\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = -\lambda \begin{pmatrix} x-\bar{x} \\ 2(y-\bar{y}) \\ 2(C\bar{x}-d) \\ 2(w-\bar{w}) \end{pmatrix}
$$

where

$$
\begin{aligned}
\bar{x} &= g_X(x-A^Ty+C^Tz-E^Tw)\\
\bar{y} &= g_Y(y+A\bar{x}-b)\\
\bar{w} &= (w+E\bar{x}-f)^+.
\end{aligned}
$$

Similarly, we can derive the following network from NN-II for solving (9):

NN-b:
$$
\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = -\lambda \begin{pmatrix} 2(x-\tilde{x}) \\ y-\tilde{y} \\ Cx-d \\ w-\tilde{w} \end{pmatrix}
$$

where

$$
\begin{aligned}
\tilde{x} &= g_X(x-A^T\tilde{y}+C^T(z-Cx+d)-E^T\tilde{w})\\
\tilde{y} &= g_Y(y+Ax-b)\\
\tilde{w} &= (w+Ex-f)^+.
\end{aligned}
$$

For convenience, the above two networks are, respectively, termed NN-a and NN-b hereafter. Fig. 2 depicts their architectures by block diagrams. In the figure, $\{e_{qi}\}_{p\times n}=E$, $\{f_q\}_{p\times 1}=f$, $\{d_k\}_{r\times 1}=d$ and the other parameters are the same as in Fig. 1. Obviously, like their predecessors, the two schemes share the same structural complexity. Actually, from Fig. 2(b) and (c), transforming NN-a to NN-b, or vice versa, entails just changing six connecting lines and four gain factors, though the summator connecting with the "OUT7" port [see Fig. 2(a)] is redundant for NN-a but a must for NN-b.

Interestingly, it is found that another network can be also used to solve the problem (9), whose dynamic equations are

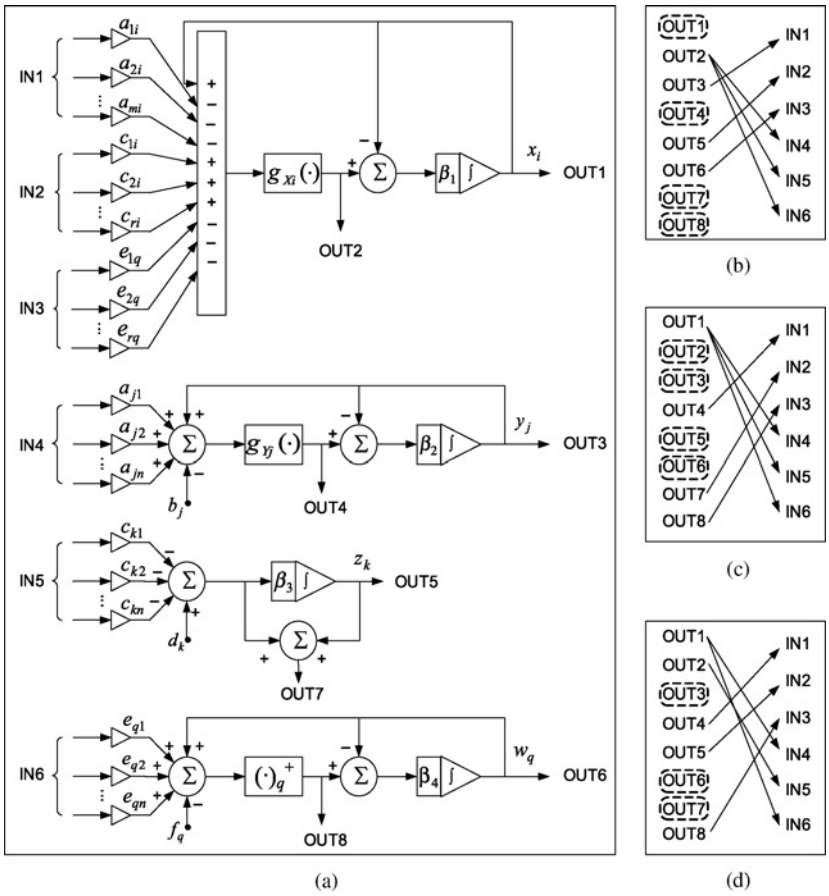


Fig. 2. Block diagrams of the neural networks NN-a, NN-b and NN-c. (a) Basic blocks with their input and output ports, where $i = 1, \cdots, n$, $j = 1, \cdots, m$, $k = 1, \cdots, r$ and $q = 1, \cdots, p$. (b)–(d) Illustrate how the blocks should be connected to constitute NN-a, NN-b and NN-c. The output ports surrounded by dashed rectangles do not need to be connected with other ports in the corresponding configuration. The gain factors in (a) for NN-a should be set as $\beta_1 = \lambda$, $\beta_2 = \beta_3 = \beta_4 = 2\lambda$, for NN-b should be set as $\beta_1 = 2\lambda$, $\beta_2 = \beta_3 = \beta_4 = \lambda$, and for NN-c should be set as $\beta_1 = \beta_3 = 2\lambda$, $\beta_2 = \beta_4 = \lambda$. Clearly, switching between these networks entails just changing some connections and gain factors.

NN-c:
$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = -\lambda \begin{pmatrix} 2(x-\hat{x}) \\ y-\tilde{y} \\ 2(C\hat{x}-d) \\ w-\tilde{w} \end{pmatrix}$$

where

$$\begin{aligned} \hat{x} &= g_X(x - A^T\tilde{y} + C^T z - E^T\tilde{w}) \\ \tilde{y} &= g_Y(y + Ax - b) \\ \tilde{w} &= (w + Ex - f)^+ \end{aligned}$$

and $\lambda > 0$. This network is called NN-c hereafter. A comparison between the dynamic equations of NN-c and NN-a shows that for accomplish one iteration of computing, the same number of mathematic operations such as summations and multiplications are needed with appropriate computing orders (actually, in NN-a $\bar{x}$ should be computed before $\bar{y}$ and $\bar{w}$ while in NN-c $\hat{x}$ should be computed after $\tilde{y}$ and $\tilde{w}$). This intuitive observation implies that circuits implementation of the two models entails the same number of elements. Fig. 2(d) illustrates the architecture of NN-c. A comparison between Fig. 2(b) and Fig. 2(d) indicates that transforming from NN-a to NN-c, or vice versa, entails just changing four connecting lines and three gain factors. Likewise, from Fig. 2(c) and Fig. 2(d) it can be seen that transforming from NN-b to NN-c, or vice versa, entails just changing two connecting lines and three gain factors. In this sense, the three networks NN-a, NN-b and NN-c can compete with each other in terms of structural complexity.

From Section IV we know that the two networks NN-a and NN-b are both globally convergent to a solution of the problem (9). We then investigate if NN-c also has this property. First, it is trivial to show that its equilibrium point set is identical to that of NN-a or NN-b, and that a point $u^* = ((x^*)^T, (y^*)^T, (z^*)^T, (w^*)^T)^T$ is an equilibrium point of these networks if and only if it satisfies

$$\begin{cases} x^* = g_X(x^* - A^T y^* + C^T z^* - E^T w^*) \\ y^* = g_Y(y^* + Ax^* - b) \\ Cx^* = d \\ w^* = (w^* + Ex^* - f)^+. \end{cases} \tag{10}$$

These equations actually describe the optimality conditions of the problem (9). Then we state the stability results of the network NN-c.

Theorem 5: With any initial point $u_0 \in \Re^{n+m+r+p}$, the neural network NN-c is stable in the sense of Lyapunov and converges to a solution of (9).

Proof: See Appendix. ■

To summarize, theoretically guaranteed performances of NN-a, NN-b and NN-c for solving (9) are the same.

From Section III-B, it can be readily concluded that the neural networks proposed in [34] and [13] may fail to solve the problem (9), but the network in (7) can guarantee the solvability theoretically. Readers can verify that solving this problem is equivalent to solving the following minimax problem: $\min_{x\in X}\max_{y\in Y,z\in\Re^r,w\in\Re^p_+} L(x,y,z,w)$ where:

$$L(x,y,z,w) = y^T(Ax-b) - z^T(Cx-d) + w^T(Ex-f).$$

With this result, one can verify that the neural network proposed in (7) reduces to NN-b exactly.

B. Bound Constraints Only

Recently, Xia and Kamel did some benchmark work in applying RNNs to parameter estimation [35], [36]. Specifically, in [35], they considered solving the following LAD problem:

$$\begin{aligned} &\min \ \|Ax-b\|_1 \\ &\text{subject to } x\in X, \end{aligned} \tag{11}$$

where the notations are the same as in (1). The following so-called *cooperative neural network* was proposed to solve the problem:

$$\frac{d}{dt}\begin{pmatrix}x\\y\end{pmatrix} = -\lambda\begin{pmatrix}x-g_X(x-A^Ty)+A^T(g_Y(y+Ax-b)-y)\\ y-g_Y(y+Ax-b)-A(g_X(x-A^Ty)-x)\end{pmatrix} \tag{12}$$

where $\lambda > 0$. As the problem (11) is a special case of the problem (9), all of the three neural networks NN-a, NN-b and NN-c can be applied to solve the problem. Specifically, the first one degenerates to

$$\frac{d}{dt}\begin{pmatrix}x\\y\end{pmatrix} = -\lambda\begin{pmatrix}x-g_X(x-A^Ty)\\ 2(y-g_Y(y+Ag_X(x-A^Ty)-b))\end{pmatrix} \tag{13}$$

and both of the other two degenerate to

$$\frac{d}{dt}\begin{pmatrix}x\\y\end{pmatrix} = -\lambda\begin{pmatrix}2(x-g_X(x-A^Tg_Y(y+Ax-b)))\\ y-g_Y(y+Ax-b)\end{pmatrix}. \tag{14}$$

One should keep in mind that g_X in (12) and (13) and (12) and (14) are not needed to be implemented twice. Even with this consideration, it is easy to see that both (13) and (14) are simpler than (12).

Another constrained LAD problem for parameter estimation formulated by Xia and Kamel is as follows [36]:

$$\begin{aligned} &\min \ \|Ax-s-b\|_1 \\ &\text{subject to } s\in\mathcal{S}, \end{aligned} \tag{15}$$

where the notations are the same as in (1) with a new variable s and a new box set $\mathcal{S}$. A neural network was proposed for solving it

$$\frac{d}{dt}\begin{pmatrix}x\\s\\y\end{pmatrix} = -\lambda\begin{pmatrix}A^Tg_Y(y+Ax-s-b)\\ s-g_S(s+y)+y-g_Y(y+Ax-s-b)\\ y+AA^Ty-g_Y(y+Ax-s-b)-s+g_S(s+y)\end{pmatrix} \tag{16}$$

where $\lambda > 0$. Again, all of the three neural networks NN-a, NN-b and NN-c can be applied to solve the problem (15). Specifically, the first one degenerates to

$$\frac{d}{dt}\begin{pmatrix}x\\s\\y\end{pmatrix} = -\lambda\begin{pmatrix}A^Ty\\ s-g_S(s+y)\\ 2(y-g_Y(y+A(x-A^Ty)-g_S(s+y)-b))\end{pmatrix} \tag{17}$$

and the other two degenerate to

$$\frac{d}{dt}\begin{pmatrix}x\\s\\y\end{pmatrix} = -\lambda\begin{pmatrix}2A^Tg_Y(y+Ax-s-b)\\ 2(s-g_S(s+g_Y(y+Ax-s-b)))\\ y-g_Y(y+Ax-s-b)\end{pmatrix}. \tag{18}$$

Clearly, both networks are simpler than (16).

C. No Constraints

Finally, we consider the unconstrained LAD problem

$$\min \ \|Ax-b\|_1. \tag{19}$$

The following network was designed in [33] for solving it:

$$\frac{d}{dt}\begin{pmatrix}x\\y\end{pmatrix} = -\lambda\begin{pmatrix}A^Tg_Y(y+Ax-b)\\ y-g_Y(y+Ax-b)+AA^Ty\end{pmatrix} \tag{20}$$

where $\lambda > 0$. Obviously, the two models (13) and (14) can solve this problem. By setting $X = \Re^n$, their dynamic equations are, respectively, simplified to be

$$\frac{d}{dt}\begin{pmatrix}x\\y\end{pmatrix} = -\lambda\begin{pmatrix}A^Ty\\ 2(y-g_Y(y+A(x-A^Ty)-b))\end{pmatrix} \tag{21}$$

and

$$\frac{d}{dt}\begin{pmatrix}x\\y\end{pmatrix} = -\lambda\begin{pmatrix}2A^Tg_Y(y+Ax-b)\\ y-g_Y(y+Ax-b)\end{pmatrix}. \tag{22}$$

Comparisons show that both of the two new models possess lower structural complexity than (20) by noticing that in (20) an additional term AA^Ty has to be calculated whereas in (21) or (22) this is unnecessary.

VI. Numerical Simulations

To illustrate the theoretical results obtained in previous sections and compare the performances of the RNNs proposed in this paper, we have simulated their dynamic systems to solve some LAD problems in MATLAB.

Experiment 1: Consider an unconstrained LAD problem (19) with

$$A = \begin{pmatrix} -1 & 1 & 0 \\ -0.5 & 1 & -.5 \\ 0.5 & -1 & 1 \\ 1 & -1 & 1 \\ 1 & -0.5 & 0.5 \\ 2 & -1 & 1 \\ 1 & -1 & 1 \\ 0.5 & -1 & 1 \\ 0.5 & -0.75 & 1 \\ 2 & -2 & 3 \\ 0 & -1 & 1 \\ 1 & -1 & 3 \end{pmatrix} \quad b = \begin{pmatrix} -1 \\ -0.5 \\ 1 \\ 1 \\ 0.5 \\ 1 \\ 0 \\ -0.5 \\ -0.25 \\ -3 \\ 0 \\ -2 \end{pmatrix}.$$

This problem is highly degenerate as there are 30 degenerate base points [30]. The optimal solution obtained by several efficient numerical algorithms is $x^* = (0.7142857, -0.7142857, -1.1428571)^T$ with the objective function value 5.78571 [30]. We simulated the two neural networks (21) and (22) to solve the problem. It was observed that from any initial point in $\Re^{15}$ the outputs always converged to this solution. Fig. 3 depicts the states of the networks with respect to the time t with $\lambda = 1000$ from a random (but identical for the two networks) initial point.

We then considered solving the same problem with an addition constraint $0 \leq 2x_1 - x_2 \leq 2$ by using NN-I and NN-II. Simulations indicated that they always converged to the solution $x^* = (0.6250, -0.7500, -1.1250)^T$ with the optimal objective function value 5.8125. Fig. 4 depicts the error function $\|x(t) - x^*\|_2$ for the two networks from four random initial points. It is seen that the value of this function decreases to zero very fast in any case.

Experiment 2: In Section III, we have shown that the proposed two neural networks NN-I and NN-II possess the same structural complexity, while in Section IV we have shown that they possess the same stability results. But their convergence rates were not compared, though all of them should be greatly faster than numerical LAD solvers and this issue is not a so important criterion for judging the validity of the networks. Anyway, for the sake of completeness, we would like to do an investigation in this regard. But it was found to be difficult to derive such results theoretically for continuous neural networks. So, we compared the convergence rates on a variety of problems, which is a strategy often adopted for comparing the speeds of numerical algorithms for solving mathematical problems. Note that the required time for accomplishing a task relies on the scaling factor λ; actually the larger the scaling factor, the less the time. Without loss of generality, in simulations λ was always set to one. Therefore, in what follows, a *time unit* denotes $1/\lambda$, where λ can be very large, e.g., 10^6.

We considered solving the LAD problem in the general form of (1) with different coefficients. Random numbers were generated uniformly between -1 and 1 and then assigned to the elements of A and C. b_i $(i = 1, \ldots, m)$ was the sum of the corresponding row of A plus a number generated from normal distribution with mean zero and standard deviation 3, denoted by $\Psi(0, 3)$. l_i $(i = 1, \ldots, r)$ (or h_i) was the sum of the corresponding row of C minus (or plus) a number generated by $\max\{0, \Psi(0, 3)\}$. $\rho_i = -1$, $\varrho_i = 1$ $(i = 1, \ldots, n)$. Six combinations of n, m, r were considered (see Table I) and in each combination 30 sets of coefficients were generated in above way.

As indicated in Section III-B, the network (7) can also solve these testing problems, though its circuits implementation is more expensive. For comparison purpose, it was simulated on the problems, too. On each problem, for NN-I and NN-II, simulations terminated when either $t \geq 1000$ time units or

$$(\|x - g_X(x - A^T y + C^T z)\|_1 + \|y - g_Y(y + Ax - b)\|_1 + \|Cx - g_\Omega(Cx - z)\|_1)/(n + m + r) \leq 10^{-4}.$$

The same stopping criteria were adopted for the network (7), but the above equation was replaced with

$$(\|x - g_X(x - A^T y + C^T z)\|_1 + \|Cx - s\|_1 + \|y - g_Y(y + Ax - b)\|_1 + \|s - g_\Omega(Cx - z)\|_1)/(n + m + 2r) \leq 10^{-4}.$$

It is not difficult to check that the left sides of the above two inequalities are equal to zero if and only if x is a solution of the problem (1). The components of the initial points for the networks were uniformly and randomly generated in $[-1, 1]$.

The amount of time units required by each run was recorded and the mean μ and standard deviation σ of this quantity over 30 runs for each problem set are shown in Table I. It was found that the three networks successfully achieved the prescribed precision 10^{-4} within 1000 time units on most problem sets, except on the fourth set (see footnotes below the table for details). That is why the running time of any network for this set of problems is much greater than for other sets in Table I.

The last two columns of Table I show the comparison results of the required time units for the networks. Paired t-tests were carried out on each problem set between the three networks. If the amounts of time units were significantly different for two networks on a problem set ($p < 0.01$), then a tailed t-test was carried out to estimate which is smaller. It was found that the running time of NN-I was less than that of NN-II and (7) on most problem sets ($p < 10^{-4}$) except on the first set where no significant difference was found ($p > 0.76$). In addition, the running time of the network (7) was greater than that of NN-I but less than that of NN-II on four sets. This is not strange by considering that the network computation is a parallel process, and it is possible that a more complex architecture solves a problem faster. From Table I, it can be seen that when $n > m$, the difference between the three networks was negligible; and when $n \leq m$, NN-I was faster than the other two. Moreover, when the number of constraints

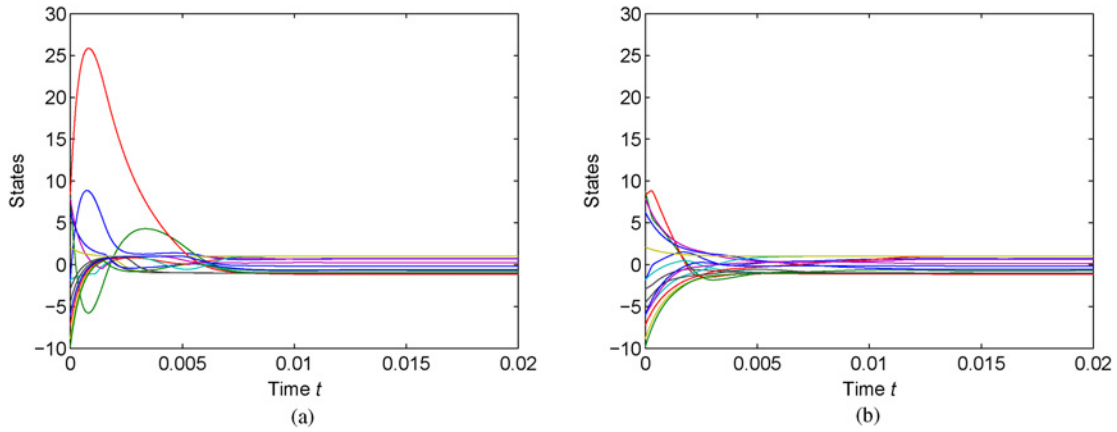


Fig. 3. States of the neural networks (21) and (22) with respect to the time t in solving the unconstrained LAD problem in Experiment 1. λ was set to 1000 for both networks. It is seen that the states of any network achieve steady values as t increases. (a) Neural Network (21). (b) Neural Network (22).

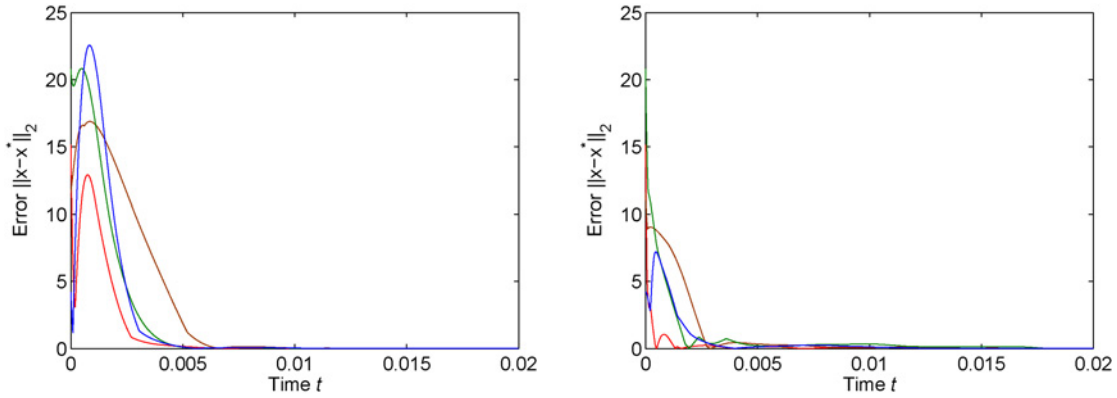


Fig. 4. Error function for the neural networks NN-I and NN-II with respect to the time t starting from four random initial points in solving the constrained LAD problem in Experiment 1 (same color in the two insets corresponds to the same initial point. Please refer to the online version of this paper for colors). λ was set to 1000 for both networks. It is seen that this function for any network decreases to zero rapidly with respect to t.

TABLE I

COMPARISONS BETWEEN NN-I, NN-II, AND NETWORK (7)

Problem Set	NN-I		NN-II		(7)		Conclusion	p-Value
	μ	σ	μ	σ	μ	σ		
$n = 100, m = 20, r = 20$	24.18	33.56	23.66	31.96	21.53	29.31	–	> 0.76
$n = 500, m = 20, r = 20$	9.42	0.13	10.25	0.28	9.71	0.09	NN-I< (7) <NN-II	$< 10^{-10}$
$n = 1000, m = 20, r = 20$	9.20	0.11	10.22	0.23	9.77	0.12	NN-I< (7) <NN-II	$< 10^{-9}$
$n = 100, m = 100, r = 100$[1]	539.0	377.8	773.5	346.6	753.2	302.1	NN-I<NN-II	6.1×10^{-5}
$n = 100, m = 500, r = 500$	38.94	12.31	86.74	31.32	67.04	22.15	NN-I< (7) <NN-II	< 0.004
$n = 100, m = 1000, r = 1000$	30.86	7.84	80.28	19.80	52.78	11.77	NN-I< (7) <NN-II	$< 10^{-8}$

TABLE II

COMPARISONS BETWEEN NN-A, NN-B, AND NN-C

Problem set	NN-a		NN-b		NN-c		Conclusion	p-value
	μ	σ	μ	σ	μ	σ		
$n = 100, m = 20, r = 20, p = 20$	9.88	0.11	9.54	0.10	9.40	0.12	NN-c<NN-b<NN-a	$< 10^{-8}$
$n = 500, m = 20, r = 20, p = 20$	9.57	0.08	9.76	0.08	9.52	0.10	NN-a<NN-b, NN-c<NN-b	$< 10^{-9}$
$n = 1000, m = 20, r = 20, p = 20$	9.36	0.09	9.80	0.07	9.54	0.07	NN-a<NN-c<NN-b	$< 10^{-8}$
$n = 500, m = 20, r = 100, p = 100$	10.24	0.07	10.24	0.04	9.92	0.06	NN-c<NN-a, NN-c<NN-b	$< 10^{-17}$
$n = 1000, m = 20, r = 200, p = 200$	10.46	0.04	10.55	0.03	10.19	0.04	NN-c<NN-a<NN-b	$< 10^{-9}$
$n = 100, m = 100, r = 20, p = 20$	115.77	58.83	134.22	68.87	123.15	51.31	–	> 0.03
$n = 100, m = 500, r = 20, p = 20$	85.58	25.57	147.24	45.57	206.70	100.11	NN-a<NN-b<NN-c	$< 10^{-3}$
$n = 100, m = 1000, r = 20, p = 20$	92.05	20.22	175.38	40.71	229.44	92.02	NN-a<NN-b<NN-c	< 0.003
$n = 100, m = 100, r = 20, p = 80$[2]	158.37	95.89	202.26	122.00	215.44	140.47	NN-a<NN-b, NN-a<NN-c	$< 10^{-3}$
$n = 100, m = 500, r = 20, p = 80$[3]	191.47	111.68	304.42	138.21	365.71	142.43	NN-a<NN-b<NN-c	< 0.002

are relatively large with respect to the number of variables, all networks require large amounts of time units.

Experiment 3: We were also interested in the rates of the neural networks NN-a, NN-b, and NN-c discussed in Section V-A for solving the LAD problems in the form of (9). Thus, these dynamic systems were simulated on computers to solve a variety of randomly generated problems. Table II lists the statistics of the recorded time units required by the networks. The coefficients of the problem (9) were generated as follows. The elements of A, C and E were uniformly generated between -1 and 1. The element b_i $(i = 1, \ldots, m)$ was the sum of the corresponding row of A plus a number generated from normal distribution with mean zero and standard deviation 3, denoted by $\Psi(0, 3)$. The element d_i $(i = 1, \ldots, r)$ was the sum of the corresponding row of C plus a number generated in $\Psi(0, 3)$, and the element d_i $(i = 1, \ldots, p)$ was the sum of the corresponding row of E minus a number generated in $\Psi(0, 3)$. $\rho_i = 0, \varrho_i = +\infty$ $(i = 1, \ldots, n)$. Ten combinations of n, m, r, p were considered and in each combination 30 sets of coefficients were generated. For each problem, the initial point was generated in $[-1, 1]^{n+m+r+p}$ and the same for NN-a, NN-b and NN-c. Simulations terminated when either $t \geq 500$ time units or

$$\begin{aligned}&(\|x - g_X(x - A^T y + C^T z - E^T w)\|_1 + \|Cx - d\|_1 \\ &+ \|y - g_Y(y + Ax - b)\|_1 + \|w - (w + Ex - f)^+\|_1) \\ &/(n + m + r + p) \leq 10^{-4}.\end{aligned}$$

As in Experiment 2, the scaling factor λ was set to one.

The amount of time units required by each run was recorded and the mean μ and standard deviation σ of this quantity over 30 runs for each problem set are shown in Table II. The three networks successfully achieved the prescribed precision 10^{-4} within 500 time units on most problem sets, except on the last two sets (see footnotes below the table for details).

The results of paired t-tests between the three networks on each problem set are shown in the last two columns of Table II. It is seen that when $n \geq m$ and the number of constraints is relatively small with respect to the number of variables, the amounts of time units for the three networks are all small and the differences between them are negligible; when $n \leq m$ and the number of constraints is relatively large with respect to the number of variables, the amounts of time units for the networks are all large and NN-a is faster than the other two.

VII. Conclusion

We have designed a set of RNNs for solving LAD problems (also called L_1 minimization problems) with various combinations of linear constraints. If two-sided inequality constraints are present, two neural networks with the same structural complexity, called NN-I and NN-II, respectively, can both solve the problem. If the inequality constraints are one-sided, two networks as special cases of NN-I and NN-II can solve the problem, which are called NN-a and NN-b, respectively. In addition, for solving this particular problem, another network sharing the same structural complexity with NN-a and NN-b, called NN-c, was also found to be competent. The three networks NN-a, NN-b and NN-c can be tailored to new models for solving various special LAD problems, and these networks are often simpler than the counterparts in the literature in terms of structural complexity. Moreover, all of the networks designed in this paper can solve degenerate LAD problems naturally without resort to additional efforts, which is in contrast to numerical algorithms.

The design of several models for solving same problem takes advantage of the fact that keeping blocks of an RNN intact while changing connections between them can lead to new networks with similar structural complexity. This simple idea has inspired two new RNNs for solving linear or quadratic programming problems recently [20], [21]. Here another successful application is witnessed.

Though all of the RNNs designed in this paper were rigorously shown to be capable of solving the LAD problems, the convergence rates were not analyzed. Some preliminary numerical simulations predicted that in general NN-I was faster than NN-II, while NN-a was faster than NN-b and NN-c. Anyway, it is believed that all of such hardware solvers could run extremely faster than conventional numerical algorithms that are executed on digital computing equipments.

Appendix

Proof of Theorem 5: Similar to Lemma 3, it is easy to prove that for any initial point u_0, there exists a unique continuous solution $u(t) = (x(t)^T, y(t)^T, z(t)^T, w(t)^T)^T$ of NN-c for $t \in [t_0, \tau)$ with $u(t_0) = u_0$. Similar to Lemma 4 it can be proved that the following function is convex and continuously differentiable on $\Re^{n+m+r+p}$:

$$\begin{aligned}\eta(u) = \frac{1}{2}(&\|y + Ax - b\|_2^2 - \|y + Ax - b - \tilde{y}\|_2^2 \\ &+ \|w + Ex - f\|_2^2 - \|w + Ex - f - \tilde{w}\|_2^2)\end{aligned}$$

whose gradient is given by

$$\nabla\eta(u) = \begin{pmatrix} A^T\tilde{y} + E^T\tilde{w} \\ \tilde{y} \\ 0 \\ \tilde{w} \end{pmatrix}.$$

Define a Lyapunov function with respect to $u(t)$

$$V_3(u(t)) = \frac{1}{2}\|u - u^*\|_2^2 + \eta(u) - \eta(u^*) - (u - u^*)^T\nabla\eta(u^*)$$

where u^* stands for a finite equilibrium point of NN-c. The convexity of $\eta(u)$ implies that $V_3(u) \geq \frac{1}{2}\|u - u^*\|_2^2$. Based on the equivalence between the projection equations in (10) and variational inequalities [23], we can have

$$\begin{aligned}(\hat{x} - x^*)^T(E^T w^* + A^T y^* - C^T z^*) \geq 0 \\ (\tilde{y} - y^*)^T(-Ax^* + b) \geq 0 \\ (\tilde{w} - w^*)^T(-Ex^* + f) \geq 0.\end{aligned}$$

By Lemma 2, we have

$$(\hat{x} - x^*)^T(x - A^T\tilde{y} + C^Tz - E^T\tilde{w} - \hat{x}) \geq 0$$
$$(\tilde{y} - y^*)^T(y + Ax - b - \tilde{y}) \geq 0$$
$$(\tilde{w} - w^*)^T(w + Ex - f - \tilde{w}) \geq 0.$$

Adding these equations gives

$$(\hat{x} - x^*)^T(x - A^T\tilde{y} + C^Tz - E^T\tilde{w} - \hat{x} + E^Tw^* + A^Ty^* - C^Tz^*) \geq 0$$
$$(\tilde{y} - y^*)^T(y + Ax - \tilde{y} - Ax^*) \geq 0$$
$$(\tilde{w} - w^*)^T(w + Ex - \tilde{w} - Ex^*) \geq 0.$$

With similar algebraic manipulations as in the proofs of Theorems 3 and 4, it can be reasoned that

$$\begin{aligned}
&\frac{1}{\lambda}\frac{dV_3(u(t))}{dt} \\
&= -2(x - \hat{x})^T(x - x^* + A^T\tilde{y} - A^Ty^* + E^T\tilde{w} - E^Tw^*) \\
&\quad - (y - \tilde{y})^T(y - 2y^* + \tilde{y}) - 2(C\hat{x} - d)^T(z - z^*) \\
&\quad - (w - \tilde{w})^T(w - 2w^* + \tilde{w}) \\
&= -2\|x - \hat{x}\|_2^2 - \|y - \tilde{y}\|_2^2 - \|w - \tilde{w}\|_2^2 \\
&\quad - 2(\hat{x} - x^*)^T(x - \hat{x} - A^T\tilde{y} + A^Ty^* - E^T\tilde{w} + E^Tw^* \\
&\quad + C^Tz - C^Tz^*) - 2(\tilde{y} - y^*)^T(y - \tilde{y} + Ax - Ax^*) \\
&\quad - 2(\tilde{w} - w^*)^T(w - \tilde{w} + Ex - Ex^*) \\
&\leq -2\|x - \hat{x}\|_2^2 - \|y - \tilde{y}\|_2^2 - \|w - \tilde{w}\|_2^2 \leq 0.
\end{aligned}$$

Therefore, the network NN-c is stable in the sense of Lyapunov. Since $V_3(u) \geq \|u - u^*\|_2^2/2$, any level set of $V_3(u)$, for example, $\mathcal{L} = \{u \in \Re^{n+m+r} | V_3(u) \leq V_3(u_0)\}$ is bounded. Then, $u(t) \in \mathcal{L}$ is bounded for all $t \geq t_0$. Therefore, $\tau = +\infty$.

According to the LaSalle invariance principle, $u(t)$ converges to the largest invariant set $\mathcal{M}$ in $\{u \in \Re^{n+m+r+p} | dV_3(u)/dt = 0\}$. In what follows, we show $\mathcal{M} = U^*$, where U^* denotes the equilibrium point set of the network. Clearly, any point in U^* also belongs to $\mathcal{M}$. Consider any point $u \in \mathcal{M}$. Since $dV_3/dt = 0$, then $x = \hat{x}$, $y = \tilde{y}$ and $w = \tilde{w}$ from above analysis, which implies $dx/dt = -2\lambda(x - \hat{x}) = 0, dy/dt = -\lambda(y - \tilde{y}) = 0$ and $dw/dt = -\lambda(w - \tilde{w}) = 0$. It follows that x is in the steady state (a constant), so is $\hat{x}$. Denote $C\hat{x} - d$ by c where c is a constant. If $c \neq 0$, then $dz/dt = -2\lambda c$ and $z \to \infty$ when $t \to +\infty$, which contradicts the boundedness of $u(t)$. Consequently, $c = 0$ and $dz/dt = 0$. It follows that $u \in U^*$. Hence, $\mathcal{M} = U^*$.

The rest of the proof is similar to the proof of Theorem 3, and is omitted here.

REFERENCES

[1] M. P. Barbarosou and N. G. Maratos, "A nonfeasible gradient projection recurrent neural network for equality-constrained optimization problems," *IEEE Trans. Neural Netw.*, vol. 19, no. 10, pp. 1665–1677, Oct. 2008.

[2] P. Bloomfield and W. L. Steiger, *Least Absolute Deviations: Theory, Applications, and Algorithms*. Boston, MA: Brikhäuser, 1983.

[3] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge University Press, 2004.

[4] L. Cheng, Z. G. Hou, and M. Tan, "A neutral-type delayed projection neural network for solving nonlinear variational inequalities," *IEEE Trans. Circuits Syst. II*, vol. 55, no. 8, pp. 806–810, Aug. 2008.

[5] A. Cichocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*. New York: Wiley, 1993.

[6] G. Costantini, R. Perfetti, and M. Todisco, "Quasi-lagrangian neural network for convex quadratic optimization," *IEEE Trans. Neural Netw.*, vol. 19, no. 10, pp. 1804–1809, Oct. 2008.

[7] Y. Dodge and J. Jana, *Adaptive Regression*. New York: Springer, 2000.

[8] M. Forti, P. Nistri, and M. Quincampoix, "Generalized neural network for nonsmooth nonlinear programming problems," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 9, pp. 1741–1754, Sep. 2004.

[9] A. L. Gall and V. Zissimopoulos, "Extended Hopfield models for combinatorial optimization," *IEEE Trans. Neural Netw.*, vol. 10, no. 1, pp. 72–80, Jan. 1999.

[10] X. Gao, "A novel neural network for nonlinear convex programming," *IEEE Trans. Neural Netw.*, vol. 15, no. 3, pp. 613–621, May 2004.

[11] X. Gao and L. Liao, "A novel neural network for a class of convex quadratic minimax problems," *Neural Comput.*, vol. 18, no. 8, pp. 1818–1846, Aug. 2006.

[12] X. Gao, L. Liao, and L. Qi, "A novel neural network for variational inequalities with linear and nonlinear constraints," *IEEE Trans. Neural Netw.*, vol. 16, no. 6, pp. 1305–1317, Nov. 2005.

[13] X. Gao, L. Liao, and W. Xue, "A neural network for a class of convex quadratic minimax problems with constraints," *IEEE Trans. Neural Netw.*, vol. 15, no. 3, pp. 622–628, May 2004.

[14] S. Haykin, *Adaptive Filter Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[15] J. J. Hopfield and D. W. Tank, "Computing with neural circuits: A model," *Science*, vol. 233, no. 4764, pp. 625–633, Aug. 1986.

[16] Z. Hou, M. Gupta, P. Nikiforuk, M. Tan, and L. Cheng, "A recurrent neural network for hierarchical control of interconnected dynamic systems," *IEEE Trans. Neural Netw.*, vol. 18, no. 2, pp. 466–481, Mar. 2007.

[17] X. Hu and J. Wang, "Solving pseudomonotone variational inequalities and pseudoconvex optimization problems using the projection neural network," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1487–1499, Nov. 2006.

[18] X. Hu and J. Wang, "Solving generally constrained generalized linear variational inequalities using the general projection neural networks," *IEEE Trans. Neural Netw.*, vol. 18, no. 6, pp. 1697–1708, Nov. 2007.

[19] X. Hu and J. Wang, "An improved dual neural network for solving a class of quadratic programming problems and its k-winners-take-all application," *IEEE Trans. Neural Netw.*, vol. 19, no. 12, pp. 2022–2031, Dec. 2008.

[20] X. Hu and B. Zhang, "A new recurrent neural network for solving convex quadratic programming problems with an application to the k-winners-take-all problem," *IEEE Trans. Neural Netw.*, vol. 20, no. 4, pp. 654–664, Apr. 2009.

[21] X. Hu and B. Zhang, "Another simple recurrent neural network for quadratic and linear programming," in *Proc. 6th Int. Symp. Neural Netw.*, May 2009, pp. 116–125.

[22] X. Hu and B. Zhang, "An alternative recurrent neural network for solving variational inequalities and related optimization problems," *IEEE Trans. Syst., Man, Cybern. B*, vol. 39, no. 6, pp. 1640–1645, Dec. 2009.

[23] D. Kinderlehrer and G. Stampcchia, *An Introduction to Variational Inequalities and Their Applications*. New York: Academic, 1980.

[24] W. Li and J. J. Swetits, "The linear l_1 estimator and the Huber M-estimator," *SIAM J. Optimization*, vol. 8, no. 2, pp. 457–475, 1998.

[25] Q. Liu and J. Wang, "A one-layer recurrent neural network with a discontinuous hard-limiting activation function for quadratic programming," *IEEE Trans. Neural Netw.*, vol. 19, no. 4, pp. 558–570, Apr. 2008.

[26] J. Malick, "A dual approach to semidefinite least-squares problems," *SIAM J. Matrix Anal. Appl.*, vol. 26, no. 1, pp. 272–284, 2004.

[27] R. K. Miller and A. N. Michel, *Ordinary Differential Equations*. New York: Academic, 1982.

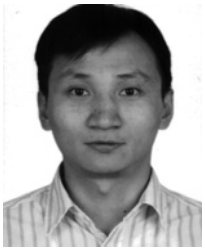
[28] M. Ohlasson, C. Pelterson, and B. Soderberg, "Neural networks for optimization problems with inequality constraints: The knapsack problem," *Neural Comput.*, vol. 5, no. 2, pp. 331–339, 1993.

[29] S. A. Ruzinsky and E. T. Olsen, "L_1 and L_∞ minimization via a variant of Karmarkar's algorithm," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 37, no. 2, pp. 245–253, Feb. 1989.

[30] M. Shi and M. A. Lukas, "An L_1 estimation algorithm with degeneracy and linear constraints," *Comput. Statis.& Data Anal.*, vol. 39, no. 1, pp. 35–55, 2002.

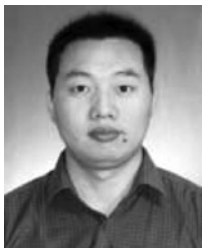
[31] D. W. Tank and J. J. Hopfield, "Simple neural optimization networks: An A/D converter, signal decision circuit, and a linear programming

circuit," *IEEE Trans. Circuits Syst.*, vol. 33, no. 5, pp. 533–541, May 1986.
[32] E. J. Teoh, K. C. Tan, H. J. Tang, C. Xiang, and C. Goh, "An asynchronous recurrent linear threshold network approach to solving the traveling salesman problem," *Neurocomputing*, vol. 71, nos. 7–9, pp. 1359–1372, Mar. 2008.
[33] Z. Wang, Z. He, and J. Chen, "Robust time delay estimation of bioelectric signals," *IEEE Trans. Biomed. Eng.*, vol. 52, no. 3, pp. 454–42, Mar. 2005.
[34] Z. Wang and B. S. Peterson, "Constrained least absolute deviation neural networks," *IEEE Trans. Neural Netw.*, vol. 19, no. 2, pp. 273–283, Feb. 2008.
[35] Y. Xia and M. S. Kamel, "Cooperative recurrent neural networks for the constrained L_1 estimator," *IEEE Trans. Signal Process.*, vol. 55, no. 7, pp. 3192–3206, Jul. 2007.
[36] Y. Xia and M. S. Kamel, "A generalized least absolute deviation method for parameter estimation of autoregressive signals," *IEEE Trans. Neural Netw.*, vol. 19, no. 1, pp. 107–118, Jan. 2008.
[37] Y. Xia and J. Wang, "A recurrent neural network for nonlinear convex optimization subject to nonlinear inequality constraints," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 7, pp. 1385–1394, Jul. 2004.
[38] Y. Yang and J. Cao, "Solving quadratic programming problems by delayed projection neural network," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1630–1634, Nov. 2006.
[39] Y. Zhang and S. S. Ge, "Design and analysis of a general recurrent neural network model for time-varying matrix inversion," *IEEE Trans. Neural Netw.*, vol. 16, no. 6, pp. 1477–1490, Nov. 2005.



Xiaolin Hu (S'01–M'09) received the B.E. and M.E. degrees in automotive engineering from the Wuhan University of Technology, Wuhan, China, and the Ph.D. degree in automation and computer-aided engineering from the Chinese University of Hong Kong, New Territories, Hong Kong, in 2001, 2004, 2007, respectively.

He is currently an Assistant Professor with the State Key Laboratory of Intelligent Technology and Systems, the Tsinghua National Laboratory for Information Science and Technology, and the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His current research interests include artificial intelligence and computational neuroscience.



Changyin Sun (M'04) received the Ph.D. degree in electrical engineering from Southeast University, Nanjing, China, in 2004.

From 2007 to 2009, he was the Program Director with the Department of Information Sciences, National Natural Science Foundation of China. Currently, he is a Professor with the School of Automation, Southeast University. He has published more than 50 papers. His current research interests include neural networks, pattern recognition, and intelligent control theory.

Dr. Sun has received the First Prize of Natural Science Award from the Ministry of Education of China. He is on the editorial boards of several international journals, including IEEE TRANSACTIONS ON NEURAL NETWORKS, *Neural Processing Letters*, and *International Journal of Swarm Intelligence Research*.

Bo Zhang graduated from the Department of Automatic Control, Tsinghua University, Beijing, China, in 1958.

He is currently a Professor with the State Key Laboratory of Intelligent Technology and Systems, the Tsinghua National Laboratory for Information Science and Technology, the Department of Computer Science and Technology, Tsinghua University, and a Fellow of Chinese Academy of Sciences, Beijing, China. He has published about 150 papers and three monographs in these fields. His current research interests include artificial intelligence, robotics, intelligent control, and pattern recognition.