A New Recurrent Neural Network for Solving Convex Quadratic Programming Problems With an Application to the k-Winners-Take-All Problem

Xiaolin Hu and Bo Zhang

***Abstract*—In this paper, a new recurrent neural network is proposed for solving convex quadratic programming (QP) problems. Compared with existing neural networks, the proposed one features global convergence property under weak conditions, low structural complexity, and no calculation of matrix inverse. It serves as a competitive alternative in the neural network family for solving linear or quadratic programming problems. In addition, it is found that by some variable substitution, the proposed network turns out to be an existing model for solving minimax problems. In this sense, it can be also viewed as a special case of the minimax neural network. Based on this scheme, a k-winners-take-all (k-WTA) network with $O(n)$ complexity is designed, which is characterized by simple structure, global convergence, and capability to deal with some ill cases. Numerical simulations are provided to validate the theoretical results obtained. More importantly, the network design method proposed in this paper has great potential to inspire other competitive inventions along the same line.**

***Index Terms*—Asymptotic stability, k-winners-take-all (k-WTA), linear programming, neural network, quadratic programming.**

I. Introduction

In this paper, we consider solving the following convex quadratic programming (QP) problem:

$$\begin{aligned} \min \quad & \frac{1}{2}x^TQx + p^Tx \\ \text{subject to} \quad & Ax \le b, \qquad Cx = d, \qquad x \in X \end{aligned} \tag{1}$$

where $Q = Q^T \in \Re^{n\times n}, p \in \Re^n, A \in \Re^{m\times n}, b \in \Re^m, C \in \Re^{r\times n}, d \in \Re^r$ are given with Q being positive semidefinite, and X is a nonempty box set defined as $X = \{x \in \Re^n \,|\, l_i \le x \le h_i, i = 1, \ldots, n\}$. Note that some l_i's can be $-\infty$ and some h_i's can be $+\infty$. In particular, if $Q = 0$, the problem degenerates to a linear programming (LP) problem.

Problem (1) is a typical as well as basic problem in the optimization context and there exit many efficient numerical algorithms for solving it [1]. However, in many engineering applications such as signal processing, system identification, and robot motion control, real-time solutions are often desired. These problems may be high in dimension and dense in structure. Conventional numerical methods, however, may not be efficient anymore in such occasions due to stringent requirement on computing time. A promising approach to handle these problems is to employ artificial neural-network-based circuit implementation. Because of their dynamic nature in the process of searching for optima, neural networks can be implemented physically by designated hardware such as application-specific integrated circuits, where the optimization procedure is truly done in a parallel manner.

Perhaps the concept of doing optimization by using analog circuits was coined by Pyne half a century ago [2] but had not gained much popularity until the mid 1980s when Hopfield and Tank published some milestone articles [3], [4]. Now, this concept has attracted tremendous interests from different disciplines and has enjoyed a prosperous progress (see, e.g., [5]–[34] and references therein). As for solving LP or QP problems, early development in neural networks refers to [5], [6], and [7] among many others. But these models have drawbacks either in the one-to-one correspondence between the network equilibria and the solutions of the problems or in the convergence property. In [8], Wu *et al.* considered solving a quadratic (or linear) programming problem in the following form:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}x^TQx + p^Tx \\ \text{subject to} \quad & Cx = d, \qquad x \ge 0 \end{aligned} \tag{2}$$

where the notations are the same as in (1). They proposed a neural network whose equilibria are exactly solutions of the problem and there is no need to tune parameters [8]. In addition, this network is globally convergent to its equilibria. The disadvantage lies in its relatively complicated structure. For solving the same problem, a simpler scheme that retains all of its merits was proposed in [9]. The dynamics of the improved scheme can be described by

$$\frac{d}{dt}\begin{pmatrix} x \\ z \end{pmatrix} = -\begin{pmatrix} (I+Q)(x-(x-Qx-p+C^Tz)^+)+C^T(Cx-d) \\ C(x-Qx-p+C^Tz)^+-d \end{pmatrix} \tag{3}$$

Manuscript received November 05, 2007; revised August 28, 2008; accepted December 02, 2008. First published February 18, 2009; current version published April 03, 2009. The work was supported by the National Natural Science Foundation of China under Grants 60805023, 60621062, and 60605003; by the National Key Foundation R&D Project under Grants 2003CB317007, 2004CB318108, and 2007CB311003; by the China Postdoctoral Science Foundation under Grants 20080430032 and 200801072; and by the Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList).

The authors are with the State Key Laboratory of Intelligent Technology and Systems, Tsinghua National Laboratory for Information Science and Technology (TNList), and the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: xiaolin.hu@gmail.com; dcszb@mail.tsinghua.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.2008.2011266

where $(x)^+ = (x_1^+, x_2^+, \ldots, x_n^+)^T$ and $x_i^+ = \max(x_i, 0)$. In 2006, this network was further simplified to [10]

$$\frac{d}{dt}\begin{pmatrix} x \\ z \end{pmatrix} = -\begin{pmatrix} (I+Q)(x-(x-Qx-p+C^Tz)^+) \\ C(x-Qx-p+C^Tz)^+ - d \end{pmatrix}. \quad (4)$$

In fact, an even simpler neural network had already been presented in [11] before that

$$\frac{d}{dt}\begin{pmatrix} x \\ z \end{pmatrix} = -\begin{pmatrix} x - \mathcal{P}_X(x-Qx-p+C^Tz) \\ C\mathcal{P}_X(x-Qx-p+C^Tz) - d \end{pmatrix} \quad (5)$$

where X is a box set defined as in (1) and $\mathcal{P}_X(x)$ is a projection operator defined as $\mathcal{P}_X(x) = (\mathcal{P}_{X_1}(x_1), \ldots, \mathcal{P}_{X_n}(x_n))^T$ with

$$\mathcal{P}_{X_i}(x_i) = \begin{cases} l_i, & x_i < l_i \\ x_i, & l_i \le x_i \le h_i \\ h_i, & x_i > h_i. \end{cases} \quad (6)$$

Clearly, the operator $(\cdot)^+$ is a special case of $\mathcal{P}_X(\cdot)$ where X is set to $\Re_+^n$, the nonnegative quadrant of $\Re^n$. In other words, neural network (5) can solve a more general problem than (2) [$x \ge 0$ in (2) can be replaced by $l \le x \le h$]. Another idea to solve (2) is to use the time-delayed projection neural network [12]. But this network requires calculation of matrix inverse, which makes it not very much suitable for real-time applications because in such scenarios this calculation cannot be performed beforehand. Clearly, the aforementioned neural networks in [9]–[12] can be used to solve problem (1) as well, but inevitably, slack variables are needed to transform the inequality constraints to equality constraints first which will enlarge the scale of the networks. A suitable choice for resolving this difficulty is to adopt the following neural network invented in [13]:

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = -\lambda \begin{pmatrix} 2(x-\mathcal{P}_X(x-Qx-p-A^T\tilde{y}+C^T\tilde{z})) \\ y-(y+Ax-b)^+ \\ Cx-d \end{pmatrix} \quad (7)$$

where $\tilde{y} = (y+Ax-b)^+$, $\tilde{z} = z - Cx + d$, and $\lambda > 0$ is a constant scaling factor. Other candidates may refer to [14] and [15]. However, it is easy to see that the model in [14] entails much more interconnections than (7), and the model in [15] suffers from calculating multiplications and inverses of some parameter matrices, which makes it not very much suitable for online optimization. In fact, the primary advantage of the model in [15] lies in its fewer states and integrators for solving small scale optimization problems.

If Q is positive definite, several other salient networks are available for solving (1), e.g., [18]–[20], and [32]. The networks in [18] and [19] both suffer from the same difficulties as the networks in [12] and [15], while the network developed in [32], [20] does not, which is called *extended projection neural network*

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = -\lambda \begin{pmatrix} x-\mathcal{P}_X(x-Qx-p-A^Ty+C^Tz) \\ y-(y+Ax-b)^+ \\ Cx-d \end{pmatrix} \quad (8)$$

where $\lambda > 0$ is a scaling factor.

A comparison of (7) and (8) leads to the motivation of this paper. In the equilibrium state, the right-hand side of (8) is equal to zero. Then, we have $y = (y+Ax-b)^+$ and $z = z - Cx + d$. Substituting them into the first line on the right-hand side yields a neural network very much like (7). The only difference is that there is a scaling parameter 2 in (7). Such a substitution offers an advantage over the original neural network in dealing with problem (1) with Q being positive semidefinite only. If we look at the neural network (8) from a different angle, then some interesting questions will arise: Is it useful to substitute $x = \mathcal{P}_X(x - Qx - p - A^Ty + C^Tz)$ to the second and third lines of the right-hand side of (8)? Does the resulting neural network have a merit similar to (7), that is, have the ability to solve problem (1) with Q being positive semidefinite only? One of the primary aims of this paper is to answer the two questions. Note that if the inequality constraints are absent in (1), both answers are positive—see [11] and refer to (5). This encouraging fact seems to support our conjecture. Actually, in Section II, we will show that when the inequality constraints are present, the answers are also positive. The second aim of this paper is to design a simple neural network based on the above idea for solving the k-winners-take-all (k-WTA) problem, a basic problem with many applications. This is the theme of Section III. Section V summarizes the contents of this paper and discusses several future directions.

Throughout this paper, if a is a vector, then $\|a\| = \sqrt{\sum a_i^2}$. $\Re_+^n$ denotes the nonnegative quadrant of $\Re^n$. Moreover, it is always assumed that the feasible region of problem (1) is nonempty, which ensures that the solution set of the problem is nonempty.

II. A New Neural Network Model

A. Model Description

The following recurrent neural network is proposed in this paper for solving (1):

- state equation

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = -\lambda \begin{pmatrix} x - \tilde{x} \\ 2(y-(y+A\tilde{x}-b)^+) \\ 2(C\tilde{x}-d) \end{pmatrix}; \quad (9a)$$

- output equation

$$\tilde{x} = \mathcal{P}_X(x - Qx - p - A^Ty + C^Tz); \quad (9b)$$

where λ is any positive constant. The output of the network can be regarded as x as well. This is because, from Section II-C, it will be seen that the network is globally convergent to its equilibrium set, and in the equilibrium state, x is equal to $\tilde{x}$.

The block diagram of the proposed neural network is plotted in Fig. 1. In the figure, $Q = \{q_{ij}\}_{n\times n}, A = \{a_{ij}\}_{m\times n}, C = \{c_{ij}\}_{r\times n}, p = \{p_i\}_{n\times 1}, b = \{b_i\}_{m\times 1}, d = \{d_i\}_{r\times 1}$. Similar to neural networks (3), (4), (5), (7), and (8), analogy circuit implementation of the proposed neural network will entail some electronic integrators (for realizing integration/derivation), amplifiers (for realizing activation functions), multipliers, adders,

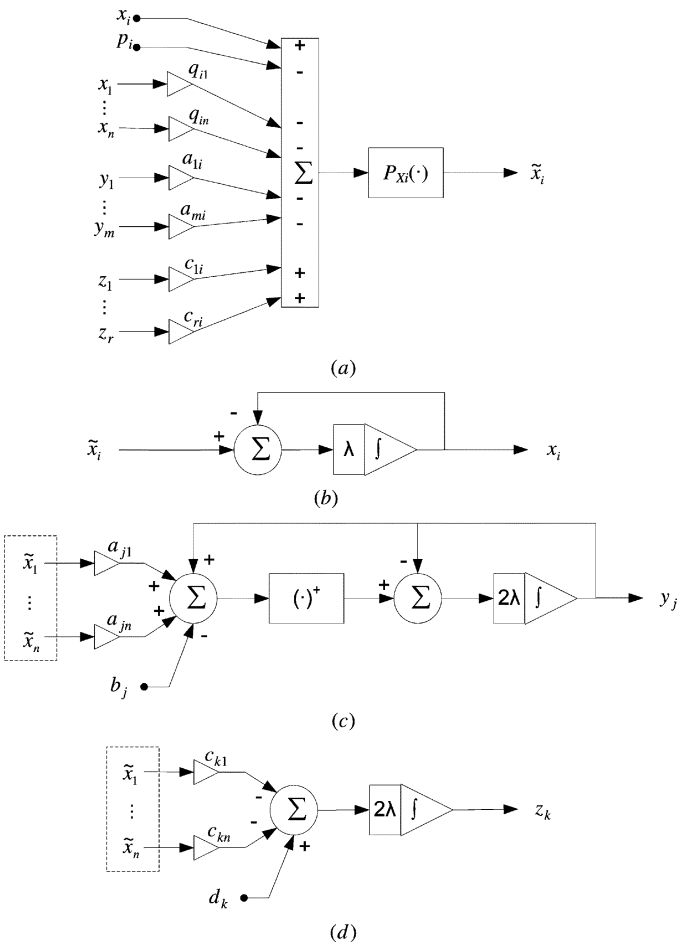

Fig. 1. Architecture of the proposed neural network (9): (a) outputs $\tilde{x}_i (i = 1, \ldots, n)$, which is fed into (b)–(d) on their left-hand sides; (b)–(d) accept $\tilde{x}_i$ and output $x_i (i = 1, \ldots, n)$, $y_j (j = 1, \ldots, m)$ and $z_k (k = 1, \ldots, r)$, respectively, which are then fed into (a) on the left-hand side as its input.

and some interconnections among them. Circuits practitioners may refer to [35] for general idea.

B. Model Comparisons

First, we would like to point out the connection of the proposed neural network with an existing model. If we let $\mathcal{U} = \Re_+^m \times \Re^r, H = 0$ and

$$u = \begin{pmatrix} y \\ z \end{pmatrix} \quad W = \begin{pmatrix} A \\ -C \end{pmatrix} \quad q = \begin{pmatrix} b \\ -d \end{pmatrix}$$

then the neural network (9) can be written as

$$\frac{d}{dt}\begin{pmatrix} u \\ x \end{pmatrix} = -\lambda \begin{pmatrix} 2(u - \mathcal{P}_{\mathcal{U}}(u - Hu - q + Wv)) \\ x - v \end{pmatrix} \quad (10)$$

where $v = \mathcal{P}_X(x - Qx - p - W^T u)$. In fact, (10) represents a neural network model invented in [25] for solving a class of convex quadratic *minimax problems*. In this sense, the neural network (9) can be viewed as an application of (10) for solving QP problems. This novel discovery will eliminate the necessity of detailed stability analysis of (9). See Section II-C.

As discussed in the introduction, for solving the QP problem (1), among the neural networks that entail no calculation of multiplication and inverse of time-varying matrices, the two neural networks (7) and (8) are representatives of the state of the art in terms of both convergence performance and structural complexity. In what follows, we will show that the proposed network (9) can compete with them.

Next, we show the equivalence of the proposed network and the network (8) in terms of model complexity. First, it is evident that the number of integrators of the two networks are the same because both of them have the same state variable $(x^T, y^T, z^T)^T$. Second, (8) entails n amplifiers for realizing the activation function $\mathcal{P}(\cdot)$ and m amplifiers for realizing the activation function $(\cdot)^+$. In order to implement (9), the same number of amplifiers is needed. (Though the output variable $\tilde{x}$ appears three times on the right-hand side of (9), it does not imply that the same operator is needed to be implemented three times.) Third, based on the same idea [that is, $\mathcal{P}(\cdot)$ in (9) is needed to be implemented once only], it is easy to see that the numbers of multipliers and adders to be implemented for (9) are the same as those for (8), respectively. Finally, consider the interconnections among the two neural networks. It is noticed that in Fig. 1 if x_i's instead of $\tilde{x}_i$'s are fed into Fig. 1(c) and (d) as their inputs (see dashed rectangles in Fig. 1), the diagram becomes nearly the same as that of the network (8); the only difference would be the different scaling factors (time constants). In other words, to obtain the diagram of (9) from the diagram of the network (8), what we only need to do is to change the directions of some connections, and no single connection, or other element, is needed to be added to the diagram. In summary, the proposed neural network has exactly the same structural complexity as the neural network (8).

According to the same comparison criteria, it can be shown that the neural network (7) shares nearly the same structural complexity of the neural networks (8) and (9). Nevertheless, this model entails r additional adders and some corresponding connections in contrast to (8) and (9). It is because, in implementing $C^T\tilde{z} = C^T(z - Cx + d)$, $-Cx + d$ are needed to be added to z though the calculation of $-Cx + d$ is not necessary as it can be obtained from the bottom equation on the right-hand side of (7). But the other two neural networks do not need this additional operation. From this sense, (8) and (9) are slightly superior to (7) in general.

Regarding the convergence performance, however, the neural network (7) outperforms the neural network (8) because it is able to solve degenerate QP problems [i.e., in (1), Q is positive semidefinite only] while (8) has not exhibited this capability yet through rigorous analysis. In Section II-C, we will show that the new neural network (9) also has this capability.

Overall, based on the above criteria, the proposed model (9) is the best among the three competitors.

Note that the models (3), (4), (5), and (7) mentioned before are capable of solving both LP and QP problems. There exist other neural networks capable of solving LP problems only, and two typical models are presented in [22] and [23]. We remark that even for solving LP problems only, which are in the form of (1) (Q is set to zero), the proposed neural network is simpler in structure than those in [22] and [23]. For brevity, detailed discussion is omitted here. Interested readers may refer to the two references for a comparison.

C. Stability Results

In this section, we will show that the proposed neural network (9) is stable and globally convergent to a solution of problem (1). First, a lemma is introduced.

Lemma 1 [20, Lemma 1]: A point $x^* \in \Re^n$ is a solution of problem (1) if and only if there exist $y^* \in \Re^m$ and $z^* \in \Re^r$ such that

$$\begin{cases} x^* = \mathcal{P}_X(x^* - Qx^* - p - A^T y^* + C^T z^*) \\ y^* = (y^* + Ax^* - b)^+ \\ Cx^* = d. \end{cases} \tag{11}$$

According to Lemma 1, it is trivial to show one of the significant properties of the neural network (9).

Theorem 1: A point $x^* \in \Re^n$ is a solution of problem (1) if and only if there exist $y^* \in \Re^m$ and $z^* \in \Re^r$ such that they constitute an equilibrium point of the neural network (9).

Another significant property of the proposed neural network refers to its stability and global convergence. This can be shown by rewriting it into (10), which is in the form of a neural network for solving minimax problems in [25]. The following results follow from [25, Th. 3] directly.

Theorem 2: With any initial point $(x(t_0)^T, y(t_0)^T, z(t_0)^T)^T \in \Re^{n+m+r}$, the neural network (9) is stable in the sense of Lyapunov and converges to a solution of (1). In addition, i) if the neural network has a unique equilibrium point, then it is globally asymptotically stable; ii) if λ is large enough, the state trajectory will reach an equilibrium point within finite time.

Theorem 2 actually answers the questions posted in the introduction. It shows that for solving problem (1), the proposed neural network (9) requires the positive semidefiniteness on Q only, instead of positive definiteness that is required by neural network (8). In this sense, neural network (9) is as excellent as neural network (7). Moreover, both of the neural networks do not require that the initial point $x(t_0) \in X$ and $y(t_0) \in \Re^m_+$, but (8) does.

The following corollary is an immediate consequence of Theorem 2.

Corollary 1: In problem (1), let $a_i, c_j \in \Re^{1\times n}$ denote the row vectors of A and C, respectively $(i = 1, \ldots, m, j = 1, \ldots, r)$. If Q is positive definite and the vectors a_i for $i \in \{i \,|\, a_i x^* = b_i\}$, and c_j for $j = 1, \ldots, r$ are linearly independent, then the neural network (9) is globally asymptotically stable in the sense of Lyapunov at its unique equilibrium point $((x^*)^T, (y^*)^T, (z^*)^T)^T$.

Proof: The positive definiteness of Q ensures the uniqueness of the solution x^* to problem (1), while the other conditions ensures the uniqueness of Lagrangian multipliers y^* and z^* [1]. Then, the result follows from Theorem 2 immediately. □

Note that if Q is positive semidefinite only, problem (1) may also have a unique solution, and neural network (9) may also be asymptotically stable.

According to Theorem 2, in circuit implementation, λ should be made as large as possible. But how large is "large enough" is a problem. A useful hint in practice is to specify a solution precision ϵ such that when all trajectories of $\tilde{x}(t)$ enter the set $\{x_i \in \Re \,||x_i(t) - x_i^*| < \epsilon\}$ we terminate the solving procedure. This mechanism works in applications where the optimum solution x^* has special properties, e.g., any component must be either zero or one. See Section III for such an application.

III. A New k-Winners-Take-All Network

A. Introduction

The k-WTA networks for selecting the most prominent elements are central processing components in competitive learning neural networks and nearest-neighbor pattern classifiers. Their task can be mathematically described as follows. Suppose that a list of items $s_1, s_2, \ldots, s_n$ is supplied as inputs to a network. Let $\sigma : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ be the (unknown) permutation that gives the decreasing order of the list

$$s_{\sigma(1)} \geq s_{\sigma(2)} \geq \cdots \geq s_{\sigma(k)} > s_{\sigma(k+1)} \geq \cdots \geq s_{\sigma(n)} \tag{12}$$

where $1 \leq k \leq n-1$ is a given integer. The output of the networks should satisfy

$$\begin{cases} v_{\sigma(1)}, \ldots, v_{\sigma(k)} \in \mathcal{V}_1 \\ v_{\sigma(k+1)}, \ldots, v_{\sigma(n)} \in \mathcal{V}_0 \end{cases} \tag{13}$$

where $v_{\sigma(i)}$ corresponds to the input $s_{\sigma(i)}$, and $\mathcal{V}_1, \mathcal{V}_0$ are two disjoint sets.

There exist many k-WTA networks that can be classified into two categories in terms of their model complexities $O(n^2)$ or $O(n)$. The *complexity* is defined with respect to the number of connections among *neurons* (amplifiers). The study of k-WTA network by using $O(n^2)$ circuits is mainly based on the Hopfield–Tank neural network model invented in [3] and [4], and an example can be found in [36] (see also references therein). Because of the high complexity in circuits design, these networks are limited to solving small scale problems. Some $O(n)$ k-WTA networks can be found in the literature, e.g., [19] and [37]–[41], to list a few. In particular, if all inequalities in (12) take the strict inequality signs, i.e.,

$$s_{\sigma(1)} > s_{\sigma(2)} > \cdots > s_{\sigma(k)} > s_{\sigma(k+1)} > \cdots > s_{\sigma(n)} \tag{14}$$

Marinov and Hopfield proposed the following network [40]:

- state equation

$$\begin{cases} c\dfrac{dx_i}{dt} = -ax_i + b\theta g(\gamma z) + s_i \qquad \forall i = 1, \ldots, n \\ c_0\dfrac{dz}{dt} = -a_0 z - b_0\vartheta \sum_{i=1}^n g(\beta x_i) + b_0\vartheta(2k-n); \end{cases} \tag{15a}$$

- output equation

$$v_i = g(\beta x_i) \qquad \forall i = 1, \ldots, n; \tag{15b}$$

where a, b, c, a_0, b_0, c_0 are fixed circuit parameters and $\theta, \vartheta, \gamma, \beta$ are parameters that need to be appropriately selected according to the input signal s. The function $g(\cdot)$ needs to satisfy some conditions and a typical example is the sigmoid function $g(y) = (e^y - e^{-y})/(e^y + e^{-y})$, which can be regarded as a smoothed projection function defined in (6) by setting $X_i = [0, 1]$. Ultimately, the outputs $v_{\sigma(1)}, \ldots, v_{\sigma(k)}$ will enter $S_1 = [1-\eta, 1)$

($\eta > 0$ is a parameter that needs to be appropriately chosen together with other parameters), while the others will not. A systematic procedure was presented for choosing these parameters. As one example in [40] shows, failing to select proper parameters may lead to incorrect output.

In some references, e.g., [19], [37], and [42], the two sets $\mathcal{V}_1$ and $\mathcal{V}_0$ in (13) are specified as two point sets $\{1\}$ and $\{0\}$, respectively. Then, the k-WTA networks should output a sequence of 1 and 0 only. A newly developed network was presented in [41]

- state equation

$$\frac{1}{\lambda}\frac{dz}{dt} = -\sum_{i=1}^{n} v_i + k; \tag{16a}$$

- output equation

$$v_i = \mathcal{P}_{\Omega_i}\left(z + \frac{s_i}{2a}\right) \qquad \forall i = 1, \ldots, n; \tag{16b}$$

where $z \in \Re$, $\lambda > 0$ is a scaling factor, and $\Omega_i = [0, 1]$ for $i = 1, \ldots, n$. In contrast to (15), this network can solve the problem with the general condition (12). It was shown to be an $O(n)$ network as well. In order to guarantee the global convergence to the correct output, the parameter a should be chosen such that $0 < 2a \leq v_{\sigma(k)} - v_{\sigma(k+1)}$. Clearly, this selection is much easier than selecting those parameters in the network (15). But this network still has limitations. First, anyway, it is demanded to properly select a parameter. Since the signals s_i's are unknown beforehand, a should be selected as small as possible. But smaller a in general leads to slower convergence. For choosing this parameter, one has to make a balance between precision and speed. Second, it was only shown to be able to solve k-WTA problems with the assumption $s_{\sigma(k)} > s_{\sigma(k+1)}$ and may not handle the more general situation

$$s_{\sigma(1)} \geq s_{\sigma(2)} \geq \cdots \geq s_{\sigma(k)} \geq s_{\sigma(k+1)} \geq \cdots \geq s_{\sigma(n)}. \tag{17}$$

Actually, sometimes the following relationship may be true:

$$\begin{aligned} s_{\sigma(1)} \geq \cdots \geq s_{\sigma(k-r-1)} > s_{\sigma(k-r)} = \cdots s_{\sigma(k)} \\ = s_{\sigma(k+1)} \cdots = s_{\sigma(q+k-r-1)} > s_{\sigma(q+k-r)} \geq \cdots \geq s_{\sigma(n)} \end{aligned}$$

where $2 \leq q \leq n, 0 \leq r \leq q-2, k-r \geq 1, q+k-r-1 \leq n$. In what follows, the elements $s_{\sigma(k-r)}, \ldots s_{\sigma(k)}, \ldots, s_{\sigma(q+k-r-1)}$ are all called the kth largest elements in s. One needs to randomly select some of these elements to constitute the k largest ones. This is an ill case of the k-WTA problem, but in many applications, it cannot be avoided. Obviously, in this case, there is no way to choose a proper a for the network (16).

Recently, another k-WTA network was devised, which can conquer the difficulties encountered by (16), [42]:

- state equation

$$\begin{cases} \dfrac{1}{\lambda}\dfrac{du_i}{dt} = -u_i + \displaystyle\sum_{j=1}^{n-1} u_j + \tilde{u}_i + \tilde{w} \qquad \forall i = 1, \ldots, n-1 \\ \dfrac{1}{\lambda}\dfrac{dw}{dt} = 2\displaystyle\sum_{j=1}^{n-1} u_j - \sum_{j=1}^{n-1} \tilde{u}_j + \tilde{w}; \end{cases} \tag{18a}$$

- output equation

$$\begin{cases} v_1 = -\displaystyle\sum_{j=1}^{n-1} u_j + k \\ v_{i+1} = u_i \qquad \forall i = 1, \ldots, n-1; \end{cases} \tag{18b}$$

where $\tilde{u}_i = \mathcal{P}_{\mathcal{U}_i}(u_i - w - s_1 + s_{i+1}), \tilde{w} = \mathcal{P}_{\mathcal{W}}(-\sum_{j=1}^{n-1} u_j - w), \mathcal{U} = \{u \in \Re^{n-1} | 0 \leq u \leq 1\}, \mathcal{W} = \{w \in \Re | -k \leq w \leq 1-k\}$. It is seen that no parameter is needed to choose. Moreover, in view of its formulation in [42] and the analysis in Section III-B, it is easy to see that in the ill case of (17), the network still works. The disadvantage is its relatively complicated structure in contrast to (16). In what follows, we design a simpler network while its merits are retained.

B. Linear Programming Formulation

Let $\mathcal{V}_1$ and $\mathcal{V}_0$ in (13) be $\{1\}$ and $\{0\}$, respectively. It is known that the k-WTA problem described in (12) and (13) is equivalent to the following 0–1 programming problem [37]:

$$\begin{aligned} \text{minimize} \quad & -s^T v \\ \text{subject to} \quad & \sum_{i=1}^{n} v_i = k, \qquad v_i \in \{0, 1\}, \qquad i = 1, \ldots, n. \end{aligned} \tag{19}$$

The following result was claimed in [42] without a proof. Here a complete proof is provided.

Theorem 3: If (12) holds, then problem (19) is equivalent to the following continuous LP problem:

$$\begin{aligned} \text{minimize} \quad & -s^T v \\ \text{subject to} \quad & \sum_{i=1}^{n} v_i = k, \qquad v_i \in [0, 1], \qquad i = 1, \ldots, n. \end{aligned} \tag{20}$$

Proof: In view of the equivalence of the k-WTA problem and (19), if (12) holds, there exists a unique solution v^* to (19). Clearly, v^* is a feasible solution to (20). It is only needed to show that v^* is a strict (local) minimum solution of (20). Consider the perturbation of v^* in the feasible direction, denoted by $v^* + \Delta$, where $|\Delta_i|$ is small for $i = 1, \ldots, n$. In order to satisfy the constraints, the following rules must be fulfilled:

$$\begin{aligned} \Delta_{\sigma(i)} &\leq 0 \qquad \forall i = 1, \ldots, k \\ \Delta_{\sigma(j)} &\geq 0 \qquad \forall j = k+1, \ldots, n \\ \sum_{i=1}^{n} \Delta_{\sigma(i)} &= 0. \end{aligned}$$

Let $\Pi_0 = \{i | \Delta_{\sigma(i)} < 0\}$ and $\Pi_1 = \{j | \Delta_{\sigma(j)} > 0\}$. Then

$$\sum_{i \in \Pi_0} \Delta_{\sigma(i)} + \sum_{j \in \Pi_1} \Delta_{\sigma(j)} = 0.$$

Since $\Delta \neq 0$, neither Π_0 nor Π_1 is null. Define $s_m = \min\{s_{\sigma(i)} | i \in \Pi_0\}$. It follows:

$$s_m \sum_{i \in \Pi_0} \Delta_{\sigma(i)} \geq \sum_{i \in \Pi_0} s_{\sigma(i)} \Delta_{\sigma(i)}.$$

TABLE I
A k-WTA ALGORITHM

1) Solve the LP problem (20) and obtain an optimum v^*.
2) Pick the elements s_i one by one corresponding to $v_i^* = 1$ until k elements are selected.
3) If there are not so many 1's in v^*, then randomly pick some elements s_i corresponding to $0 < v_i^* < 1$ to constitute the k largest elements.

Since problem (19) is equivalent to the k-WTA problem and v^* solves (19), we know from (12) that $s_m > s_{\sigma(j)}, \forall j \in \Pi_1$, which follows:

$$s_m \sum_{j\in\Pi_1} \Delta_{\sigma(j)} > \sum_{j\in\Pi_1} s_{\sigma(j)}\Delta_{\sigma(j)}.$$

Adding the above two inequalities yields

$$\sum_{i\in\Pi_0} s_{\sigma(i)}\Delta_{\sigma(i)} + \sum_{j\in\Pi_1} s_{\sigma(j)}\Delta_{\sigma(j)} < s_m\left(\sum_{i\in\Pi_0}\Delta_{\sigma(i)} + \sum_{j\in\Pi_1}\Delta_{\sigma(j)}\right) = 0.$$

Denote the objective function in (20) by $f(v)$. Then, the perturbation of the objective function near v^* is

$$f(v^*+\Delta) - f(v^*) = -\sum_{i\in\Pi_0} s_{\sigma(i)}\Delta_{\sigma(i)} - \sum_{j\in\Pi_1} s_{\sigma(j)}\Delta_{\sigma(j)} > 0.$$

By considering that Δ is arbitrary, it is concluded that v^* is a strict minimum of (20), which completes the proof. □

Now consider the case of (17). By defining three auxiliary sets

$$\mathcal{S}_1 = \{s_{\sigma(1)}, \ldots, s_{\sigma(k-r-1)}\}$$
$$\mathcal{S}_\gamma = \{s_{\sigma(k-r)}, \ldots, s_{\sigma(k)}, s_{\sigma(k+1)}, \ldots, s_{\sigma(q+k-r-1)}\}$$
$$\mathcal{S}_0 = \{s_{\sigma(q+k-r)}, \ldots, s_{\sigma(n)}\}$$

where all elements in $\mathcal{S}_1$ are greater than those in $\mathcal{S}_\gamma$, all elements in $\mathcal{S}_\gamma$ are greater than those in $\mathcal{S}_0$, and all elements in $\mathcal{S}_\gamma$ are equal, we can obtain the following results.

Theorem 4: Suppose that (17) is valid. Let v^* denote an optimum solution of (20).

1) If $v_i^* = 1$, then $s_i \in \mathcal{S}_1 \cup \mathcal{S}_\gamma$.
2) If $v_i^* = 0$, then $s_i \in \mathcal{S}_0 \cup \mathcal{S}_\gamma$.
3) If $0 < v_i^* < 1$, then $s_i \in \mathcal{S}_\gamma$ and there must exist at least one $j \neq i$ so that $0 < v_j^* < 1$ and $s_j \in \mathcal{S}_\gamma$.
4) There exist at least k nonzero elements in v^*.

Proof: We prove 1) only because 2) and 3) can be reasoned similarly and 4) is obviously true. Construct a feasible solution $\tilde{v}$ to problem (20) by letting $\tilde{v}_i = 0$ and $\tilde{v}_j = v_j^* + \delta_j, \forall j \in \mathcal{J}$, where $0 \le \delta_j \le 1$ and $\mathcal{J} = \{j | j = 1, \ldots, n; j \neq i\}$. In order to satisfy the constraint in (20), we have

$$\sum_{j\in\mathcal{J}} \delta_j = \sum_{j\in\Pi} \delta_j = 1$$

where $\Pi = \{j \in \mathcal{J} | \delta_j > 0\}$. Since v^* is also a feasible solution to problem (20), it has at most k elements equal to 1. As a consequence, we can construct $\tilde{v}$ by letting at least $n-k$ components equal to $v_j^* + \delta_j$ with positive δ_j. In other words, there are at least $n-k$ elements in Π. Let $s_m = \min\{s_j | j \in \Pi\}$, then

$$s_m = s_m \sum_{j\in\Pi} \delta_j \le \sum_{j\in\Pi} s_j\delta_j.$$

Denote the objective function of (20) with v^* and $\tilde{v}$, respectively, by $f(v^*)$ and $f(\tilde{v})$. Then

$$f(\tilde{v}) = f(v^*) + s_i - \sum_{j\in\Pi} s_j\delta_j \le f(v^*) + s_i - s_m$$

which follows $s_i \ge s_m$ because $f(\tilde{v}) \ge f(v^*)$. That is, s_i is greater than or equal to at least $n-k$ elements of the input, which implies that s_i cannot belong to $\mathcal{S}_0$. □

Theorem 4 gives a simple algorithm to select k largest elements of an input s. See Table I. We emphasize that when $\mathcal{S}_1$ or $\mathcal{S}_0$ or both of them are empty, Theorem 5 is still valid and the algorithm in Table I still guarantees the correct output in these extremely ill cases.

C. A New k-WTA Network

According to Table I, the critical step for the k-WTA operation is to solve the LP problem (20). A new k-WTA network based on (9) can be designed for this purpose. Note that in the k-WTA networks (15) and (16), the symbol v is used to represent the output. For the sake of consistency in notations, we need to replace v in (20) with the symbol x first. Then, the following k-WTA network can be designed:

- state equation

$$\begin{cases} \dfrac{1}{\lambda}\dfrac{dx_i}{dt} = -x_i + v_i & \forall i = 1, \ldots, n \\ \dfrac{1}{2\lambda}\dfrac{dz}{dt} = -\displaystyle\sum_{i=1}^{n} v_i + k; \end{cases} \tag{21a}$$

- output equation

$$v_i = \mathcal{P}_{\Omega_i}(x_i + s_i + z) \qquad \forall i = 1, \ldots, n; \tag{21b}$$

where $\Omega_i = [0, 1]$ for $i = 1, \ldots, n$ and λ is any positive constant. Note that x_i can be also regarded as the output of this network, similar to network (9).

The block diagram of this network is illustrated in Fig. 2. Clearly, the complexity of the network is $O(n)$.

According to Theorem 2 and Corollary 1, the following is true.

Corollary 2: If (17) holds, then the k-WTA network (21) is globally convergent to a solution of the problem. If in addition λ is large enough, the convergence time is finite. In particular, if (12) holds, the k-WTA network (21) is globally asymptotically stable.

D. Model Comparisons

Theoretically, the k-WTA network (21) as well as network (18) do not need to choose any parameter (the scaling factor λ can be any positive number), which can be deemed as a great advantage over networks (15) and (16). In addition, the two networks (18) and (21) allow for some equal input signals [see (17)], a situation often encountered in practice but excluded by

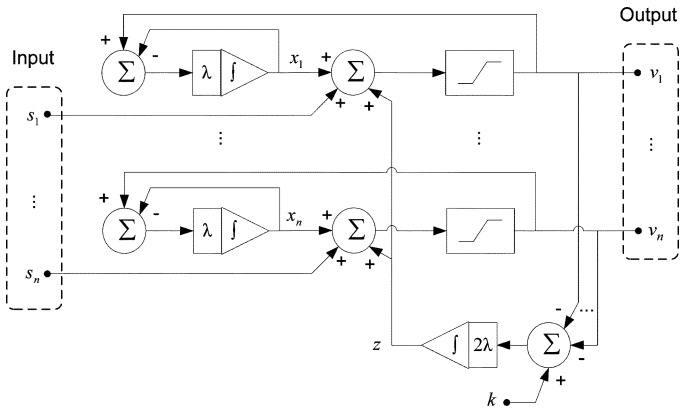

Fig. 2. Architecture of k-WTA network (21).

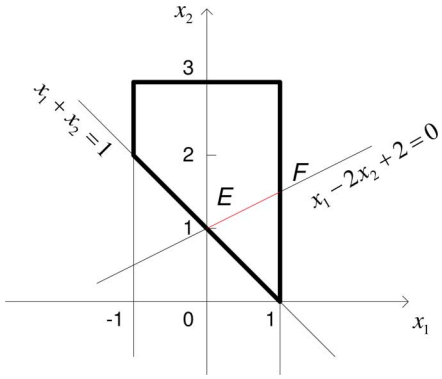

Fig. 3. Analytical solutions to the problem in Example 1.

TABLE II
COMPARISON OF FOUR k-WTA NETWORKS

Model	(15)	(16)	(18)	(21)
Integrators	$n+1$	1	n	$n+1$
Amplifiers	$n+1$	n	n	n
Multipliers	$5n+7$	$n+1$	$n+1$	$n+1$
Adders	$5n+3$	$2n$	$8n-6$	$4n$
Applicability	(14)	(12)	(17)	(17)
Parameter tuning	Yes	Yes	No	No

networks (15) and (16). These facts have been indicated in the last two rows of Table II for a clear comparison. Nevertheless, the latter two have their own merits. For instance, network (15) is much versatile as it can identify the $(k+1)$th largest signal simultaneously.

Then, we compare the structural complexities of the four k-WTA networks (15), (16), (18), and (21). Because all of them are of $O(n)$ complexity, it is necessary to give a more accurate estimation of the number of elements in them. See Table II. Note that in order to distinguish these networks in more details, one *multiplier* is defined in terms of a multiplication operation of two scalars, and similarly, one *adder* is defined in terms of an addition operation of two scalars. Thus, these two numbers are closely related to the number of interconnections among a network.

Let us take the model (21), for example, to show how the numbers in Table II are obtained. It is obvious that this model entails $n+1$ integrators and n amplifiers because it has $n+1$ states and n activation functions. Multiplying $1/\lambda$ to dx_i/dt (or multiplying λ to the right-hand side of the equation) needs n multipliers, and multiplying $1/(2\lambda)$ to dz/dt needs one multiplier. This is why the third entry in the table corresponding to the network is $n+1$. On the right-hand side of the first state equation, there are two scalars needed to be added for each i, so in total these equations need n adders. The second state equation needs n adders as well, because there are $n+1$ items to be added together. Every output equation needs two adders and n such equations need $2n$ adders. Summation of these numbers gives $4n$ adders. The other three networks can be analyzed in the same way by paying attention to the following two points. First, when counting the number of adders in (18), one should keep in mind that the items $\sum_{j=1}^{n-1} u_j$ and $\tilde{u}_i$ need to be calculated once only though they appear at several places. Second, for the sake of fairness, when counting the number of multipliers in (15), the circuit parameters a, b, a_0, b_0 are not taken into account, and each sigmoid function is roughly defined to entail two adders and three multipliers for a quantitative comparison.

From Table II, it is seen that the k-WTA (16) is the simplest model, and network (21) entails one more integrator than (18) but fewer adders (the difference between the numbers is $4n-6$). When n is large, the superiority of (21) over (18) becomes prominent.

From above comparisons, it can be concluded that the new k-WTA network makes a good balance between performance and structural complexity.

Remark 1: Because the LP problem (20) has no inequality constraints, all of the neural networks (3), (4), (5), and (7) can be tailored to solve the problem as well. The simplest one among them, namely, (5), would assume exactly the same architecture as illustrated in Fig. 2, and the only difference would lie in the scaling factors.

IV. NUMERICAL EXAMPLES

In this section, we use several numerical examples to illustrate the performances of proposed neural networks (9) and (21). The simulations are conducted in MATLAB.

Example 1: Consider a QP problem in the form of (1) with

$$Q=\begin{pmatrix}2 & -4\\ -4 & 8\end{pmatrix} \quad p=\begin{pmatrix}4\\ -8\end{pmatrix}$$
$$A=\begin{pmatrix}-1 & -1\end{pmatrix} \quad b=-1$$

and $X=[-1,1]\times[-3,3]$. The equality constraints $Cx=d$ is absent here. It is noted that Q is positive semidefinite only and there are multiple minima to the problem. To see this point, we rewrite the problem in the following equivalent form:

$$\begin{aligned}&\text{minimize} && (x_1-2x_2+2)^2\\ &\text{subject to} && x_1+x_2\geq 1, \qquad x_1\in[-1,1], \qquad x_2\in[-3,3].\end{aligned}$$

The feasible region is depicted in Fig. 3, which is enclosed by bold line segments. Obviously, any point on the line segment $\overline{EF}$ solves the problem.

Neural network (9) is simulated to solve the problem with $\lambda=1$. Fig. 4 displays the state trajectories of the network

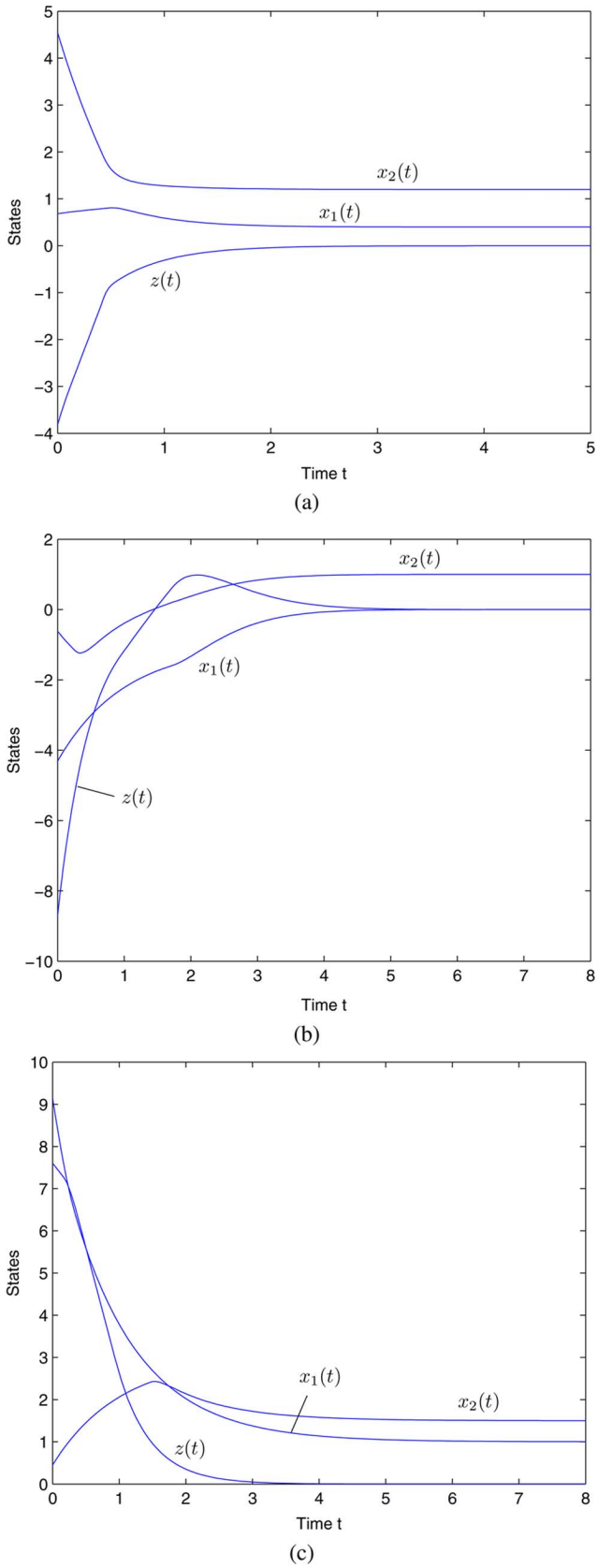

Fig. 4. State trajectories of neural network (9) in Example 1 with three random initial points.

in three independent runs with different initial points. Ultimately, the network converges to $(0.3969, 1.1985, -0.0000)^T$, $(0.0016, 1.0003, 0.0000)^T$, and $(1.0002, 1.5000, 0.0000)^T$, respectively. It is easily verified that the points $(0.3969, 1.1985)^T$, $(0.0016, 1.0003)^T$, and $(1.0002, 1.5000)^T$ are on the line segment $\overline{EF}$ in Fig. 3.

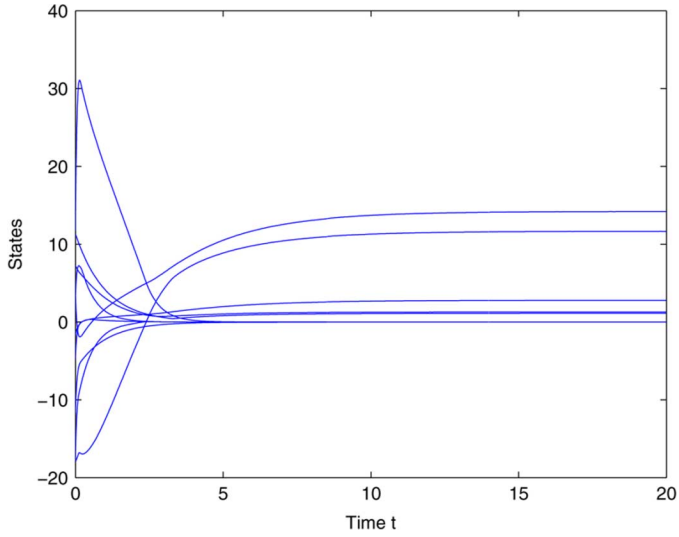

Fig. 5. State trajectories of neural network (9) in Example 2 with a random initial point.

Example 2: Consider a QP problem in the form of (1) with

$$Q = \begin{pmatrix} 11 & 4 & -3 & 4 & 1 \\ 4 & 9 & 0 & 1 & 0 \\ -3 & 0 & 12 & -3 & 0 \\ 4 & 1 & -3 & 4 & 0 \\ 1 & 0 & 0 & 0 & 5 \end{pmatrix} \quad p = \begin{pmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{pmatrix}$$

$$A = \begin{pmatrix} -1 & 2 & 3 & 0 & 0 \\ 2 & 0 & 0 & -2 & 0 \\ 0 & 1 & -1 & -5 & 0 \end{pmatrix} \quad b = \begin{pmatrix} 9 \\ 5 \\ 0 \end{pmatrix}$$

$$C = \begin{pmatrix} -1 & 2 & 0 & -2 & 0 \\ 0 & 0 & 1 & 3 & 0 \end{pmatrix} \quad d = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

and $X = \Re^5_+$. It is easy to check that Q is positive definite. If the neural network (9) is used to solve the problem, according to Theorem 2, it will be globally asymptotically stable with any positive λ. Experiments verified this fact. Actually, all trajectories converged to the unique solution of the problem. For instance, Fig. 5 depicts the transient behavior of the network with $\lambda = 1$ started from a random initial point. The solution found in this run is $x^* = (-0.0000, 2.9560, 0.6320, 1.4559, 0.0000)^T$.

Example 3: Let us solve a k-WTA problem with network (21). The problem is taken from [40], which was designed to test the performance of (15). The list of inputs to be processed is described by

$$s_i = \begin{cases} (181 - 2i)\delta, & i = 1, \ldots, 10 \\ (i+1)2\delta, & i = 11, \ldots, 78 \\ 159\delta, & i = 79 \\ i2\delta, & i = 80, \ldots, 90 \\ (2i - 178)\delta, & i = 91, \ldots, 100. \end{cases}$$

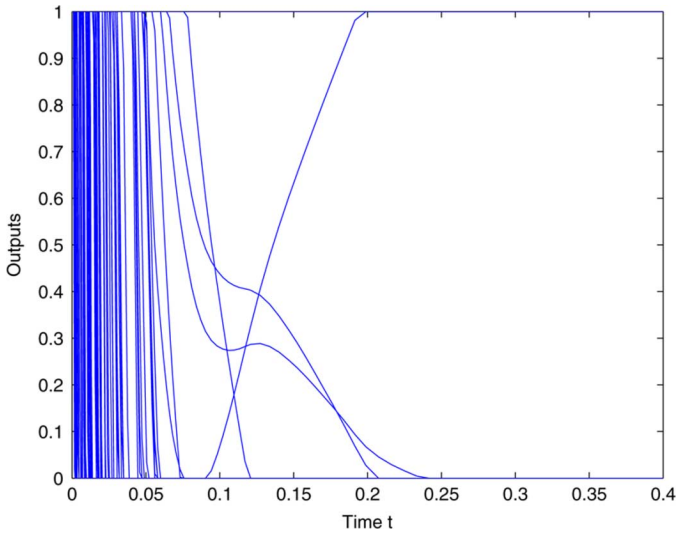

Fig. 6. Output trajectories of k-WTA network (21) in Example 3 with a random initial point.

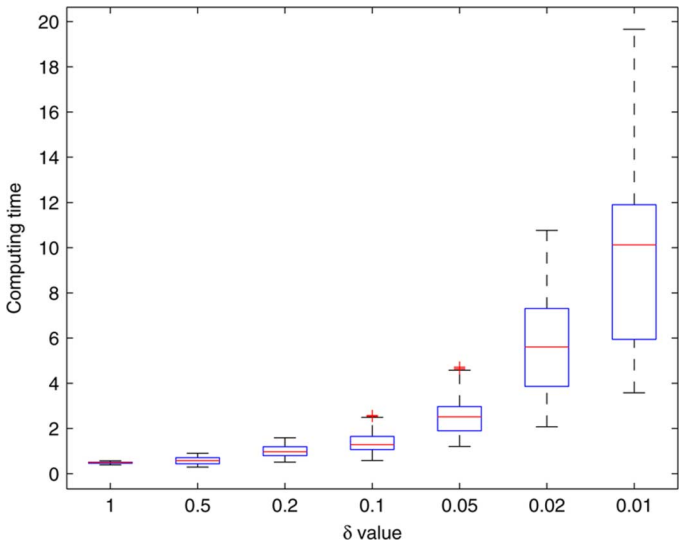

Fig. 7. Computing time for different δ values.

Whatever δ is, the descending list order is as follows:

$$\begin{cases} s_{90} > s_1 > s_{89} > s_2 > s_{88} > \cdots > s_9 > s_{81} > s_{10} \\ \text{the first 20 elements} \\ > s_{80} > s_{79} > \cdots > s_{12} > s_{11} > s_{100} > s_{99} > \cdots > s_{91} \\ \text{the smallest 80 elements.} \end{cases}$$

The task is to signal the largest 20 elements, namely, $s_1, \ldots, s_{10}, s_{81}, \ldots, s_{90}$. This input list is a "difficult" one as the distance between the 20th and 21st largest ones is only δ.

First, we set $\delta = 0.2$ and observe the dynamic behavior of the proposed k-WTA network. It is seen that from any initial point, the network always converges to the correct solution v^*. Fig. 6 depicts the output trajectories in one of these runs, where the scaling factor λ is set to 10.

In what follows, we examine the influence of δ value on the performance of the proposed k-WTA network. The evaluation criterion is the convergence time. It is certainly unrealistic to

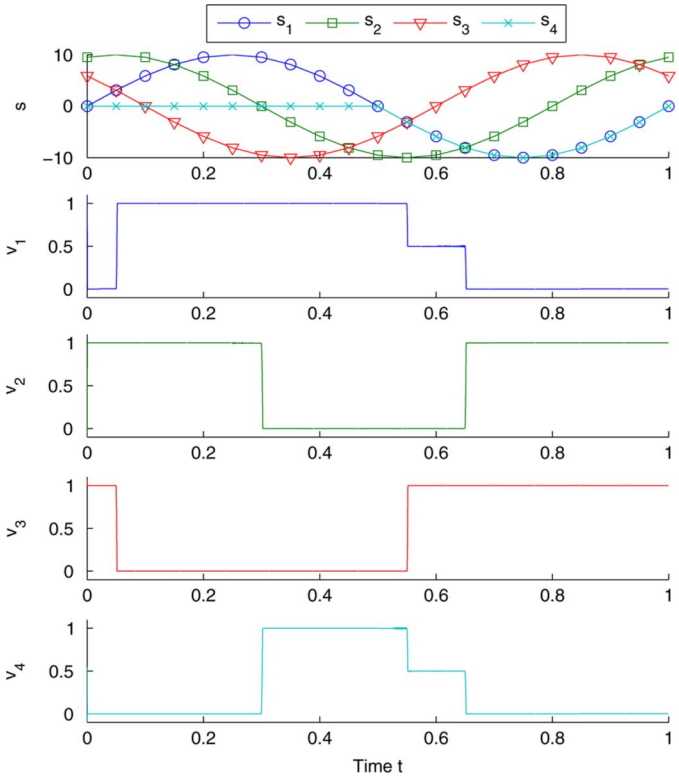

Fig. 8. Inputs and outputs of k-WTA network (21) in Example 4.

expect that the outputs will exactly achieve zeros or ones in the presence of numerical errors (round-off error, integration error, etc.). So we specify a precision ϵ to judge the convergence, namely, when the following holds:

$$\max_{i=1,\ldots,n} \{|v_i(t) - v_i^*|\} \leq \epsilon$$

where v^* stands for the exact solution, which is known in this case. We consider δ equal to $1, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01$, respectively. Fifty simulations are carried out for each δ started from random initial points between -50 and 50. In all simulations, λ is set to 10 and ϵ is set to 10^{-3}. The statistical results are plotted in Fig. 7 by using *boxplot* method.[1] Clearly, if δ decreases, the computing time increases, which is in agreement with the fact that smaller δ makes the 10th and 80th elements closer, and in the sequel, makes the problem more difficult to solve.

Example 4: The last example is used to test whether the proposed k-WTA (21) works if condition (17) is true. In k-WTA problem, let $k = 2, n = 4$ and the inputs $s_i = 10\sin[2\pi(t + 0.2(i-1))], \forall i = 1, 2, 3$, $s_4 = 0$ if $s_1 \geq 0$ and $s_4 = s_1$ otherwise, where t increases from 0 to 1 continuously, which leads to four continuously time-varying input signals (see the top graph in Fig. 8). When $t \geq 0.5$, s_4 coincides with s_4, and t is within approximately $[0.55, 0.65]$, where either of them can be regarded as the second largest element, the k-WTA network (16)

[1]The box has lines at the lower quartile, median, and upper quartile values. The whiskers are lines extending from each end of the box to show the extent of the rest of the data. Outliers denoted by "+" are data with values beyond the ends of the whiskers. A detailed description can be found in MATLAB's documentation.

cannot be applied here. We use network (21) with $\lambda = 10^4$ to accomplish the task and the outputs are recorded in the other four graphs of Fig. 8. Notice that when t is within about $[0.55, 0.65]$, v_1 and v_4 are both equal to 0.5, which implies that either s_1 or s_2 should be selected as a winner according to Theorem 4.

V. Conclusion and Future Works

In this paper, we consider solving a general class of convex QP problems by using a new recurrent neural network. This network shares much similarity in structure with a series of existing neural networks in the literature, and is comparable with the very best in this series in terms of either convergence results or structural complexity. A notable feature of the proposed network is that it can be also used to solve LP problems. The contribution of this invention is twofold. On one hand, it enriches the family of recurrent neural networks for solving LP and QP problems. On the other hand, its design idea and stability analysis may shed light to the development of more powerful models.

Related to this neural network, many further investigations are worth conducting, and we believe that the following directions should be highlighted. It has been seen that the new model (9) is obtained by substituting $\tilde{x}$ for x in the second and third lines of (8). So, on one side, does this substitution offer other advantages besides weaker convergence conditions (e.g., higher convergence rate, more convenience for analog circuit implementation) or, on the contrary, does it sacrifice any performance of its predecessor? On the other side, how about doing such a substitution in the second line or the third line alone? And more generally, how about the combinations of these substitutions: $\tilde{x} \leftrightarrow x, \tilde{y} \triangleq (y + Ax - b)^+ \leftrightarrow y, \tilde{z} \triangleq z - Cx + d \leftrightarrow z$? For instance, what are the characteristics of the following neural network (or the like):

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = -\lambda \begin{pmatrix} 2(x - \mathcal{P}_X(x - Qx - p - A^T\tilde{y} + C^T z)) \\ y - (y + Ax - b)^+ \\ 2(C\tilde{x} - d) \end{pmatrix}.$$

There exists a variety of combinations of such substitutions. But the global convergence results of the resulting neural networks may not be easily relayed from one to another. Most likely, it should be studied one by one. A systematic investigation in this regard is highly demanded, which will, we believe, expand the family of recurrent neural networks for optimization to a large extent.

The latter part of this paper concerns an application of the proposed neural network on the k-WTA problem. The problem is first transformed into an LP problem, and then solved by using the proposed neural network. Compared with other distinguished k-WTA networks, the new scheme features no need for carefully choosing parameters and the capability to handle some ill cases, together with a relatively simple structure. Further investigations on this topic may include circuit realization by means of techniques presented in [27], [35], and [37]–[39] and clocking time estimation for precessing a sequence of signal input lists, which come one by one at a constant rate [43].

Acknowledgment

The authors would like to thank one of the anonymous reviewers for pointing out the connection of the proposed neural network with an existing neural network, which led to the results in Theorem 2 directly. They would also like to thank other reviewers for their suggestions that led to improved quality of this paper.

References

[1] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, 2nd ed. New York: Wiley, 1993.
[2] I. B. Pyne, "Linear programming on an electronic analogue computer," *Trans. Amer. Inst. Electr. Eng.*, vol. 75, pp. 139–143, May 1956.
[3] J. J. Hopfield and D. W. Tank, "Computing with neural circuits: A model," *Science*, vol. 233, no. 4764, pp. 625–633, Aug. 1986.
[4] 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. CAS-33, no. 5, pp. 533–541, May 1986.
[5] M. P. Kennedy and L. O. Chua, "Neural networks for nonlinear programming," *IEEE Trans. Circuits Syst.*, vol. CAS-35, no. 5, pp. 554–562, May 1988.
[6] A. Rodríguez-Vázquez, R. Domínguez-Castro, A. Rueda, J. L. Huertas, and E. Sánchez-Sinencio, "Nonlinear switched-capacitor neural networks for optimization problems," *IEEE Trans. Circuits Syst.*, vol. 37, no. 3, pp. 384–397, Mar. 1990.
[7] J. Wang, "Analysis and design of a recurrent neural network for linear programming," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.* , vol. 40, no. 9, pp. 613–618, Sep. 1993.
[8] X. Wu, Y. Xia, and W. Chen, "A high-performance neural network for solving linear and quadratic programming problems," *IEEE Trans. Neural Netw.*, vol. 7, no. 3, pp. 643–651, May 1996.
[9] Y. Xia, "A new neural network for solving linear and quadratic programming problems," *IEEE Trans. Neural Netw.*, vol. 7, no. 6, pp. 1544–1547, Nov. 1996.
[10] H. Ghasabi-Oskoei and N. Mahdavi-Amiri, "An efficient simplified neural network for solving linear and quadratic programming problems," *Appl. Math. Comput.*, vol. 175, no. 1, pp. 452–464, Apr. 2006.
[11] Q. Tao, J. Cao, and D. Sun, "A simple and high performance neural network for quadratic programming problems," *Appl. Math. Comput.*, vol. 124, no. 2, pp. 251–260, Nov. 2001.
[12] 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.
[13] X. Gao, "A novel neural network for nonlinear convex programming," *IEEE Trans. Neural Netw.*, vol. 15, no. 3, pp. 613–621, May 2004.
[14] X. Xue and W. Bian, "A project neural network for solving degenerate convex quadratic program," *Neurocomputing*, vol. 70, no. 13–15, pp. 2449–2459, Aug. 2007.
[15] X. Hu and J. Wang, "Design of general projection neural networks for solving monotone linear variational inequalities and linear and quadratic optimization problems," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 37, no. 5, pp. 1414–1421, Oct. 2007.
[16] X. Hu and J. Wang, "A recurrent neural network for solving a class of general variational inequalities," *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 37, no. 3, pp. 528–539, Jun. 2007.
[17] 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.
[18] Y. Xia, G. Feng, and J. Wang, "A recurrent neural network with exponential convergence for solving convex quadratic program and linear piecewise equations," *Neural Netw.*, vol. 17, no. 7, pp. 1003–1015, Sep. 2004.
[19] S. Liu and J. Wang, "A simplified dual neural network for quadratic programming with its KWTA application," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1500–1510, Nov. 2006.
[20] Y. Xia, "An extended projection neural network for constrained optimization," *Neural Comput.*, vol. 16, no. 4, pp. 863–883, Apr. 2004.
[21] Y. Xia and G. Feng, "On convergence conditions of an extended projection neural network," *Neural Comput.*, vol. 17, no. 3, pp. 515–525, Mar. 2005.
[22] Y. Xia and J. S. Wang, "Neural-network for solving linear-programming problems with bounded variables," *IEEE Trans. Neural Netw.*, vol. 6, no. 2, pp. 515–519, Mar. 1995.
[23] Y. Xia, "A new neural network for solving linear programming problems and its application," *IEEE Trans. Neural Netw.*, vol. 7, no. 2, pp. 525–529, Mar. 1996.

[24] 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.
[25] 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.
[26] M. Forti, P. Nistri, and M. Quincampoix, "Generalized neural network for nonsmooth nonlinear programming problems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 9, pp. 1741–1754, Sep. 2004.
[27] R. Perfetti and E. Ricci, "Analog neural network for support vector machine learning," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 1085–1091, Jul. 2006.
[28] Q. Liu, J. Cao, and Y. Xia, "A delayed neural network for solving linear projection equations and its analysis," *IEEE Trans. Neural Netw.*, vol. 16, no. 4, pp. 834–843, Jul. 2005.
[29] Q. Liu and J. Cao, "Globally projected dynamical system and its applications," *Neural Inf. Process.—Lett. Rev.*, vol. 7, no. 1, pp. 1–9, Apr. 2005.
[30] Q. Liu and J. Wang, "A one-layer recurrent neural network with a discountinuous hard-limiting activation function for quadratic programming," *IEEE Trans. Neural Netw.*, vol. 19, no. 4, pp. 558–570, Apr. 2008.
[31] Y. Yang and J. Cao, "A delayed network method for solving convex optimization problems," *Int. J. Neural Syst.*, vol. 16, no. 4, pp. 295–303, Aug. 2006.
[32] Y. Xia and J. Wang, "A recurrent neural network for nonlinear convex optimization subject to nonlinear inequality constraints," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 7, pp. 1385–1394, Jul. 2004.
[33] 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.
[34] X. Gao and L. Liao, "A neural network for monotone variational inequalities with linear constraints," *Phys. Lett. A*, vol. 307, no. 2–3, pp. 118–128, Jan. 2003.
[35] L. O. Chua, C. A. Desoer, and E. S. Kuh, *Linear and Nonlinear Circuits*. New York: McGraw-Hill, 1987.
[36] B. D. Calvert and C. A. Marinov, "Another K-winners-take-all analog neural network," *IEEE Trans. Neural Netw.*, vol. 11, no. 3, pp. 829–838, Jul. 2000.
[37] K. Urahama and T. Nagao, "K-winners-take-all circuit with O(N) complexity," *IEEE Trans. Neural Netw.*, vol. 6, no. 3, pp. 776–778, May 1995.
[38] B. Sekerkiran and U. Cilingiroglu, "A CMOS K-winners-take-all circuit with $O(N)$ complexity," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 46, no. 1, pp. 1–5, Jan. 1999.
[39] B. P. Tan and D. M. Wilson, "Semiparallel rank order filtering in analog VLSI," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 2, pp. 198–205, 2001.
[40] C. A. Marinov and J. J. Hopfield, "Stable computational dynamics for a class of circuits with $O(N)$ interconnections capable of kwta and rank extractions," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 5, pp. 949–959, May 2005.
[41] 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.
[42] X. Hu and J. Wang, "Solving the k-winners-take-all problem and the oligopoly Cournot-Nash equilibrium problem using the general projection neural networks," in *Proc. 14th Int. Conf. Neural Inf. Process.*, Kitakyushu, Japan, Nov. 2007, vol. 4984, pp. 703–712.
[43] C. A. Marinov and B. D. Calvert, "Performance analysis for a K-winners-take-all analog neural network: Basic theory," *IEEE Trans. Neural Netw.*, vol. 14, no. 3, pp. 766–780, Jul. 2003.

Xiaolin Hu received the B.E. and M.E. degrees in automotive engineering from Wuhan University of Technology, Wuhan, China, and the Ph.D. degree in automation and computer-aided engineering from The Chinese University of Hong Kong, Hong Kong, in 2001, 2004, and 2007, respectively.

Currently, he is a Postdoctoral Fellow at the State Key Laboratory of Intelligent Technology and Systems, Tsinghua National Laboratory for Information Science and Technology (TNList), and Department of Computer Science and Technology, Tsinghua University, Beijing, China. His current research interests include artificial intelligence and computational neuroscience.

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

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