Neurocomputing 72 (2009) 1679–1687

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

MATLAB Simulink modeling and simulation of LVI-based primal–dual neural network for solving linear and quadratic programs

Yunong Zhang [a,*], Weimu Ma [a], Xiao-Dong Li [a], Hong-Zhou Tan [a], Ke Chen [b]

[a] School of Information Science and Technology, Sun Yat-Sen University, Guangzhou 510275, China
[b] School of Software, Sun Yat-Sen University, Guangzhou 510275, China

A R T I C L E I N F O

Article history:
Received 11 January 2008
Received in revised form
12 May 2008
Accepted 11 July 2008
Communicated by T. Heskes
Available online 10 September 2008

Keywords:
Neural networks
Circuit implementation
Linear programs
Quadratic programs
MATLAB Simulink modeling and simulation

A B S T R A C T

In view of parallel-processing nature and circuit-implementation convenience, recurrent neural networks are often employed to solve optimization problems. Recently, a primal-dual neural network based on linear variational inequalities (LVI) was developed by Zhang et al. for the online solution of linear-programming (LP) and quadratic-programming (QP) problems simultaneously subject to equality, inequality and bound constraints. For the final purpose of field programmable gate array (FPGA) and application-specific integrated circuit (ASIC) realization, we investigate in this paper the MATLAB Simulink modeling and simulative verification of such an LVI-based primal-dual neural network (LVI-PDNN). By using click-and-drag mouse operations in MATLAB Simulink environment, we could quickly model and simulate complicated dynamic systems. Modeling and simulative results substantiate the theoretical analysis and efficacy of the LVI-PDNN for solving online the linear and quadratic programs.

1. Introduction

Linear-programming (LP) and quadratic-programming (QP) problems have been investigated extensively for the past decades, in view of their fundamental roles arising in a wide variety of scientific and engineering applications, such as, pattern recognition [1], signal processing [2], human movement analysis [6], robotic control [20,21], and data regression [22]. In the literature, researches usually solve LP and QP problems separately. In addition, they usually handle optimization problems only subject to one or two special kinds of constraints [23]. However, motivated by real engineering applications of LP and QP in robotics [19–21,24], we have always preferred the following general formulation:

$$\text{minimize} \quad x^{\mathrm{T}}Wx/2+q^{\mathrm{T}}x, \tag{1}$$
$$\text{subject to} \quad Jx=d, \tag{2}$$
$$Ax\leqslant b, \tag{3}$$
$$\xi^{-}\leqslant x\leqslant \xi^{+}, \tag{4}$$

where coefficient-matrix W is assumed to be positive semi-definite (or even zero) such that the convex QP and LP problems are both to be handled in this research work.

The dynamic-system approach is one of the most important parallel computational methods for solving online optimization problems and related issues [2,7,10,13,20,21]. Recently, due to the in-depth research in neural networks, numerous dynamic and analogue solvers based on recurrent neural networks have been developed and investigated [2,7,9,13,16,19–21,23,24]. The neural-dynamic approach is now regarded as a powerful alternative to online computation, in view of parallel distributed processing nature and circuit-implementation convenience [1,2,4,7–10,13,20].

Recently, a primal-dual neural network model has been discovered, developed and exploited by Zhang et al. [18,20,21], which is designed based on the conversion of LP and QP problem formulation into linear variational inequalities (LVI). This LVI-based primal-dual neural network (LVI-PDNN) has a simple piecewise-linear dynamics, global exponential convergence to optimal LP and QP solutions, and superior capability of handling QP and LP problems in the same unified inverse-free manner. As compared with other neural LP/QP solvers such as dual neural networks [23,24], there is no explicit coefficient-matrix inversion of expensively $O(\dim(W)^3)$ operations in this LVI-PDNN. The neural-network architecture and/or computational complexity could thus be much simpler than other existing recurrent neural networks', which is also in view of the piecewise-linear dynamics of such an LVI-based neural network. In addition, compared with traditional primal-dual neural network [12], this LVI-PDNN has a simpler dynamic equation with no high-order nonlinear terms

* Corresponding author. Tel.: +86 20 84113597; fax: +86 20 84113673.
E-mail addresses: ynzhang@ieee.org, zhynong@mail.sysu.edu.cn (Y. Zhang).
URL: http://www.ee.sysu.edu.cn/teacher/detail.asp?sn=129 (Y. Zhang).

0925-2312/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2008.07.008

and also has a more desirable convergence property (i.e., global exponential convergence to optimal solution). These advantages might make the LVI-PDNN approach more probably and widely used in practice [1,7,18,20–22].

Towards the final purpose of field programmable gate array (FPGA) and application-specific integrated circuit (ASIC) realization, we investigate in this paper the MATLAB Simulink modeling and simulative verification [14] of such an LVI-PDNN. To the best of our knowledge, there is little work dealing with the MATLAB Simulink modeling of recurrent neural networks in the literature at present stage. However, the MATLAB Simulink modeling might be an important and necessary step for the final hardware realization of recurrent neural networks [3], in addition to mathematical analysis. Moreover, by using the Simulink HDL Coder (with HDL denoting Hardware Description Language), the recurrent neural networks developed in Simulink environment could further be extended to the HDL code and then to the FPGA and ASIC realization [11,15].

The remainder of this paper is organized in five sections. The mathematical model of LVI-PDNN for solving online QP/LP (1)–(4) is presented and analyzed in Section 2. MATLAB Simulink modeling and simulation techniques are studied and exploited in Section 3 for such an LVI-PDNN model. The neural-network behavior of global exponential convergence to optimal solutions is illustrated via numerical experiments in Section 4. Lastly, Section 5 concludes the paper with final remarks. The main contributions of this paper are worth pointing out as follows.

(1) In this paper, we present an LVI-PDNN, which could handle the hybrid-constraint QP and LP problems depicted in (1)–(4). In addition to global exponential convergence to optimal solutions, such a recurrent neural network has a simple piecewise-linear dynamics and might be realized via FPGA and ASIC implementation more readily than other existing neural models.
(2) In this paper, we show that, by using convenient click-and-drag mouse operations (instead of conventional compilation of program code), the LVI-PDNN simulation model can be built up systematically and easily by means of MATLAB Simulink function blocks. Programming efforts could thus be reduced drastically.
(3) To show the characteristics of the LVI-PDNN, two illustrative examples with a large number of numerical experiments are presented. The first example is about online solution of a strictly-convex quadratic program, while the second one is about online solution of a linear program for comparison purposes. Modeling and simulative results substantiate the efficacy and hardware-realizability of the LVI-PDNN for solving both QP and LP problems.

2. The LVI-PDNN model

In this section, we briefly describe the LVI-PDNN, its structure and theoretical results.

2.1. Model description

To solve online the QP or LP problem depicted in (1)–(4), based on the conversion of such a QP/LP to a set of LVI and then to a system of piecewise linear equations, a primal-dual neural network could be developed with its dynamics given as follows [18,20,21]:

$$\dot{y} = \gamma(I + H^{\mathrm{T}})\{\mathscr{P}_{\Omega}(y - (Hy + p)) - y\}, \tag{5}$$

where $\gamma > 0$ is an inductance parameter or the reciprocal of a capacitance parameter used to scale the network convergence, and should be set as large as possible in hardware implementation [2,7,9,13,20]. The primal-dual decision vector $y \in \mathbb{R}^n$ and its upper/lower bounds $\varsigma^{\pm}$ are defined, respectively, as

$$y = \begin{bmatrix} x \\ u \\ v \end{bmatrix}, \quad \varsigma^{+} = \begin{bmatrix} \xi^{+} \\ \varpi \\ \varpi \end{bmatrix} \in \mathbb{R}^n, \quad \varsigma^{-} = \begin{bmatrix} \xi^{-} \\ -\varpi \\ 0 \end{bmatrix} \in \mathbb{R}^n, \tag{6}$$

where, with $n = \dim(x) + \dim(u) + \dim(v)$ defined,

- $x \in [\xi^{-}, \xi^{+}]$ is evidently the original decision variable vector used in primal QP/LP (1)–(4);
- $u \in \mathbb{R}^{\dim(d)}$ denotes the dual decision variable vector defined for equality constraint (2);
- $v \geqslant 0 \in \mathbb{R}^{\dim(b)}$ denotes the dual decision variable vector defined for inequality constraint (3); and,
- ϖ is a vector of appropriate dimensions with each element sufficiently large to replace $+\infty$ numerically.

In LVI-PDNN dynamic Eq. (5), $\mathscr{P}_{\Omega}(\cdot)$ is an operator which projects an n-dimensional vector onto the set $\Omega := [\varsigma^{-}, \varsigma^{+}] \subset \mathbb{R}^n$. In addition, the ith element of $\mathscr{P}_{\Omega}(y)$ is

$$\mathscr{P}_{\Omega}^{(i)}(y_i) = \begin{cases} \varsigma_i^{-} & \text{if } y_i < \varsigma_i^{-}, \\ y_i & \text{if } \varsigma_i^{-} \leqslant y_i \leqslant \varsigma_i^{+}, \\ \varsigma_i^{+} & \text{if } y_i > \varsigma_i^{+}. \end{cases} \tag{7}$$

The coefficient-matrix $H \in \mathbb{R}^{n \times n}$ and vector $p \in \mathbb{R}^n$ appearing in (5) are defined by augmentation as follows:

$$H = \begin{bmatrix} W & -J^{\mathrm{T}} & A^{\mathrm{T}} \\ J & 0 & 0 \\ -A & 0 & 0 \end{bmatrix}, \quad p = \begin{bmatrix} q \\ -d \\ b \end{bmatrix}. \tag{8}$$

2.2. Network architecture

Expressed in the ith-neuron form (with $i = 1, \ldots, n$), LVI-PDNN model (5) can be further written as

$$\frac{\mathrm{d}y_i}{\mathrm{d}t} = \gamma \sum_{j=1}^{n} c_{ij} \left[\mathscr{P}_{\Omega}^{(j)} \left(\sum_{k=1}^{n} s_{ik} y_k - p_i \right) - y_j \right], \tag{9}$$

where c_{ij} denotes the ijth entry of scaling matrix $C = I + H^T$, and s_{ik} denotes the ikth entry of matrix $S = I - H$. The block diagram realization of LVI-PDNN (5) could be given in Fig. 1, while a more detailed architecture of the neural network based on neuron expression (9) is shown in Fig. 2. The piecewise-linear projection operator (or called piecewise-linear activation function) $\mathscr{P}_{\Omega}(\cdot)$ can be implemented by using operational amplifiers known as limiter [2,5,9,23].

Before ending this subsection, it is worth noting the following two remarks.

Remark 1. As bound constraint (4) is neatly cast into the projection set Ω, the dual decision variable vector originally defined for bound constraint (4) is finally removed. Thus, the size of LVI-PDNN neural system (5) is only n, being the dimensional sum of equality constraint (2), inequality constraint (3) and primal decision vector x, of which the number is surprisingly the same as that in dual neural network [23,24].

Remark 2. In view of Eq. (9) and Fig. 2, the circuit implementing the LVI-PDNN (5) consists of n integrators, n limiters (being the piecewise-linear activation functions), $2n^2$ multipliers and

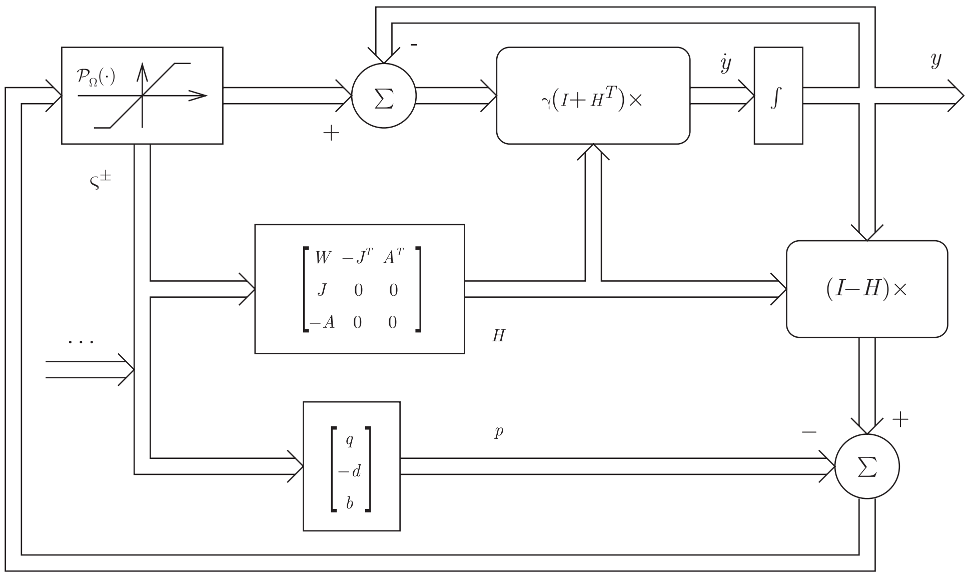

$\mathcal{P}_\Omega(\cdot)$
$\varsigma^\pm$
Σ
$\gamma(I+H^T)\times$
$\dot{y}$
$\int$
y
$\begin{bmatrix} W & -J^T & A^T \\ J & 0 & 0 \\ -A & 0 & 0 \end{bmatrix}$
H
$(I-H)\times$
$\begin{bmatrix} q \\ -d \\ b \end{bmatrix}$
p

Fig. 1. Block diagram realization of LVI-PDNN (5) for solving QP/LP (1)–(4).

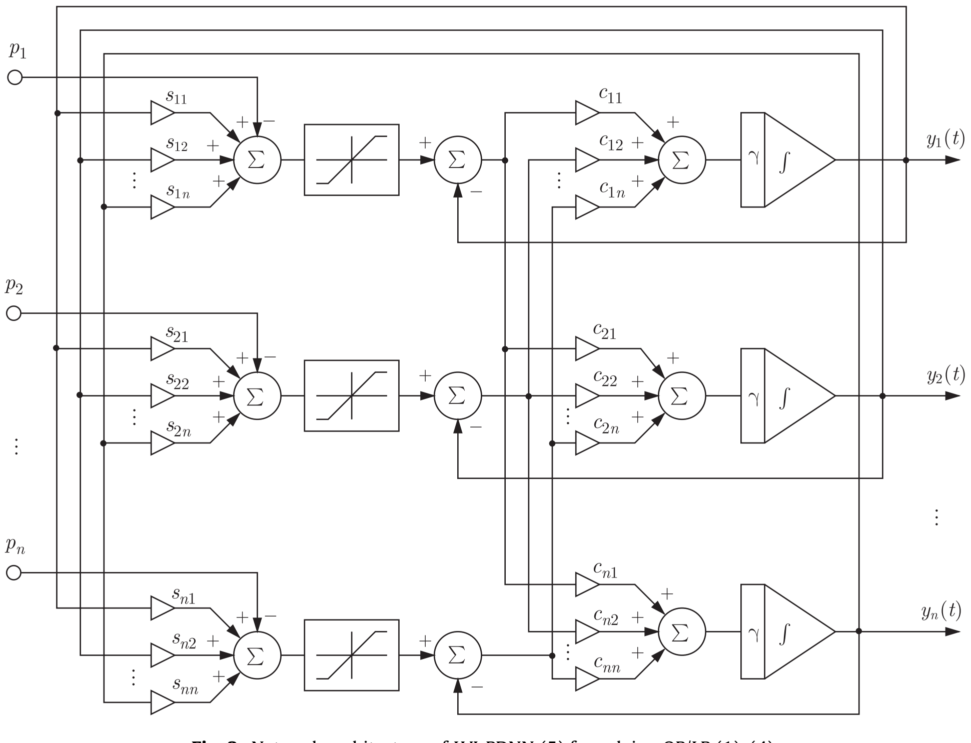

p_1
p_2
p_n
s_{11} s_{12} s_{1n}
s_{21} s_{22} s_{2n}
s_{n1} s_{n2} s_{nn}
c_{11} c_{12} c_{1n}
c_{21} c_{22} c_{2n}
c_{n1} c_{n2} c_{nn}
γ
$y_1(t)$
$y_2(t)$
$y_n(t)$

Fig. 2. Network architecture of LVI-PDNN (5) for solving QP/LP (1)–(4).

$5n$ summers. In contrast, for solving the same LP and QP problem, even without counting the computational complexity $O(\dim(W)^3)$ of online matrix inversion, the circuit implementation of dual neural network [23,24] needs n integrators, n limiters, $(2+\dim(W))n^2+(\dim(W)^2+\dim(W))n$ multipliers and $n^2+(\dim(W)+4)n$ summers. Modeling results here and in the ensuing section both substantiate that the presented LVI-PDNN could have an at-least half reduction of the structural and computational complexity (especially, in terms of the numbers of multiplications and additions/subtractions), as compared to the dual-neural-network model.

2.3. Global convergence

Moreover, summarized from the analysis results of [17,18,20,21], the following lemmas could be given about the global exponential convergence and stability of the LVI-PDNN (5) [or its neuron form (9)].

Lemma 1. *Assume the existence of at least one optimal solution x^* to the QP/LP optimization problem* (1)–(4). *Starting from any initial state $y(0)\in\mathbb{R}^n$, the state vector $y(t)$ of the LVI-PDNN* (5) *is always convergent to an equilibrium state y^*, of which the first* $\dim(x)$

elements constitute the optimal solution x^*. Furthermore, the exponential convergence can be achieved for LVI-PDNN (5)*, provided that there exists a constant $\varrho > 0$ such that $\|y - \mathscr{P}_\Omega(y - (Hy + p))\|_2^2 \geqslant \varrho\|y - y^*\|_2^2$ holds true.*

Lemma 2. *The existence of equilibrium state y^* of the LVI-PDNN* (5) *is equivalent to the condition that the feasible region made by constraints* (2)–(4)*, denoted by Φ, is nonempty. In contrast to Lemma* 1*, if the feasible region Φ made by constraints* (2)–(4) *is empty* (*i.e., $\Phi = \varnothing$*)*, then the state vector $y(t)$ of the LVI-PDNN* (5) *is generally divergent.*

3. MATLAB Simulink modeling

MATLAB Simulink is a graphical-design based modeling tool which exploits existing function blocks to construct mathematical and logical models and process flow [14]. The MATLAB Simulink modeling could thus be viewed as a virtual implementation of a real system that satisfies a set of engineering requirements. Moreover, as mentioned before, the neural-network model developed in MATLAB Simulink environment could be extended to the HDL code and then to the final FPGA and ASIC realization [11,15]. In this section, the MATLAB Simulink modeling techniques are investigated as follows for the LVI-PDNN (5).

3.1. Basic function blocks

MATLAB Simulink includes a comprehensive block library of sinks, sources, linear and nonlinear components, and connectors. The Simulink function blocks used to construct the LVI-PDNN model (5) are shown in Fig. 3 and briefed below.

- The *Constant* block, which outputs a constant specified by its parameter "constant value", could be used to generate the matrices W, J, A and the vectors q, d, b, $\xi^{\pm}$ appearing in QP/LP (1)–(4) and LVI-PDNN (5).
- The *Gain* block could be used to scale the LVI-PDNN network convergence, e.g., as a representation of design parameter γ in the neural-network system (5).
- The *Product* block, specified as the standard matrix-product mode, could be used to multiply the matrices and vectors involved in the neural-network system (5).
- The *MATLAB Fcn* block could be used to generate zero matrix 0, identity matrix I, vector of ones, and so on.

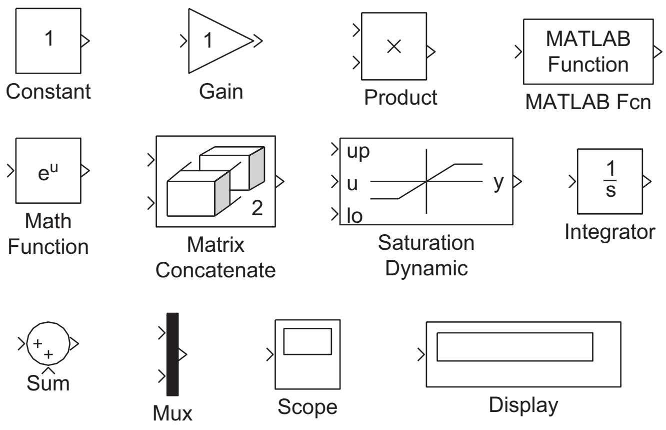


Fig. 3. Basic Simulink function blocks used for constructing LVI-PDNN (5).

- The *Math Function* block could perform various common mathematical operations, and is used in our context only for generating the transpose of a matrix.

Other Simulink function blocks involved in modeling (5) will be introduced later when encountered. By interconnecting these basic Simulink function blocks and setting appropriate block-parameters, the model of LVI-PDNN (5) can thus be built up readily.

3.2. Generating matrix H

As seen from Eq. (8) and Fig. 1, the augmented coefficient-matrix H in LVI-PDNN (5) is composed of matrices W, J, A, their transposes, and zero matrices. For simple simulation purposes, one may pre-assign the value of H directly from Eq. (8) and then use H for constructing neural-network model (5). However, for the purposes of systematic modeling and further circuit-implementation, we prefer to use the *Math Function* block and *Matrix Concatenate* block to generate matrix H automatically via Eq. (8) and from matrices W, J and A.

- To construct the sub-model for generating matrix H, using the *Math Function* block, we can select the "transpose" function from its "function" parameter list.
- The *Matrix Concatenate* block concatenates input matrices to create an output matrix. In our modeling context, this block operates in a multidimensional-array mode and uses its "concatenate dimension" parameter to specify the output dimension along which to concatenate the input matrices. For example, to concatenate two matrices vertically, we could specify its "concatenate dimension" parameter as "1"; and, to concatenate two matrices horizontally, we could specify its "concatenate dimension" parameter as "2".

The Simulink subsystem for generating matrix H is shown in Fig. 4, which is a part of the Simulink model of LVI-PDNN (5). Its corresponding block-diagram design and representation are shown in Fig. 4(a). In total, to construct this subsystem, we exploit two *Math Function* blocks, four *Matrix Concatenate* blocks, two *Gain* blocks, one *MATLAB Fcn* block, and three *Constant* blocks.

3.3. Constructing projection operator $\mathscr{P}_\Omega(\cdot)$

As seen from Eq. (5) and Fig. 1, before constructing the LVI-PDNN in MATLAB Simulink environment, we have to build up the projection operator $\mathscr{P}_\Omega(\cdot)$ defined in Eq. (7). This projection operator is also termed piecewise-linear activation function [5,23] or Saturation function [14]. In modeling the projection operator $\mathscr{P}_\Omega(\cdot)$, we can exploit the Simulink *Saturation Dynamic* block as its upper and lower bound-values could be determined dynamically via the first and third inputs. Then, the *Saturation Dynamic* block limits the range of the second input within the first input (as upper bound) and the third input (as lower bound). As seen from the defining Eqs. (6) and (7), the upper and lower bound-values of such a *Saturation Dynamic* block in our case are

$$\varsigma^+ = \begin{bmatrix} \xi^+ \\ \varpi \\ \varpi \end{bmatrix}, \quad \varsigma^- = \begin{bmatrix} \xi^- \\ -\varpi \\ 0 \end{bmatrix}.$$

The Simulink subsystem for constructing projection operator $\mathscr{P}_\Omega(\cdot)$ could thus be developed and depicted in Fig. 5, being a part of the Simulink model of LVI-PDNN (5). Here, we use 10^9 to constitute ϖ through the *MATLAB Fcn* blocks and *Gain* blocks, with

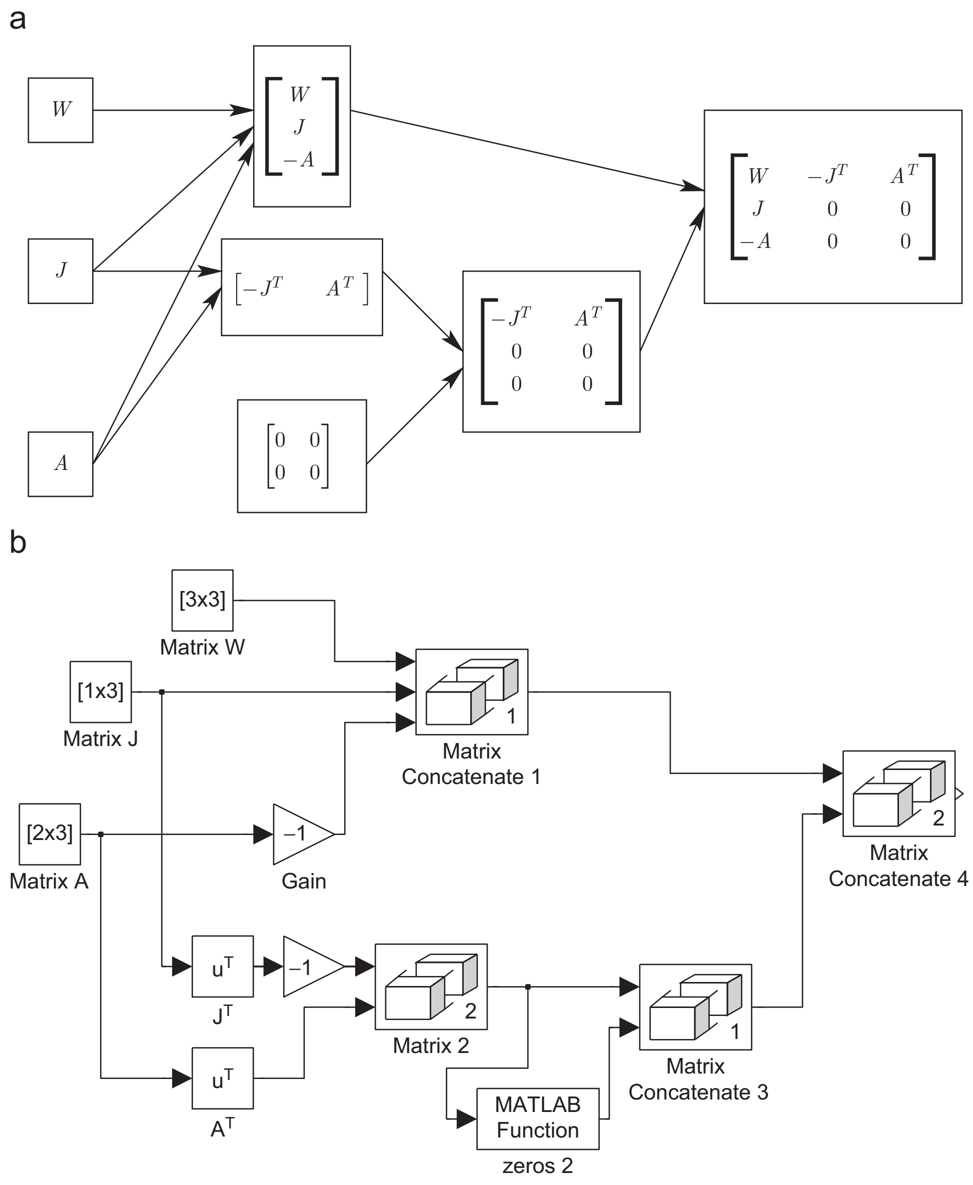

a
W
J
A
$\begin{bmatrix} W \\ J \\ -A \end{bmatrix}$
$\begin{bmatrix} -J^T & A^T \end{bmatrix}$
$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
$\begin{bmatrix} -J^T & A^T \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$
$\begin{bmatrix} W & -J^T & A^T \\ J & 0 & 0 \\ -A & 0 & 0 \end{bmatrix}$
b
[3x3] Matrix W
[1x3] Matrix J
[2x3] Matrix A
−1 Gain
Matrix Concatenate 1
Matrix Concatenate 4
u^T J^T
u^T A^T
−1
Matrix 2
MATLAB Function zeros 2
Matrix Concatenate 3

Fig. 4. Subsystem for generating matrix H used in LVI-PDNN (5). (a) Block-diagram representation and (b) corresponding Simulink model.

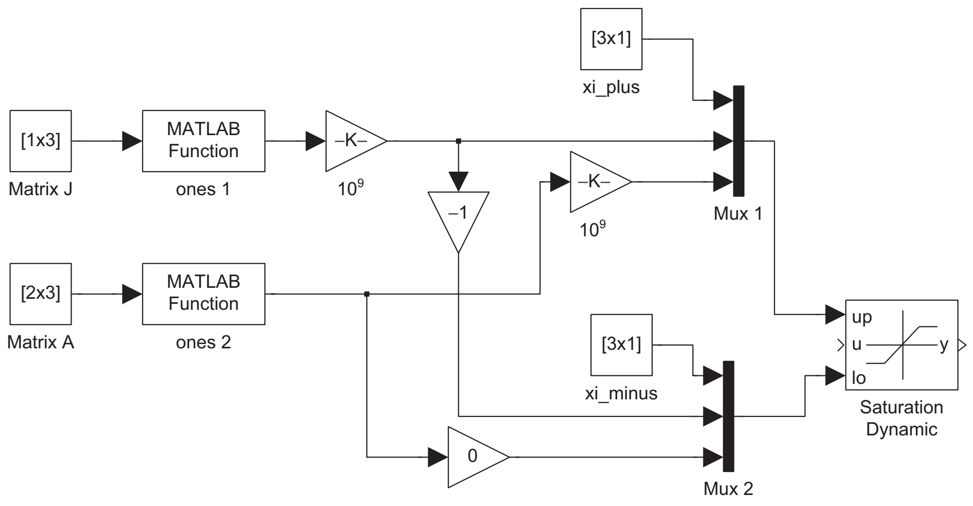


Fig. 5. Simulink model for constructing projection function $\mathscr{P}_{\Omega}(\cdot)$.

MATLAB Fcn blocks generating properly-dimensioned vectors made of ones.

3.4. Final options setting

After developing the subsystems for generating H and $\mathscr{P}_{\Omega}(\cdot)$ as shown in the preceding two subsections, the overall Simulink model of LVI-PDNN (5) can thus be built up easily and presented in Fig. 6. Prior to running the Simulink model, we may need to set up a few environment-options and parameters, such as,

- Start time (e.g., 0.0) and Stop time (e.g., 6.0);
- Solver (i.e., integrator algorithm): "ode15s";
- Max step size (e.g., 0.2) and Min step size (e.g., "auto");
- Initial step size (e.g., "auto");
- Relative tolerance (e.g., 10^{-3});
- Absolute tolerance (e.g., "auto");
- Maximum order (e.g., 5).

This options setting can be done by using the "configuration parameters" dialog box in MATLAB Simulink environment, which is illustrated in Fig. 7 for readers' convenience. Other environment-options may be assigned to be their default values. For example, to draw more clearly the x-trajectory figures (such as Fig. 9), we can save the following variables to workspace by ticking them off: time (tout) and states (yout).

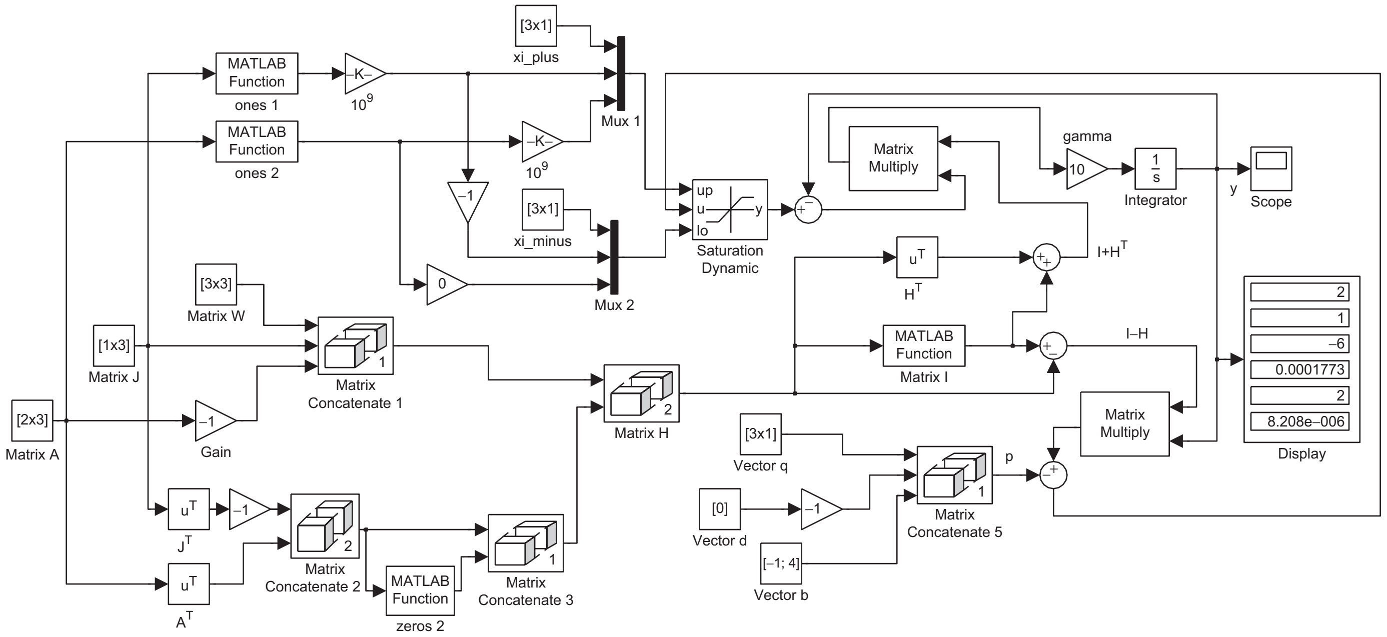
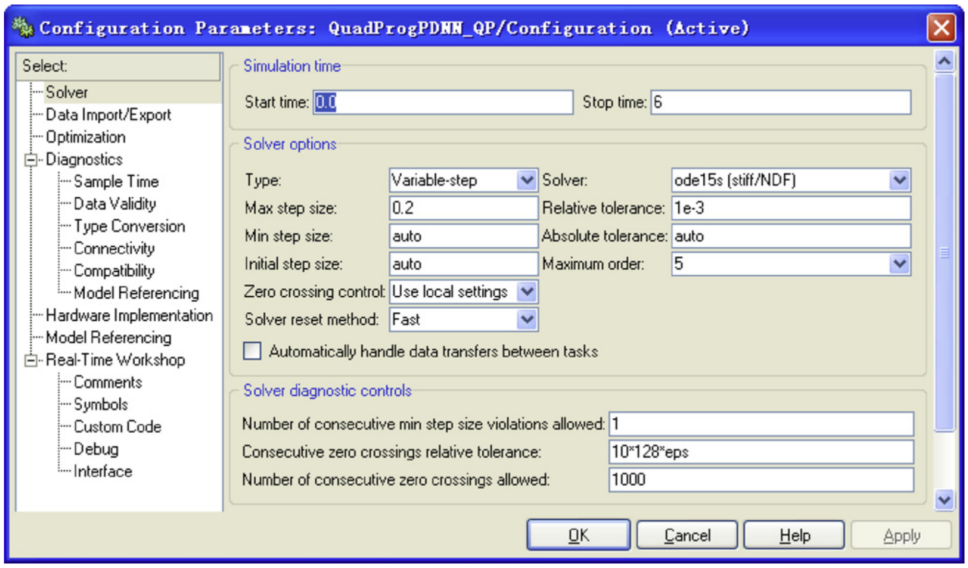


Fig. 6. Overall Simulink modeling of LVI-based primal-dual neural network (5) for solving online the QP and LP problems depicted in (1)–(4).


Configuration Parameters: QuadProgPDNN_QP/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
  - Sample Time
  - Data Validity
  - Type Conversion
  - Connectivity
  - Compatibility
  - Model Referencing
- Hardware Implementation
- Model Referencing
- Real-Time Workshop
  - Comments
  - Symbols
  - Custom Code
  - Debug
  - Interface

Simulation time
Start time: 0.0 Stop time: 6

Solver options
Type: Variable-step Solver: ode15s (stiff/NDF)
Max step size: 0.2 Relative tolerance: 1e-3
Min step size: auto Absolute tolerance: auto
Initial step size: auto Maximum order: 5
Zero crossing control: Use local settings
Solver reset method: Fast
☐ Automatically handle data transfers between tasks

Solver diagnostic controls
Number of consecutive min step size violations allowed: 1
Consecutive zero crossings relative tolerance: 10*128*eps
Number of consecutive zero crossings allowed: 1000

Fig. 7. Simulink configuration dialog-box for setting up the environment-options and parameters of modeling LVI-PDNN (5).

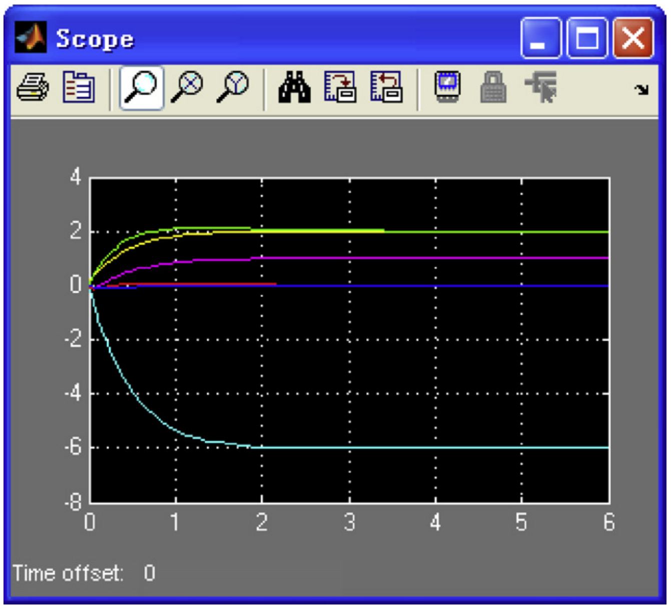


Fig. 8. Simulink "Scope" illustration of $y(t)$ of LVI-PDNN (5) solving QP problem.

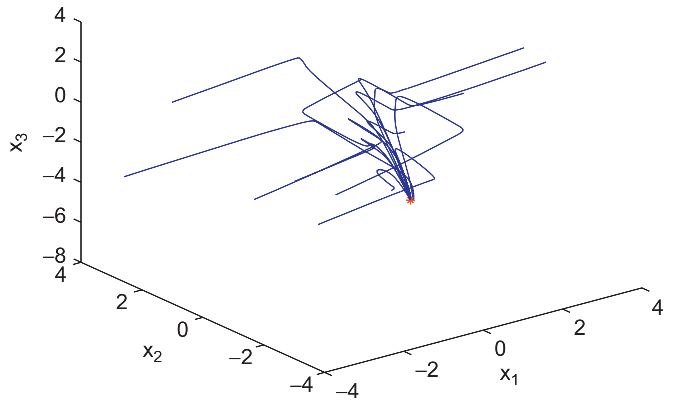


Fig. 9. Spatial trajectories of (x_1, x_2, x_3) synthesized by LVI-PDNN (5), starting from ten different initial states and converging to the joint equilibrium state x^* denoted by *, which is the theoretical optimal solution to QP (1)–(4).

4. Simulative verification

Based on the MATLAB Simulink modeling techniques investigated in the preceding section and Fig. 6, the neural-network behavior is shown in this section via the following two illustrative examples. Simply speaking, the first example is about QP, while the second one is about LP.

Example 1. We consider the online solution of the following strictly-convex QP problem [23] by LVI-PDNN (5):

$$\begin{aligned} \text{minimize} \quad & 11x_1^2 + x_2^2 + x_3^2 - 2x_1x_2 + 6x_1x_3 - 4x_1, \\ \text{subject to} \quad & 2x_1 + 2x_2 + x_3 = 0, \\ & -1x_1 + x_2 \leqslant -1, \\ & 3x_1 + x_3 \leqslant 4, \\ & -6 \leqslant x_1, x_2, x_3 \leqslant 6, \end{aligned}$$

which could be rewritten in the compact matrix form, (1)–(4), and thus we could have the coefficient matrices and vectors defined as $q = [-4, 0, 0]^T$, $J = [2, 2, 1]$, $d = 0$, $b = [-1, 4]^T$, $\xi^+ = -\xi^- = [6, 6, 6]^T$, and

$$W = \begin{bmatrix} 22 & -2 & 6 \\ -2 & 2 & 0 \\ 6 & 0 & 2 \end{bmatrix}, \quad A = \begin{bmatrix} -1 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix}.$$

After assigning the values of W, q, J, d, A, b and $\xi^\pm$ into the Simulink model depicted in Fig. 6, the transient behavior and performance of LVI-PDNN (5) could be simulated readily.

- The simulation result starting from zero initial state $y(0) = 0 \in \mathbb{R}^6$ is illustrated in Fig. 8. With $\gamma = 10$, state vector $y(t) \in \mathbb{R}^6$ of LVI-PDNN (5) converges to equilibrium state $y^* = [2, 1, -6, 1.773 \times 10^{-4}, 2, 8.208 \times 10^{-6}]^T$ in 3.72 s, and its first 3 elements constitute numerically the optimal solution $x^* = [2, 1, -6]^T$ to the above QP problem.
- The simulation result starting from random initial state $y(0) \in [-5, 5]^6$ is illustrated in Fig. 9. Note that, here, the "initial condition" parameter of the *Integrator* block is set to be $10 * (\text{rand}(6, 1) - 1/2)$. It is seen from Fig. 9, started with an initial state randomly selected within $[-5, 5]^6$, the LVI-PDNN (5) always converges to the optimal solution x^* of such a QP problem.
- We observe from a large number of numerical tests of LVI-PDNN (5) that, with design parameter $\gamma = 10^4$, the average convergence time for achieving the accuracy of 10^{-3} is about 4.46 ms. In addition, with exponential convergence guaranteed [20,21,18,17], when design parameter $\gamma = 10^7$, the average convergence time for achieving the accuracy of 10^{-3} is about 4.40 μs.

Example 2. We consider the online solution of the following LP problem [2] by LVI-PDNN (5):

$$\begin{aligned} \text{minimize} \quad & -x_1 - 5x_2 + 2x_5 + 3, \\ \text{subject to} \quad & x_1 + x_3 + 2x_4 = 7, \\ & x_1 + 2x_2 - x_3 + 4x_5 = 11, \\ & 0 \leqslant x_1 \leqslant 4, \quad 0 \leqslant x_2 \leqslant 5, \quad 0 \leqslant x_3 \leqslant 1, \\ & 0 \leqslant x_4 \leqslant 2, \quad 0 \leqslant x_5 \leqslant 3, \end{aligned}$$

which could be rewritten in the compact matrix form, (1)–(4), and thus we could have the coefficient matrices and vectors defined as $W = 0 \in \mathbb{R}^{5\times5}$, $q = [-1, -5, 0, 0, 2]^T$, $A = 0 \in \mathbb{R}^{1\times5}$, $b = 0 \in \mathbb{R}^1$, $\xi^- = 0 \in \mathbb{R}^5$, $\xi^+ = [4, 5, 1, 2, 3]^T$, and

$$J = \begin{bmatrix} 1 & 0 & 1 & 2 & 0 \\ 1 & 2 & -1 & 0 & 4 \end{bmatrix}, \quad d = \begin{bmatrix} 7 \\ 11 \end{bmatrix}.$$

After assigning the values of W, q, J, d, A, b and $\xi^\pm$ into the Simulink model depicted in Fig. 6, the transient behavior and performance of LVI-PDNN (5) could be simulated readily for solving online the above LP problem.

- The simulation result starting from zero initial state $y(0) = 0 \in \mathbb{R}^8$ is illustrated in Fig. 10. With $\gamma = 10$, state vector $y(t) \in \mathbb{R}^8$ of LVI-PDNN (5) converges to an equilibrium state $y^* = [2, 5, 1, 2, 6.505 \times 10^{-9}, 9.736 \times 10^{-2}, -1.097, 0]^T$ in 0.879 s, and its first five elements constitute numerically the optimal solution $x^* = [2, 5, 1, 2, 0]^T$ to the above LP problem.
- The simulation result starting from random initial state $y(0) \in [-5, 5]^5$ is illustrated in Fig. 11. It shows that, started with an initial state randomly selected within $[-5, 5]^5$, the LVI-PDNN (5) always converges to this unique optimal solution x^* of such an LP problem (denoted by * in red).
- We observe from a large number of numerical tests as well that, with design parameter $\gamma = 10^4$, the average convergence time for achieving the accuracy of 10^{-3} in this LP problem solving is about 0.9308 ms. In addition, with exponential

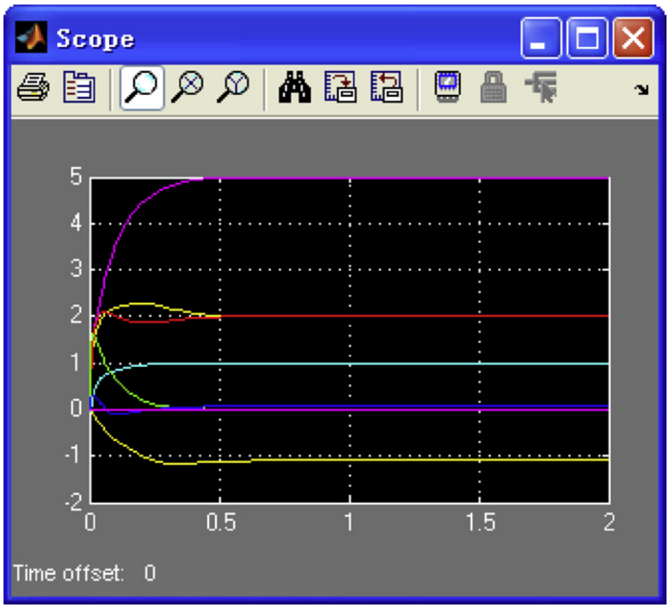


Fig. 10. Simulink "Scope" illustration of $y(t)$ of LVI-PDNN (5) solving LP problem.

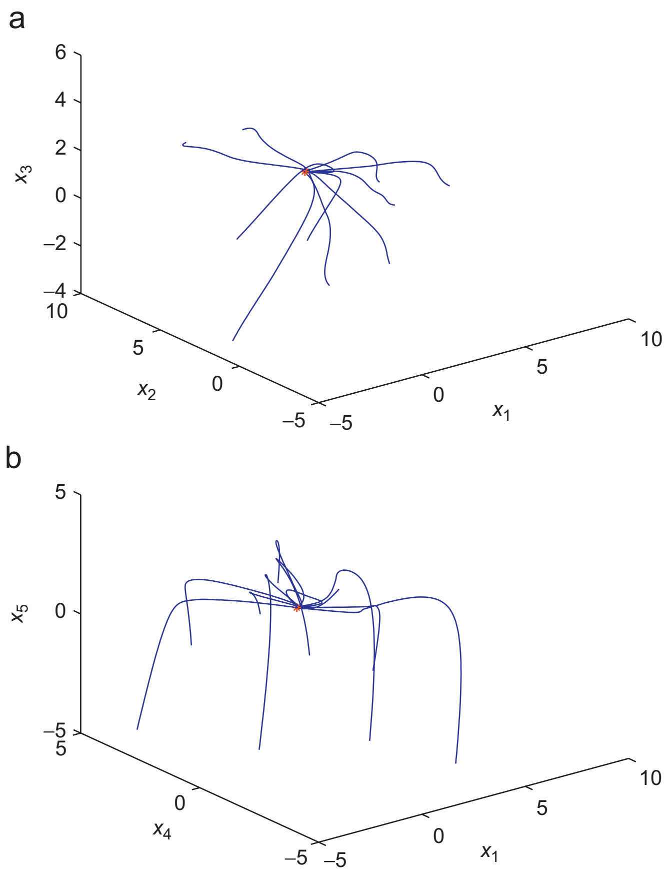


Fig. 11. Spatial trajectories of $x(t)$ synthesized by LVI-PDNN (5) starting randomly and converging finally to x^* of LP (1)–(4). (a) (x_1, x_2, x_3) convergence and (b) (x_1, x_4, x_5) convergence.

convergence guaranteed, when design parameter $\gamma = 10^7$, the average convergence time for achieving the accuracy of 10^{-3} is about 0.93 μs.

The above two illustrative examples have substantiated the efficacy of LVI-PDNN (5) and its MATLAB Simulink modeling for online solution of QP and LP problems in a unified manner.

5. Conclusions

Robot motion planning and control have motivated the online solution to general LP and QP problems [21]. In this paper, the LVI-based primal-dual neural network (LVI-PDNN) for such an online solution has been presented. Its Simulink modeling and simulative verification have been investigated as well. Instead of conventional compilation of program code, the LVI-PDNN Simulink model could be built up systematically and easily by using basic function blocks. Programming efforts are thus reduced drastically. Two illustrative examples with a large number of numerical experiments have substantiated the efficacy of the LVI-PDNN and its Simulink modeling. In addition, this LVI-PDNN Simulink modeling appears to be an important and necessary step to its FPGA and ASIC implementation, which could be done by using Simulink HDL Coder as a future research direction.

Acknowledgments

Before joining Sun Yat-Sen University in 2006, the corresponding author, Yunong Zhang, had been with National University of Ireland at Maynooth, University of Strathclyde, National University of Singapore, Chinese University of Hong Kong, since 1999. Continuing the line of this research, he has been supported by different research fellowships/assistantship. This research is funded by National Science Foundation of China under Grant 60775050 and by the Science and Technology Office of Sun Yat-Sen University.

References

[1] D. Anguita, A. Boni, S. Ridella, A digital architecture for support vector machines: theory, algorithm, and FPGA implementation, IEEE Trans. Neural Networks 14 (5) (2003) 993–1009.
[2] A. Cichocki, R. Unbehauen, Neural Networks for Optimization and Signal Processing, Wiley, Chichester, 1993.
[3] A.A.H. El-Shafei, M.I. Sobhy, A time-multiplexing simulator for cellular neural network (CNN) using SIMULINK, in: Proceedings of the IEEE International Symposium on Circuits and Systems, vol. 3, 1998, pp. 167–170.
[4] P. Ferreiraa, P. Ribeiroa, A. Antunesa, F.M. Dias, A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function, Neurocomputing 71 (2007) 71–77.
[5] H. Hikawa, A digital hardware pulse-mode neuron with piecewise linear activation function, IEEE Trans. Neural Networks 14 (5) (2003) 1028–1037.
[6] K. Iqbal, Y.C. Pai, Predicted region of stability for balance recovery: motion at the knee joint can improve termination of forward movement, J. Biomech. 33 (12) (2000) 1619–1627.
[7] M.P. Kennedy, L.O. Chua, Neural networks for nonlinear programming, IEEE Trans. Circuits Syst. 35 (5) (1988) 554–562.
[8] J. Lázaro, J. Arias, A. Astarloa, U. Bidarte, A. Zuloaga, Hardware architecture for a general regression neural network coprocessor, Neurocomputing 71 (2007) 78–87.
[9] C. Mead, Analog VLSI and Neural Systems, Addison-Wesley, Reading, 1989.
[10] I.B. Pyne, Linear programming on an electronic analogue computer, Trans. Am. Inst. Electrical Eng. 75 (1956) 139–143.
[11] M.A. Shanblatt, A Simulink-to-FPGA implementation tool for enhanced design flow, in: Proceedings of the IEEE International Conference on Microelectronic Systems Education, 2005, pp. 89–90.
[12] W.S. Tang, J. Wang, A recurrent neural network for minimum infinity-norm kinematic control of redundant manipulators with an improved problem formulation and reduced architecture complexity, IEEE Trans. Syst. Man Cybernet. 31 (1) (2001) 98–105.
[13] D. Tank, J. Hopfield, Simple neural optimization networks: an A/D converter, signal decision circuit, and a linear programming circuit, IEEE Trans. Circuits Syst. 33 (5) (1986) 533–541.
[14] The MathWorks Inc., Using Simulink, version 6.6, Natick, MA, 2007, available at ⟨http://www.mathworks.com/access/helpdesk/help/toolbox/simulink⟩.
[15] The MathWorks Inc., Simulink HDL Coder User's Guide, version 1.1, Natick, MA, 2007, available at ⟨http://www.mathworks.com/access/helpdesk/help/toolbox/slhdlcoder⟩.

[16] S. Zhang, A.G. Constantinides, Lagrange programming neural networks, IEEE Trans. Circuits Syst. 39 (7) (1992) 441–452.
[17] Y. Zhang, On the LVI-based primal-dual neural network for solving online linear and quadratic programming problems, in: Proceedings of the American Control Conference, 2005, pp. 1351–1356.
[18] Y. Zhang, Inverse-free computation for infinity-norm torque minimization of robot manipulators, Mechatronics 16 (2006) 177–184.
[19] Y. Zhang, A set of nonlinear equations and inequalities arising in robotics and its online solution via a primal neural network, Neurocomputing 70 (2006) 513–524.
[20] Y. Zhang, Towards piecewise-linear primal neural networks for optimization and redundant robotics, in: Proceedings of IEEE International Conference on Networking, Sensing and Control, 2006, pp. 374–379.
[21] Y. Zhang, S.S. Ge, T.H. Lee, A unified quadratic programming based dynamical system approach to joint torque optimization of physically constrained redundant manipulators, IEEE Trans. Syst. Man Cybernet. Part B 34 (5) (2004) 2126–2133.
[22] Y. Zhang, W.E. Leithead, Exploiting Hessian matrix and trust-region algorithm in hyperparameters estimation of Gaussian process, Appl. Math. Comput. 171 (2005) 1264–1281.
[23] Y. Zhang, J. Wang, A dual neural network for convex quadratic programming subject to linear equality and inequality constraints, Phys. Lett. A 298 (4) (2002) 271–278.
[24] Y. Zhang, J. Wang, Y. Xu, A dual neural network for bi-criteria kinematic control of redundant manipulators, IEEE Trans. Robot. Autom. 18 (6) (2002) 923–931.

Yunong Zhang is a professor at School of Information Science and Technology, Sun Yat-Sen University (SYSU), China. He received B.S., M.S., and Ph.D. degrees, respectively, from Huazhong University of Science and Technology (HUST), South China University of Technology (SCUT), and Chinese University of Hong Kong (CUHK) respectively in 1996, 1999, and 2003. Before joining SYSU in 2006, Yunong Zhang had been with National University of Ireland (NUI), University of Strathclyde, National University of Singapore (NUS) since 2003. His main research interests include neural networks and robotics.

Weimu Ma received his B.S. degree from Department of Mathematics, Sun Yat-Sen University, China in July 2006. He is now completing his M.S. degree at School of Information Science and Technology, Sun Yat-Sen University. His current research interests include neural networks, nonlinear systems, and machine learning.

Xiao-Dong Li is an associate professor at School of Information Science and Technology, Sun Yat-Sen University, China. He received the B.S. degree from Department of Mathematics, Shanxi Normal University, China, in 1987, and the M.S. degree from Eastern China Institute of Technology, China, in 1990. He received the Ph.D. degree from Department of Manufacturing Engineering and Engineering Management, City University of Hong Kong, China, in 2006.

Hong-Zhou Tan is a professor at School of Information Science and Technology, Sun Yat-Sen University, China. He received the Ph.D. degree jointly from City University of Hong Kong and South China University of Technology, China, in 1998. From 1999 to 2000, he was a research fellow at School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. From April 2000 to October 2000, he was a post-doctoral research fellow at Department of Mechanical and Industrial Engineering, University of Manitoba, Winnipeg, MB, Canada. His research interests are in the areas of control and communication systems, including system identification and fault diagnosis.

Ke Chen received his B.S. degree from School of Information Science and Technology, Sun Yat-Sen University, China in 2007. He is now pursuing his M.E. degree at School of Software, Sun Yat-Sen University. His current research interests include neural networks, robotics, nonlinear systems, and software design.