ELSEVIER

European Journal of Operational Research 93 (1996) 244-256

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

A new neural network for solving linear programming problems

A. Cichocki [a,*], R. Unbehauen [b], K. Weinzierl [b], R. Hölzel [b]

[a] *Frontier Research Program RIKEN, ABS Laboratory, Saitama 351-01, Wako, Japan*

[b] *University Erlangen-Nürnberg, Lehrstuhl für Allgemeine und Theoretische Elektrotechnik, Cauerstr. 7, 91058 Erlangen, Germany.*

Received July 1994; revised August 1995

Abstract

We propose and analyse a new class of neural network models for solving linear programming (LP) problems in real time. We introduce a novel energy function that transforms linear programming into a system of nonlinear differential equations. This system of differential equations can be solved on-line by a simplified low-cost analog neural network containing only one single artificial neuron with adaptive synaptic weights. The network architecture is suitable for currently available CMOS VLSI implementations. An important feature of the proposed neural network architecture is its flexibility and universality. The correctness and performance of the proposed neural network is illustrated by extensive computer simulation experiments.

Keywords: Linear programming; Stochastic gradient descent optimization; Neural networks; Parallel computing

1. Introduction

Linear programming (LP) plays an important role in many disciplines such as economics, strategic planning, combinatorial problems, operational research, etc. [9,11,12,14,18].

The LP problem was first solved by Danzig forty years ago [9]. The simplex method developed by him is still the most widely used numerical algorithm [11]. Although the simplex method is efficient and elegant it does not possess a property that becomes more and more desired in the last two decades: polynomial complexity. In fact in the simplex algorithm the number of arithmetical operations grew exponentially with the number of variables.

In 1984 Karmarkar published an algorithm which appears to be more efficient than the simplex method, especially when the problem size increases above some thousands of variables [14]. The simplex method is classified as an exterior-point method while Karmarkar's method is classified as an interior-point method [12,18]. A modern interior-point method outperforms the simplex method for large problems, and the most important and surprising characteristics of the interior-point method is that the number of iterations depends very little on the problem size [11,12,18].

The modern numerical algorithms are very efficient and useful in solving large LP problems, however, they do not lend themselves to problems which require solution in real time (on-line), i.e. in a time of the order of hundreds of microseconds.

* Corresponding author. E-mail: cia@hare.riken.go.jp

PII S0377-2217(96)00044-6

In some advanced problems such as robotics, satellite guidance, on-line parameter estimation in control, image reconstruction, engineering design etc. it is necessary to follow (track) a solution by slowly varying the constraints and/or cost functions [1–3,5,6,13,16,17,19,21,23–27]. One promising approach to solve optimization problems in real time is to use the neural network approach [6].

Many interesting approaches and techniques have been proposed to solve LP problems in real-time [1–3,5,6,10,13,15–17,19,20,22,21,23–27]. In fact in the last forty years researchers have proposed various dynamic solvers (analog computers) for constrained optimization problems. This approach was first proposed by Pyne in 1956 [20] and further developed by Dennis [10], Rybashov [22], Karpinskaya [15] and others. Recently, due to the renewed interest in neural networks, several new dynamic solvers using artificial neural network models have been developed, see e.g. Tank and Hopfield [23], Kennedy and Chua [16], Rodriquez-Vazquez et al. [21], Wang [24,25], Zak et al. [17,26], Cichocki and Unbehauen [6], and Cichocki and Bargiela [4].

All the dynamical solvers developed till now are based on standard optimization techniques (penalty or augmented Lagrange multiplier methods) and lead to Hopfield-like networks with a large number of processing units [1,2,16,17,21,23–27]. However, the practical VLSI implementation of Hopfield-like neural networks is still a difficult problem because of the complex connectivity between a large number of processing units. In fact, the wiring of a large number of processing units on a two-dimensional surface of a silicon wafer represents today a major bottleneck for VLSI CMOS implementation of such neural networks. Motivated by the desire to avoid or at least to alleviate this problem and to maximally simplify the neural network architecture we will propose a novel approach by formulation of a suitable energy function. This novel energy function enables us to design a simple, efficient and highly practical neural network with only one (single) adaptive processing unit (artificial neuron) with on chip learning capability.

In other words the primary objective of this paper is to present an alternative recurrent artificial neural network for solving LP problems. Possessing the same or similar dynamical properties as known dynamical solvers, the new neural network is essentially simpler in configuration and hence much easier to implement in VLSI technology.

2. Problem formulation

The linear programming (LP) problem can be expressed in a number of canonical forms.

We express it in the very general form: minimize the cost function

$$f(\boldsymbol{x}) = \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} = \sum_{j=1}^{n} c_j x_j, \tag{1}$$

subject to the linear constraints

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{2}$$

and

$$x_{j\,\min} \leqslant x_j \leqslant x_{j\,\max}, \tag{3}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $m < n$, $\operatorname{rank}\{\boldsymbol{A}\} = m$, $\boldsymbol{b} \in \mathbb{R}^m$.

We assume that the constraints are posed in the standard form in which all general constraints are equality constraints and the inequalities are simple lower and upper bounds on the variables x_j $(j = 1, 2, \ldots, n)$.

If the number of variables n were equal to the number of constraints m, equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ would be a set of simultaneous equations with (at most) a single solution. There would be no possibility of optimization. Normally n is significantly greater than m. A typical medium-size LP problem may have several thousand variables and a few thousand constraints.

Our main objective is to design a neural network based on an analog learning machine which will be able to find in real-time the optimal vector $\boldsymbol{x}^*$ that minimizes the cost function and simultaneously satisfies the constraints. This learning machine should perform on-line computation that requires little memory or data storage and requires no knowledge of if or when the input data (parameters of matrix $\boldsymbol{A}$ and/or vectors $\boldsymbol{b}$, $\boldsymbol{c}$) change.

3. Neural network models—standard approach—critical review

The mapping of a constrained optimization problem into an appropriate energy (cost) function is the

standard (commonly) applied approach in the design of neural networks [3,5]. In other words, in order to formulate the optimization problem (1)-(3) in terms of an ANN the key step is to construct an appropriate energy (cost) function $E_c(x)$ so that the lowest energy state will correspond to the desired estimate (optimal solution) x^*. The construction of a suitable energy function enables us to transform the minimization problem into a system of differential or difference equations on the basis of which we can design an associated ANN with appropriate connection weights (synaptic weights) and input excitations.

For the LP problem we can construct the general energy function on the basis of the penalty method [6],

$$\tilde{E}_c(x) = f(x) + \kappa \sum_{i=1}^{m} P[r_i(x)], \tag{4}$$

or

$$E_c(x) = \nu f(x) + \sum_{i=1}^{m} P[r_i(x)], \tag{5}$$

where $\kappa > 0$ denotes the penalty multiplier, $\nu \geqslant 0$ is the reciprocal penalty parameter, $f(x) = c^{\mathrm{T}}x$ is the cost function, $P(r_i)$ are the penalty function terms and $r_i(x)$ are the residuals (equality constraints) defined as (cf. Eq. 2)

$$r_i(x) = a_i^{\mathrm{T}}x - b_i = \sum_{j=1}^{n} a_{ij}x_j - b_i \tag{6}$$
$$(i = 1, 2, \ldots, m).$$

Exemplary penalty function terms for the equality constraints (2) can take one of the following forms [2,6]:

$$P(r) = \frac{1}{2}r^2 \quad \text{(quadratic)}, \tag{7a}$$

$$P(r) = \frac{1}{p}|r|^p \quad (p\text{-norm}), \tag{7b}$$

$$P(r) = \begin{cases} r^2/2 & \text{for } |r| \leqslant \beta \\ \beta|r| - \beta^2/2 & \text{for } |r| > \beta \end{cases}$$
(Huber's function), (7c)

$$P(r) = \beta^2 \ln\cosh\left(r/\beta\right) \quad \beta > 0$$
(logistic), (7d)

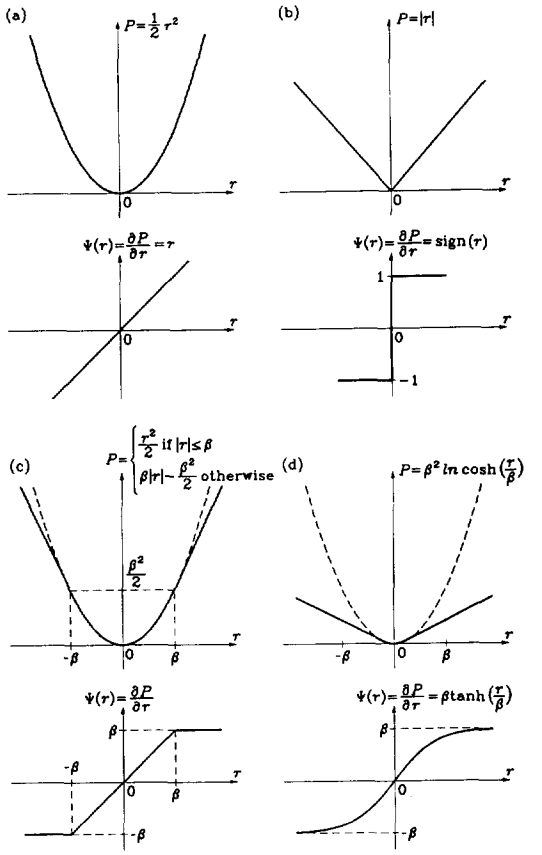

Fig. 1. Exemplary plots of some penalty functions and their derivatives.

$$P(r) = \frac{1}{p}|r|^p + \frac{1}{2}|r|^2 \quad \text{e.g. } p = \frac{9}{8}$$
(a combination penalty function). (7e)

Exemplary plots of the penalty functions and their derivatives are shown in Figs. 1a-d.

It should be noted here that instead of the penalty approach we can also employ the Lagrange multiplier or augmented Lagrange multiplier methods [5,6,27]. However, in order to streamline and simplify our further considerations we limit here our discussion to the penalty techniques.

It is known from the theory of optimization [26] that, except of trivial cases, only nondifferentiable penalty functions (see Fig. 1b) provide an exact solution of the original constrained optimization problem for a finite value of the penalty parameter κ in a single unconstrained minimization (cf. Eq. (4)). Usually, in

order to ensure a feasible solution satisfying exactly all the constraints the penalty parameter κ in Eq. (4) must tend to infinity. This is rather inconvenient from the implementation point of view. Therefore, we use Eq. (5) in which the parameter $\nu(t)$ should be gradually decreasing to zero as time goes to infinity [25]. Often a compromise is accepted by setting the parameter ν = constant to a sufficiently small value, so the obtained optimal solution can be very close to the exact unconstrained optimization problem [2,6,21].

Using the standard gradient descent approach [6] for the minimization of the energy function $E_c(\boldsymbol{x})$ the LP problem can be mapped to a nonlinear system of ordinary differential equations, i.e.

$$\frac{\mathrm{d}x_j}{\mathrm{d}t} = -\mu_j \frac{\partial E_c(\boldsymbol{x})}{\partial x_j}, \tag{8}$$

where $\mu_j > 0$ is the learning rate.

Hence taking into account Eqs. (1)-(3) and (5) we have

$$\frac{\mathrm{d}x_j}{\mathrm{d}t} = -\mu_j \left[\nu c_j + \sum_{i=1}^{m} a_{ij} \Psi(r_i) \right], \tag{9}$$

with $x_{j\,\min} \leqslant x_j \leqslant x_{j\,\max}\ \forall j\ (j = 1,2,\ldots,n)$, where $\Psi(r_i) \triangleq \partial P(r_i)/\partial r_i$ are the activation functions of the input neurons (cf. Fig. 1). The above system of differential equations can be written in the compact matrix form as

$$\dot{\boldsymbol{x}} = -\boldsymbol{\mu} \left[\nu \boldsymbol{c} + \boldsymbol{A}^{\mathrm{T}} \boldsymbol{\Psi}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \right], \tag{10}$$

where $\boldsymbol{\mu} = \mathrm{diag}\{\mu_1, \mu_2, \ldots, \mu_n\}$, $\boldsymbol{\Psi}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{\Psi}(\boldsymbol{r}) = [\Psi(r_1), \Psi(r_2), \ldots, \Psi(r_m)]^{\mathrm{T}}$.

On the basis of the set of differential equations (9), (10) we can easily construct the associated dynamic solver (ANN) with the suitable connection weights, the activation function Ψ and input excitations. The functional block diagram of the ANN is shown in Fig. 2a. The network consists of limiting integrators, adders (summing amplifiers) with associated connection weights a_{ij} and nonlinear building blocks realizing the activation function.

The network of Fig. 2a consists of two layers of processing units. The first layer computes the actual residuals $r_i(\boldsymbol{x})$ and actual errors $\Psi[r_i(\boldsymbol{x})]$, while the desired variables x_j are computed in the second layer, where the errors $\Psi(r_i)$ are combined and integrated in time by analog integrators.

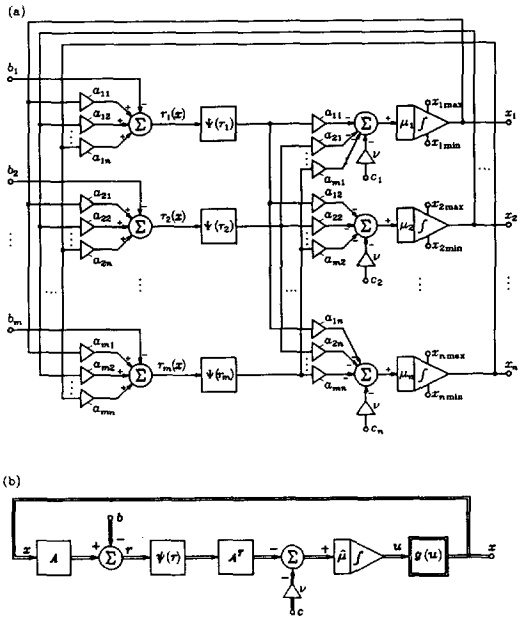

Fig. 2. ANN's solving the LP problems on the basis of the penalty method. Fig. 2a shows a network where the original values x_j are chosen as state-variables and, therefore, limiting integrators are required. In Fig. 2b the transformed states u_j are used to avoid limiting integrators. For a general view, this figure is represented in compact matrix form.

It should be noted that the simple box (bounds) constraints $x_{j\,\min} \leqslant x_j \leqslant x_{j\,\max}$ can be fulfilled by employing limiting integrators with nonlinear (hardware) limiters at their outputs. This means that the input signals of an integrator are integrated but cannot drive the output x_j beyond the specified limits. In such an approach all box constraints are "hard", i.e. the constraints must not be violated either at the final solution or during the optimization process.

An alternative approach is to introduce unlimited variables u_j that provide the nonlinear transformations

$$x_j = g_j(u_j), \tag{11}$$

e.g.

$$x_j = x_{j\,\min} + \frac{x_{j\,\max} - x_{j\,\min}}{1 + \mathrm{e}^{-\gamma u_j}} \quad \text{with } j = 1,2,\ldots,n, \tag{12}$$

where $\gamma > 0$.

Substituting (11) into (5) we obtain the new energy function without any constraints imposed on the

variables u_j:

$$E_c(\boldsymbol{u}) = \nu \sum_{j=1}^{n} c_j g_j(u_j) + \sum_{i=1}^{m} P(r_i), \tag{13}$$

where $r_i = \sum_{j=1}^{n} a_{ij} g_j(u_j) - b_i$.

Minimizing the above energy function we obtain

$$\frac{\mathrm{d}u_j}{\mathrm{d}t} = -\mu_j \frac{\partial E_c}{\partial u_j} = -\mu_j \frac{\partial E_c}{\partial x_j} \frac{\mathrm{d}x_j}{\mathrm{d}u_j}$$

$$= -\mu_j \left[\nu c_j + \sum_{i=1}^{m} a_{ij} \Psi(r_i) \right] \frac{\mathrm{d}g_j(u_j)}{\mathrm{d}u_j}. \tag{14}$$

Assuming that the activation functions $x_j = g_j(u_j)$ are differentiable and strictly monotonically increasing we note that $\mathrm{d}g_j(u_j)/\mathrm{d}u_j > 0\ \forall j$ and $\forall u_j$, hence we can write

$$\frac{\mathrm{d}u_j}{\mathrm{d}t} = -\hat{\mu}_j(t) \left[\nu c_j + \sum_{i=1}^{m} a_{ij} \Psi[r_i(\boldsymbol{x})] \right], \tag{15}$$

$$x_j = g_j(u_j), \tag{16}$$

where $\hat{\mu}_j \triangleq \mu_j dg_j(u_j)/du_j > 0$ is the learning rate, or in matrix form as (cf. Fig. 2b)

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = -\hat{\boldsymbol{\mu}} \left[\nu \boldsymbol{c} + \boldsymbol{A}^{\mathrm{T}} \boldsymbol{\Psi}(\boldsymbol{r}) \right], \tag{17}$$

$$\boldsymbol{x} = \boldsymbol{g}(\boldsymbol{u}), \tag{18}$$

where $\hat{\boldsymbol{\mu}} = \mathrm{diag}\{\hat{\mu}_1, \hat{\mu}_2, \ldots, \hat{\mu}_n\}$, $\boldsymbol{g}(\boldsymbol{u}) = [g_1(u_1), g_2(u_2), \ldots, g_n(u_n)]^{\mathrm{T}}$, $\boldsymbol{r} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$.

It should be noted that due to employing appropriate limiting activation functions in the output layer the satisfaction of the bound constraints is ensured.

It is interesting to note that the general architectures shown in Figs. 2a, b can be somewhat simplified for the special case of a quadratic penalty function. In this case an energy function can be expressed as

$$E_c(\boldsymbol{x}) = \nu \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x} + \tfrac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$$

$$= \nu \sum_{j=1}^{n} c_j x_j + \tfrac{1}{2} \sum_{i=1}^{m} r_i^2(\boldsymbol{x}), \tag{19}$$

with $x_{j\,\min} \leqslant x_j \leqslant x_{j\,\max}$.

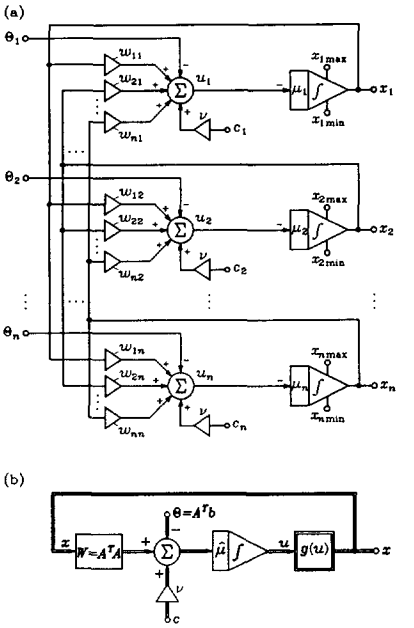

Fig. 3. The ANN's of Fig. 2 can be simplified, if a quadratic penalty function has been chosen. This networks consist of only one layer of processing units, but they cannot be used, if the matrix $\boldsymbol{A}$ and/or the vector $\boldsymbol{b}$ is time variable.

Minimizing the energy function (19) leads to a system of differential equations

$$\frac{\mathrm{d}x_j}{\mathrm{d}t} = -\mu_j \left[\nu c_j + \sum_{k=1}^{n} w_{kj} x_k - \Theta_j \right], \tag{20}$$

where $w_{kj} = \sum_{i=1}^{m} a_{ik} a_{ij}$, $\Theta_j = \sum_{i=1}^{m} a_{ij} b_i$, which can be written in compact matrix form as

$$\dot{\boldsymbol{x}} = -\boldsymbol{\mu}[\nu \boldsymbol{c} + \boldsymbol{W}\boldsymbol{x} - \boldsymbol{\Theta}], \tag{21}$$

where $\boldsymbol{W} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and $\boldsymbol{\Theta} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$.

A functional block diagram illustrating the implementation of the system of differential equations (20) is shown in Fig. 3a.

Alternatively, we can use a system of equations (cf. Fig. 3b)

$$\dot{\boldsymbol{u}} = -\hat{\boldsymbol{\mu}}[\nu \boldsymbol{c} + \boldsymbol{W}\boldsymbol{x} - \boldsymbol{\Theta}], \tag{22}$$

$$\boldsymbol{x} = \boldsymbol{g}(\boldsymbol{u}). \tag{23}$$

This is a Hopfield-type analog neural network with only one layer of processing units. However, a single

layer ANN requires extra precalculations and therefore, it is rather inconvenient for large matrices especially when the entries a_{ij} and/or b_i are slowly changing in time (i.e. they are time variable).

The techniques described above are rather simple and straightforward, however, some problems may arise in the practical implementation of the systems of differential equations, especially, if the matrix $\boldsymbol{A}$ is very large. Firstly, the VLSI implementations of the neural network architectures shown in Figs. 2a, b and Figs. 3a, b are rather a difficult problem because of the complex connectivity between a large number of processing units. Secondly, the neural networks of Figs. 2a, b and Figs. 3a, b require an extremely large number of programmable (adjustable) and precise synaptic weights a_{ij} or w_{ij}. In fact, the network of Figs. 2a, b requires in general $2mn$ precise programmable connection weights, while the network of Figs. 3a, b needs mn such weights. The connection weights may be realized as rather expensive analog four-quadrant multipliers. Thirdly, analog VLSI neural circuits are strongly influenced by device mismatches from the fabriation process and a variety of parasitic effects which consequently may degrade the final performance (accuracy).

Motivated by the desire to maximally simplify the neural network architecture and alleviate the problems mentioned above we will propose a novel approach in the next section which enables us to develop a considerably simplified neural network more suitable for VLSI implementations.

4. Simplified neural network model—novel approach

To solve the LP problem (1)–(3) by an appropriate ANN the key step is to construct a suitable computational energy function. For this purpose we have developed the following instantaneous error (penalty) function

$$\tilde{r}[\boldsymbol{x}(t)] \triangleq \boldsymbol{s}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) = \sum_{i=1}^{m} s_i(t) r_i[\boldsymbol{x}(t)], \quad (24)$$

where $r_i(\boldsymbol{x}) = \sum_{j=1}^{n} a_{ij}x_j - b_i$ and $\boldsymbol{s} = [s_1(t), s_2(t), \ldots, s_m(t)]^{\mathrm{T}}$ is in general the set of zero-mean, mutually independent (or uncorrelated) identically distributed (i.i.d.) external excitation signals (e.g. uncorrelated high frequency or pseudo random signals). Usually a high frequency of such signals is required to achieve high speed convergence.

Note that the value of the error (residuum) function $\tilde{r}[\boldsymbol{x}(t)]$ is equal to zero at any time instant (or during any time period) if and only if the constraints $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ are satisfied exactly.

The instantaneous error function can be developed as

$$\begin{aligned}\tilde{r}[\boldsymbol{x}(t)] &= \sum_{i=1}^{m} s_i(t) r_i[\boldsymbol{x}(t)] \\ &= \sum_{i=1}^{m} s_i(t) \left[\sum_{j=1}^{n} a_{ij}x_j(t) - b_i\right] \\ &= \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij}s_i(t)\right) x_j(t) - \sum_{i=1}^{m} b_i s_i(t) \\ &= \sum_{j=1}^{n} \tilde{a}_j(t)x_j(t) - \tilde{b}(t),\end{aligned}$$

where $\tilde{a}_j(t) \triangleq \sum_{i=1}^{m} a_{ij}s_i(t)$, $\tilde{b}(t) \triangleq \sum_{i=1}^{m} b_i s_i(t)$.

For the so formulated instantaneous error function $\tilde{r}[\boldsymbol{x}(t)]$, we can construct the energy (cost) function

$$E_c[\boldsymbol{x}(t)] = \nu \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + E\{P[\tilde{r}(\boldsymbol{x}(t))]\} \quad (25)$$

with $\boldsymbol{x}_{\min} \leqslant \boldsymbol{x} \leqslant \boldsymbol{x}_{\max}$ where $E\{\cdot\}$ is the expected value of its argument, and $P(\tilde{r})$ is the penalty function defined for example by one of the equations (7a)–(7e). The minimization of the energy function (25) with respect to the vector $\boldsymbol{x}(t)$ by using the standard gradient descent method leads to the system of differential equations

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = -\boldsymbol{\mu}\left[\nu\boldsymbol{c} + E\{\tilde{\boldsymbol{a}}(t)\Psi[\tilde{r}(\boldsymbol{x}(t))]\}\right], \quad (26)$$

with $\boldsymbol{x}_{\min} \leqslant \boldsymbol{x} \leqslant \boldsymbol{x}_{\max}$, where $\boldsymbol{\mu} = \mathrm{diag}\{\mu_1, \mu_2, \ldots, \mu_n\}$, $\mu_j > 0$ $\forall j$ (typically $\mu_j = \mu > 0$ $\forall j$), $\tilde{\boldsymbol{a}}(t) = [\tilde{a}_1(t), \tilde{a}_2(t), \ldots, \tilde{a}_n(t)]^{\mathrm{T}} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{s}(t)$, and $\Psi[\tilde{r}] \triangleq \partial P[\tilde{r}]/\partial\tilde{r}$, $\tilde{r}[\boldsymbol{x}(t)] = \sum_{j=1}^{n} \tilde{a}_j(t)x_j(t) - \tilde{b}(t)$.

In practice, the expected values of the vector $\boldsymbol{p}_c(t) \triangleq \tilde{\boldsymbol{a}}(t)\Psi[\tilde{r}(\boldsymbol{x}(t))]$ is not available and their computation is rather difficult. In fact, the instantaneous gradient based on the instantaneous error func-

tion (24) can be used in practice since it is readily obtained.

So, the system of differential equations (25) in our case is approximated by

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = -\boldsymbol{\mu}\{\nu\boldsymbol{c} + \tilde{\boldsymbol{a}}(t)\Psi[\tilde{r}(\boldsymbol{x}(t))]\}, \tag{27}$$

with $\boldsymbol{x}_{\min} \leqslant \boldsymbol{x} \leqslant \boldsymbol{x}_{\max}$.

Equivalently we can use the system of differential equations

$$\frac{\mathrm{d}\boldsymbol{u}(t)}{\mathrm{d}t} = -\tilde{\boldsymbol{\mu}}\{\nu\boldsymbol{c} + \tilde{\boldsymbol{a}}(t)\Psi[\tilde{r}(\boldsymbol{x}(t))]\}, \tag{28}$$

$$\boldsymbol{x}(t) = \boldsymbol{g}[\boldsymbol{u}(t)], \tag{29}$$

where $\boldsymbol{g}(\boldsymbol{u})$ is a vector of sigmoid activation functions (cf. Eqs. (11), (12)) which are bounded between $\boldsymbol{x}_{j\,\min}$ and $\boldsymbol{x}_{j\,\max}$ and $\tilde{\boldsymbol{\mu}} = \mathrm{diag}\{\mu_1\partial g_1/\partial u_1, \mu_2\partial g_2/\partial u_2 \ldots, \mu_n\partial g_n/\partial u_n\}$ is a diagonal matrix containing the transformed learning rates.

It is interesting to note that in the special case of $P(\tilde{r}) = \frac{1}{2}\tilde{r}^2$ the system of differential equations (27) simplifies to

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = -\boldsymbol{\mu}[\nu\boldsymbol{c} + \tilde{\boldsymbol{a}}(t)\tilde{r}[\boldsymbol{x}(t)]], \tag{30}$$

with $\boldsymbol{x}_{\min} \leqslant \boldsymbol{x} \leqslant \boldsymbol{x}_{\max}$.

The above system of differential equations can be considered as the family of adaptive learning algorithms of a single artificial neuron as shown in Fig. 4a. The network of Fig. 4a consists of analog limiting integrators, summers, an activation function $\Psi(\tilde{r})$ and analog four quadrant multipiers. The network is driven by the incoming data stream a_{ij} and b_j $(i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n)$ modulated (multiplied) by high frequency, zero-mean mutually uncorrelated source signals $s_i(t)$. The artificial neuron (processing unit) shown in Fig. 4a with an on-chip adaptive learning algorithm allows processing of the information fully simultaneously.

If only one pseudo-random generator is available in order to approximate m independent identically distributed excitation signals $s_i(t)$ a chain of unit delays can be employed as shown in Fig. 4b.

In order to further simplify the network implementation shown in Fig. 4a we have found that the rather expensive analog multipliers can be replaced by the simple switches S_1 to S_m (or sign reversers) as shown in Fig. 5a.

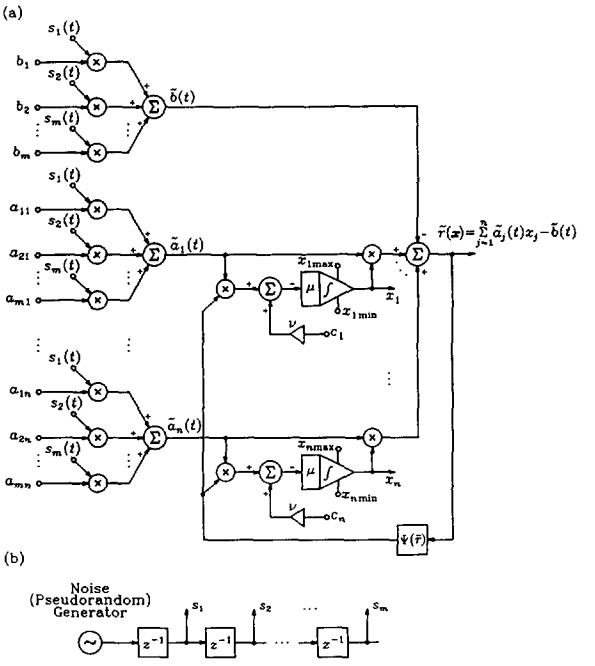

Fig. 4. The ANN of Fig. 2a has been simplified to a family of adaptive learning algorithms of a single artificial neuron. Here the realization of the simplified ANN with limiting integrators, adders, multipliers etc. is shown. The stochastic processes $s_j(t)$ are approximated by a single Noise Pseudorandom Generator and a chain of delayers, shown in Fig. 4b. If the approximation of mutually independent identically distributed white stochastic processes were exactly, the dynamical properties in the special case of a quadratic penalty function would be the same of the ANN described in Fig. 2a.

Various strategies for controlling the switches can be chosen. In the simplest strategy the switches can be controlled by a multiphase clock, i.e. the switches will be closed and opened cyclically. In this case the network processes the set of equations (equality constraints) in a cyclical order similarly to the well known Kaczmarz algorithm used for solving large unstructured linear equations [6].

On the other hand, in order to perform a fully simultaneous processing of all the constraints $\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} - b_i = 0$ $(i = 1, 2, \ldots, m)$ the switches S_1 to S_m should be controlled by a digital generator producing multiple pseudo-random, uncorrelated bit streams.

As such a generator for example, a simple feedback shift register can be used, which is able to generate

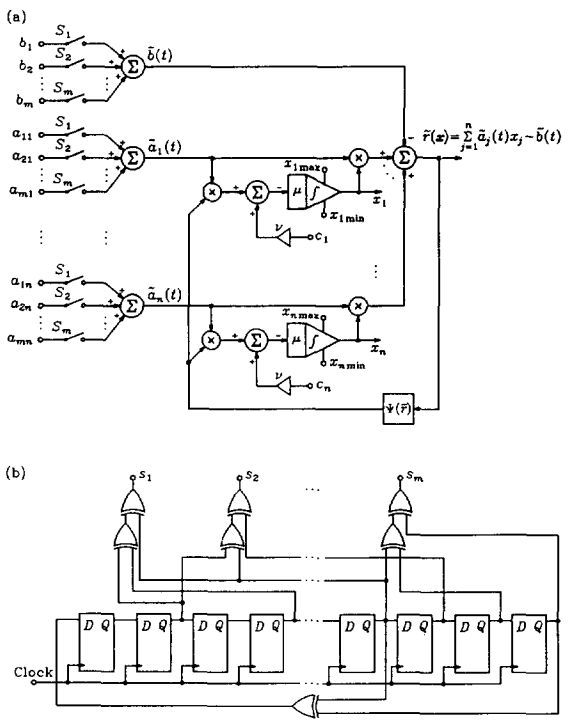

Fig. 5. Fig. 5a shows a further simplification of the ANN, described by Fig. 4a. The four quadrant analog multipliers of the inputs have been exchanged by switches. The switches are driven by the network depicted in Fig. 5b.

almost uncorrelated multiple mutually shifted pseudo-random bit streams with very good noise-like properties (see Fig. 5b) [6].

5. Discussion—convergence and stability analysis

At the beginning we will show that under some mild assumptions the new neural network is principally equivalent to the standard networks shown in Figs. 2a and 3a (or more precisely the energy functions of these networks are equivalent).

For simplicity, let us assume that the penalty function term is a quadratic function of the instantaneous error (residuum), i.e.

$$P[\tilde{r}(\boldsymbol{x}(t))] = \tfrac{1}{2}\tilde{r}^2(\boldsymbol{x}(t)) = \tfrac{1}{2}\left[s^{\mathrm{T}}(t)(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})\right]^2 = \tfrac{1}{2}(s^{\mathrm{T}}\boldsymbol{r})^2. \tag{31}$$

Then for the LP problem we can formulate the energy function

$$E_c[\boldsymbol{x}(t)] = \nu\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \tfrac{1}{2}E\{\tilde{r}^2[\boldsymbol{x}(t)]\}, \tag{32}$$

which can be evaluated as

$$E_c(\boldsymbol{x}) = \nu\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \tfrac{1}{2}\boldsymbol{r}^{\mathrm{T}}E\{\boldsymbol{s}\boldsymbol{s}^{\mathrm{T}}\}\boldsymbol{r} = \nu\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \tfrac{1}{2}\boldsymbol{r}^{\mathrm{T}}\boldsymbol{R}_{ss}\boldsymbol{r}, \tag{33}$$

where $\boldsymbol{R}_{ss} = E\{\boldsymbol{s}\boldsymbol{s}^{\mathrm{T}}\}$ is the correlation matrix of the vector $s(t)$. Assuming that $s(t)$ is a zero-mean i.i.d. white and mutually independent stochastic process the correlation matrix $\boldsymbol{R}_{ss}$ is a diagonal matrix with all diagonal elements equal to the variance $\sigma^2 \triangleq E\{s_i^2\}$ $\forall i$.

Hence the energy function (33) can be expressed as

$$E(\boldsymbol{x}) = \nu\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \tfrac{1}{2}\sigma^2\|\boldsymbol{r}\|_2^2. \tag{34}$$

Thus the above energy function is equivalent to the standard energy function (5) for quadratic penalty, assuming that the excitation signals $s_i(t)$ are zero-mean and have the unity variance ($\sigma^2 = 1$).

Let us consider now the more practical case (cf. Fig. 4a) in which the random uncorrelated identically distributed excitation signals $s_i(t)$ take only two discrete values 0 and 1 (a switch is OFF or ON). In this case the exitation signals have no longer zero-mean values. Then the stochastic process $s(t)$ can be decomposed as

$$\boldsymbol{s}(t) = \boldsymbol{s}_v(t) + \boldsymbol{s}_c, \tag{35}$$

where

$$\boldsymbol{s}_v(t) = [s_{v1}(t), s_{v2}(t), \ldots, s_{vn}(t)]^{\mathrm{T}}, \tag{36}$$

with $s_{vi}(t) \in \{-\frac{1}{2}, \frac{1}{2}\}$, is a zero-mean uncorrelated identically distributed process and

$$\boldsymbol{s}_c = s_c[1, 1, \ldots, 1]^{\mathrm{T}} \quad \text{with } s_c = \tfrac{1}{2} \tag{37}$$

is a constant process.

For this case the energy function (32) can be evaluated as

$$\begin{aligned} E_c(\boldsymbol{x}) &= \nu\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \tfrac{1}{2}E\left\{\left[(\boldsymbol{s}_v + \boldsymbol{s}_c)^{\mathrm{T}}\boldsymbol{r}\right]^2\right\} \\ &= \nu\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ &\quad + \tfrac{1}{2}E\left\{\boldsymbol{r}^{\mathrm{T}}(\boldsymbol{s}_v\boldsymbol{s}_v^{\mathrm{T}} + \boldsymbol{s}_v\boldsymbol{s}_c^{\mathrm{T}} + \boldsymbol{s}_c\boldsymbol{s}_v^{\mathrm{T}} + \boldsymbol{s}_c\boldsymbol{s}_c^{\mathrm{T}})\boldsymbol{r}\right\} \\ &= \nu\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \tfrac{1}{2}\left\{\boldsymbol{r}^{\mathrm{T}}[E\{\boldsymbol{s}_v\boldsymbol{s}_v^{\mathrm{T}}\} + \boldsymbol{s}_c\boldsymbol{s}_c^{\mathrm{T}}]\boldsymbol{r}\right\} \\ &= \nu\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \tfrac{1}{2}\boldsymbol{r}^{\mathrm{T}}(\sigma^2\boldsymbol{I} + \tfrac{1}{4}\boldsymbol{1})\boldsymbol{r}, \end{aligned}$$

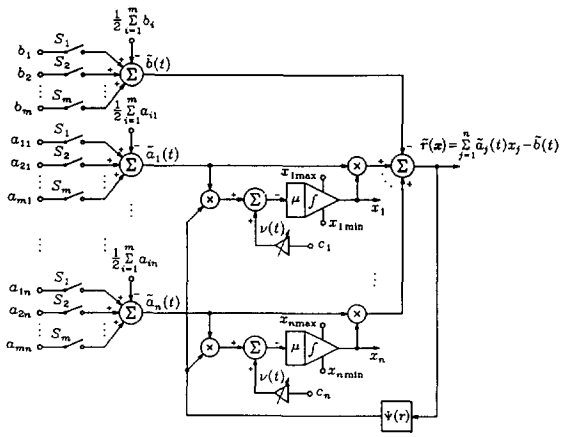

Fig. 6. If the mean values over the coefficients b_j or a_{ij} are supplemented to the network of Fig. 5a, like shown in the figure here, the dynamical properties are equal to the network depicted in Fig. 2a for the case of a quadratic penalty function.

where $\boldsymbol{I}$ is the unit matrix and $\mathbf{1}$ is the $n \times n$ constant matrix with all entries equal to one.

In this case the system of differential equations (30) takes the form

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} &= -\boldsymbol{\mu}\left[\nu\boldsymbol{c} + \boldsymbol{A}^{\mathrm{T}}(\sigma^2\boldsymbol{I} + \tfrac{1}{4}\mathbf{1})(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})\right] \\ &= -\boldsymbol{\mu}\left[\nu\boldsymbol{c} + \boldsymbol{A}^{\mathrm{T}}(\sigma^2\boldsymbol{I} + \tfrac{1}{4}\mathbf{1})\boldsymbol{A}\boldsymbol{x} \right. \\ &\qquad \left. - \boldsymbol{A}^{\mathrm{T}}(\sigma^2\boldsymbol{I} + \tfrac{1}{4}\mathbf{1})\boldsymbol{b}\right].\end{aligned}$$

Note that the matrix $[\sigma^2\boldsymbol{I} + \frac{1}{4}\mathbf{1}]$ is symmetric positive definite and consequently the matrix $\boldsymbol{A}^{\mathrm{T}}(\sigma^2\boldsymbol{I} + \frac{1}{4}\mathbf{1})\boldsymbol{A}$ is positive semidefinite, so the system is stable with $\nu(t) \to 0$ as $t \to \infty$. However, the dynamics of the system are not identical with the dynamics of the system described by (20), (21).

In order to ensure such an equivalence it is necessary to extract appropriate constants (local mean values) as illustrated in Fig. 6. In such a case the preprocessed signals can be expressed as

$$\tilde{a}_j(t) = \sum_{i=1}^{m} a_{ij}\hat{s}_i(t) \quad (j = 1, 2, \ldots, n) \tag{38}$$

and

$$\tilde{b}(t) = \sum_{i=1}^{m} b_i\hat{s}_i(t), \tag{39}$$

where $\hat{s}_i(t) = s_i(t) - \gamma$, $0 \leqslant \gamma \leqslant 1/2$ and the signals $s_i(t)$ can take only one of the two discrete (binary) values 0 or 1. The values of the parameter γ depend on the strategy employed for controlling the switches. If all the switches are operating fully parallel the parameter γ should be set to $1/2$.

It should be noted that the proposed learning algorithm has been developed on the basis of the standard gradient descent method, therefore the algorithm is always stable independent of initial conditions [6,16].

6. Computer simulation results

In order to check the correctness, robustness and performance of the proposed algorithm and associated neural network structures we have simulated them extensively on a computer [13]. Due to limited space we shall present in this paper only some illustrative examples.

In all our computer simulation experiments the ideal integrators with the transfer function $G(s) = \mu/s$ were replaced by a realistic model for the high frequency range as

$$G(s) = \mu_0\frac{\omega_T}{s}\frac{1}{1 + 1.7sT + s^2T^2},$$

where $\omega_T = 10^8$ is the gain bandwidth product and $T = 5 \cdot 10^{-11}$ were chosen. The value for T can be found according to the phase margin at 60° typical for real operational amplifiers [13].

All the computer simulation results presented in this paper have been achieved by using a general program for simulation of a wide class of nonlinear dynamical systems (developed at Lehrstuhl für Allgemeine und Theoretische Elektrotechnik at University Erlangen-Nuernberg). The program is similar in performance to the well known SIMULINK/MATLAB program.

Example 1.

Consider an LP example as

minimize $f(\boldsymbol{x}) = \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$

subject to

$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x} > \mathbf{0}$,

where

$$\boldsymbol{c} = [1,1,1,1,1,1]^{\mathrm{T}},$$

$$\boldsymbol{b} = [2,1,-4]^{\mathrm{T}},$$

$$\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 4 & 0 & 3 & 1 \\ 5 & 1 & -3 & 1 & 2 & 0 \\ 1 & -2 & 1 & -5 & -1 & 4 \end{bmatrix}.$$

The optimal solution of this problem is $\boldsymbol{x}_T^* = [0,0,0.1923,0.7564,0.4103,0]^{\mathrm{T}}$, the minimal objective function value is $f(\boldsymbol{x}_T^*) = 1.359$. For the neural network of Fig. 6 we used the following parameters: $\mu_0 = 0.1$, $\nu = 10^{-3}$, $T = 5 \cdot 10^{-11}$ and the clock frequency $f_c = 100\,\mathrm{MHz}$. The chosen value for μ_0 corresponds to a learning rate $\mu = \omega_T \mu_0 = 10^7\,\mathrm{s}^{-1}$. The simulation results are shown in Figs. 7a, b. The network was able to find the solution:

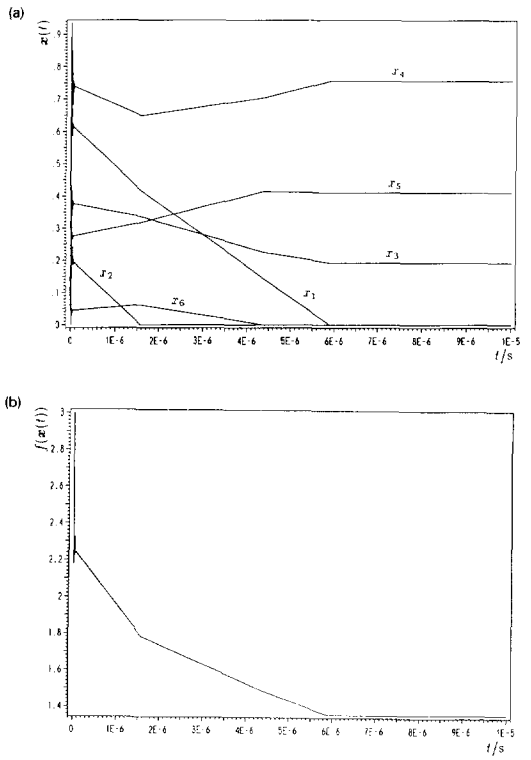

Fig. 7. (a) The ANN of Fig. 6 finds the optimal solution $\boldsymbol{x}^*$ of Example 1 with an accuracy of $\leqslant 0.1\%$ in time less than $10\,\mu\mathrm{s}$. The simulation results takes real integrators with a transit frequency $f_T = 10^8/2\pi\,\mathrm{s}^{-1}$ and a phase margin of 60° at f_T as a basis. Real integrators are assumed for all further simulation results, respectively. The clock frequency for the switches is 100 MHz. (b) The objective function value appertaining to the simulation of Example 1.

$$\boldsymbol{x}^* = [0,0,0.19224,0.75635,0.41021,0]^{\mathrm{T}}$$

in less than $10\,\mu\mathrm{s}$. The small parasitic oscillations observed in the first phase of the simulation are caused by the above mentioned nonidealities of the real integrators. They can be eliminated by choosing a little smaller value for μ_0.

Example 2. Let us consider the following problem [26]. The circuit shown in Fig. 8 should be designed to use a 30 V source to charge 10 V, 6 V and 20 V batteries connected parallel. The currents I_1, I_2, I_3, I_4 and I_5 are limited to the maximum of 4 A, 3 A, 2 A, 3 A and 2 A, respectively. The batteries may not be discharged, i.e. all currents must be nonnegative. The problem is to find the optimal values of the currents such that the total power transferred to the batteries is maximized. The problems can be equivalently expressed as

maximize the power:

$$p(\boldsymbol{x}) = 10x_2 + 6x_4 + 20x_5,$$

subject to the constraints

$$x_1 = x_2 + x_3, \quad x_3 = x_4 + x_5, \quad 0 \leqslant x_1 \leqslant 4,$$
$$0 \leqslant x_2 \leqslant 3, \quad 0 \leqslant x_3 \leqslant 3, \quad 0 \leqslant x_4 \leqslant 2,$$
$$0 \leqslant x_5 \leqslant 2,$$

where $x_j \equiv I_j \quad \forall j\ (j = 1,2,\ldots,5)$.

The problem can be transformed to the standard form as

minimize $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$

subject to

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \quad \text{and} \quad \boldsymbol{x} \geqslant \boldsymbol{0},$$

where

$$\boldsymbol{x} \in \mathbb{R}^{10},$$

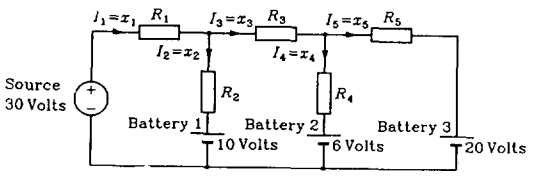

Fig. 8. Circuit of Example 2 to charge three batteries with a single battery charger.

$$\boldsymbol{c} = [0, -10, 0, -6, -20, 0, 0, 0, 0, 0]^{\mathrm{T}},$$

$$\boldsymbol{b} = [0, 0, 4, 3, 3, 2, 2]^{\mathrm{T}},$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Alternatively, the problem can also be formulated as minimize

$$f_1(\boldsymbol{x}) = -10x_2 - 6x_4 - 20x_5$$

subject to the constraints

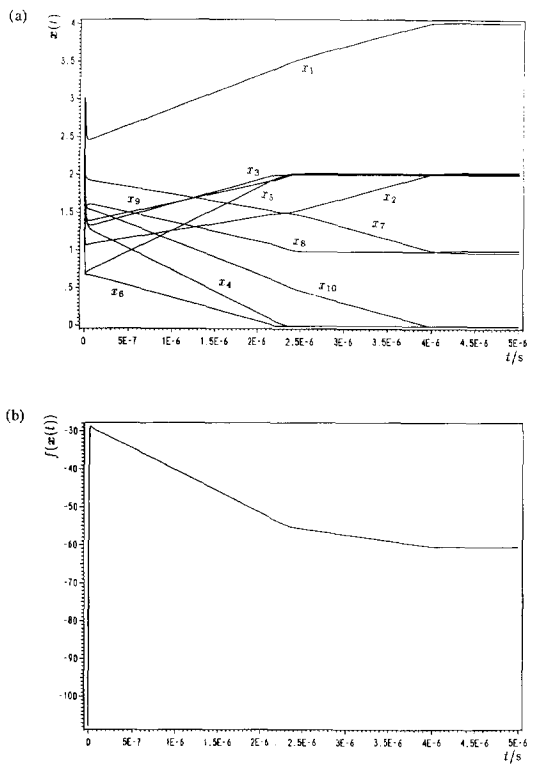

Fig. 9. Simulation results for Example 2: Fig. 9a shows the state-variables $x_j (1 \leqslant j \leqslant 10)$, Fig. 9b the plot objective function.

$$\begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$0 \leqslant x_1 \leqslant 4, \quad 0 \leqslant x_2 \leqslant 3, \quad 0 \leqslant x_3 \leqslant 3,$$
$$0 \leqslant x_4 \leqslant 2, \quad 0 \leqslant x_5 \leqslant 2.$$

One can easily check that the solution of the problem in standard form is

$$\boldsymbol{x}_T^* = [4, 2, 2, 0, 2, 0, 1, 1, 2, 0].$$

In Figs. 9a, b the transient behaviour of the neural network is depicted with initial conductions $x_j(0) = 3\ \forall j$, $\mu_0 = 0.1$, $\nu = 0.001$, $T_0 = 5 \cdot 10^{-11}$, $f_c = 100\,\mathrm{MHz}$. The network finds an optimal solution

$$\boldsymbol{x}^* \cong [4.0146, 2.0298, 1.9998, 0, 2.0152, 0, 0.9705, 1.0002, 2, 0]$$

and

$$f(\boldsymbol{x}^*) = -60.6$$

in less than 5 μs.

Example 3. Let us consider the following very ill conditioned problem: find the vector $\boldsymbol{x} \in \mathbb{R}^n$ which satisfies the matrix equation

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is the Hilbert matrix with $a_{ij} = 1/(i+j-1)$, $b_i = \sum_{j=1}^{n} \mathrm{e}^{1/j} a_{ij}$, $\mathrm{e} = 2.718281828$ and $\boldsymbol{c} \equiv \boldsymbol{0}$.

The theoretical solution is

$$\boldsymbol{x}_T^* = \left[\mathrm{e}, \mathrm{e}^{1/2}, \ldots, \mathrm{e}^{1/n}\right]^{\mathrm{T}}.$$

Using the neural network of Fig. 5a we found the solution for $n = 10$, $\nu = 0$, $\mu_0 = 0.1$, $T = 5 \cdot 10^{-11}$

$$x^* = \begin{bmatrix} 2.71723 \\ 1.65009 \\ 1.40740 \\ 1.28416 \\ 1.21001 \\ 1.16553 \\ 1.14085 \\ 1.12750 \\ 1.12790 \\ 1.13633 \end{bmatrix},$$

which is close to the theoretical solution (cf. Fig. 10) in a time of 1 ms.

7. Conclusion

A new, very simple and low-cost analog neural network architecture for solving LP problems has been proposed. The network architecture is suitable for currently available CMOS VLSI implementations. The proposed network consists of only one (single) neuron with adaptive synaptic weights and a simple preprocessing circuit. The synaptic weights are adjusted (updated) according to a simple learning algorithm. The continuous-time (analog) formalism employed in the proposed algorithm (in fact the basic learning algorithm is expressed completely by a system of nonlinear differential equations) enable us to select a very high learning rate $\mu(t)$ (to ensure an extremely high computation speed) without affecting the stability of the network. In contrast, for the associated discrete-time iterative scheme (using, e.g. Euler's rule) the corresponding learning rate must be upper bounded in a small range or otherwise the network will be unstable (i.e. the learning algorithm will diverge).

An interesting and important feature of the proposed algorithmic scheme is its universality and flexibility.

It allows either a processing of all equality constraints fully simultaneously in time or a processing of groups of constraints called blocks. These blocks of constraints need not be fixed but may vary rather dynamically during the optimization process, i.e. the number of blocks, their sizes and the assignment of the constraint equations to the blocks may all vary in time. This feature makes this scheme especially convenient for LP problems with a large number of constraints.

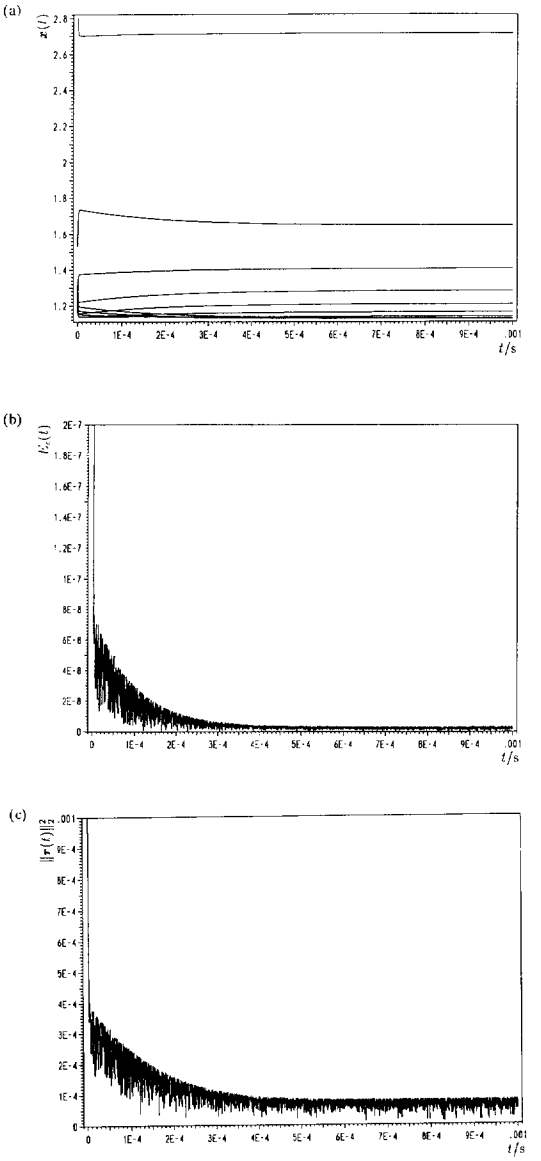

Fig. 10. (a) State-variables, (b) energy-function and (c) residuum $\|Ax - b\|_2^2$ for the very ill conditioned problem described in Example 3. The variables $x_j(t)$ all have been startet for $t = 0$ at $x_j(0) = 2$, $0 \leqslant j \leqslant 10$. Due to the long simulation time (1ms), the short-time dynamical behavior is not visible in the plots of (a)–(c).

The proposed neural network can serve as an effective computational model for solving real-time and large-scale LP problems. It may be especially attractive for real-time and/or high throughput rate appli-

cations in which the cost function and the constraints are slowly changing in time and if it is necessary to continuously track or update the optimal solution. The developed approach can be easily extended to other convex programming problems. The dynamic behavior and performance of the proposed network has been illustrated through extensive computer simulations.

References

[1] Bouzerdoum, A., and Pattison, T.R., "Neural networks for quadratic optimization with bound constraints", *IEEE Transactions on Neural Networks* 4 (1993) 293–304.

[2] Chen, J., Shanblatt, M.A., and Maa, C.Y., "Improved neural networks for linear and nonlinear programming", *International Journal of Neural Systems* 2 (1992) 331–339.

[3] Chua, L.O., and Lin, G.N., "Nonlinear programming without computation", *IEEE Transactions on Circuits and Systems* CAS-31 (1984) 182–188.

[4] Cichocki, A., and Bargiela, A., "Neural networks for solving linear inequality systems", *J. Parallel Computing* (in print); http://www.bip.riken.go.jp./absl, 1996.

[5] Cichocki, A., and Unbehauen, R., "Switched-capacitor neural networks for differential optimization", *International Journal of Circuit Theory and Applications* 19 (1991) 161–187.

[6] Cichocki, A., and Unbehauen, R., *Neural Networks for Optimization and Signal Processing*, Teubner–Wiley, Chichester, corrected edition, 1994.

[7] Cichocki, A., and Unbehauen, R., "Simplified neural networks for solving linear least squares and total least squares problems in real time", *IEEE Transactions on Neural Networks* 5 (1994) 910–923.

[8] Cichocki, A., Unbehauen, R., Lendl, M., and Weinzierl, K., "Neural networks for linear inverse problems with incomplete data especially in applications to signal and image reconstruction", *Neurocomputing* 8/1 (1995) 7–41.

[9] Danzig, G.B., *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[10] Dennis, J.B., *Mathematical Programming and Electrical Networks*, Chapman and Hall, London, 1959.

[11] Forrest, J.J.H., and Tomlin, J.A., "Implementing simplex method for the optimization subroutine library", *IBM Systems Journal* 31 (1992) 11–38.

[12] Gonzaga, C.C., "Path-following methods for linear programming", *SIAM Review* 34 (1992) 167–224.

[13] Hölzel, R., "Investigation of neural networks for linear and quadratic programming", Studienarbeit, University Erlangen-Nürnberg, Lehrstuhl für Allgemeine und Theoretische Elektrotechnik, 1993 (in German).

[14] Karmarkar, N., "A new polynomial-time algorithm for linear programming", *Combinatorica* 4 (1984) 373–395.

[15] Karpinskaya, N.N., "Method of 'penalty' functions and the foundation of Pyne's method", *Automation and Remote Control* 28 (1967) 124–129.

[16] Kennedy, M.P., and Chua, L.O., "Neural networks for nonlinear programming", *IEEE Transactions on Circuits and Systems* 35, 554–562.

[17] Lillo, W.E., Hui S., and Zak, S.H., "Neural networks for constrained optimization problems", *International Journal of Circuit Theory and Applications* 21 (1993) 293–304.

[18] Osborne, M.R., *Finite Algorithms in Optimization and Data Analysis*, Wiley, Chichester, 1985.

[19] Martinelli, G., and Perfetti, R., "Neural networks for real-time synthesis of FIR filters", *Electronics Letters* 25/17 (1989) 1199–1200.

[20] Pyne, I.B., "Linear programming on an electronic analogue computer", *Transactions of the American Institute of Electrical Engineers* 75 (1956) 139–143.

[21] Rodriquez-Vazquez, A., Dominguez-Castro, R., Rueda, A. Huertas, J.L., and Sanchez-Sinencio E. (1990), "Nonlinear switched-capacitor 'neural' networks for optimization problems", *IEEE Transactions on Circuits and Systems* 37 (1988) 384–397.

[22] Rybashov, M.V., "The gradient method of solving convex programming problems on electronic analog computers", *Automation and Remote Control* 26 (1965) 1886–1898.

[23] Tank, D.W., and Hopfield, J.J., "Simple 'neural' optimization networks: an A/D converter, signal decision circuit and a linear programming circuit", *IEEE Transactions on Circuits and Systems* CAS-33 (1986) 533–541.

[24] Wang, J., "Analysis and design of a recurrent neural network for linear programming", *IEEE Transactions on Circuits and Systems* 40 (1993) 613–618.

[25] Wang, J., "A deterministic annealing neural network for convex programming", *Neural Networks* 7 (1994) 629–641.

[26] Zak, S.H., Upatising, V., and Hui, S., "Neural networks for solving linear programming problems", *IEEE Transactions on on Circuits and Systems* (in print).

[27] Zhu, X., Zhang, S., and Constantinides, A.G., "Lagrange neural networks to linear programming", *Journal of Parallel and Distibuted Computing* 14 (1992) 354–360.