Common Nature of Learning between BP and Hopfield-Type Neural Networks for Convex Quadratic Minimization with Simplified Network Models

Yunong Zhang¹, Yanyan Shi², Binghuang Cai¹, Zhan Li¹, Chenfu Yi¹, Jianzhang Mai²
¹. School of Information Science and Technology ². School of Software
Sun Yat-Sen University (SYSU), Guangzhou 510275, China
zhynong@mail.sysu.edu.cn, ynzhang@ieee.org

Abstract—In this paper, two different types of neural networks are investigated and employed for the online solution of strictly-convex quadratic minimization; i.e., a two-layer back-propagation neural network (BPNN) and a discrete-time Hopfield-type neural network (HNN). As simplified models, their error-functions could be defined directly as the quadratic objective function, from which we further derive the weight-updating formula of such a BPNN and the state-transition equation of such an HNN. It is shown creatively that the two derived learning-expressions turn out to be the same (in mathematics), although the presented neural-networks are evidently different from each other in terms of network architecture, physical meaning and training patterns. Computer-simulations further substantiate the efficacy of both BPNN and HNN models on convex quadratic minimization and, more importantly, their common nature of learning.

Index Terms—Convex quadratic minimization, BP neural networks, Hopfield networks, common nature of learning.

I. INTRODUCTION

The problem of online minimizing convex quadratic functions has been an important topic in various scientific fields [1]-[8], such as, nonlinear optimization, optimal control, and their engineering applications (note that quadratic programs subject to linear constraints can be transformed readily to unconstrained (quadratic) minimization problems via penalty, Lagrange and/or other methods [3]). Here, as of our concern and interest, the quadratic function to be minimized is as follows (being an illustrative example of this research):

\[ V(x) = x^T P x/2 + q^T x, \]

where \( x \in \mathbb{R}^{n \times 1} \) is an independent decision-variable-vector, \( P \in \mathbb{R}^{n \times n} \) is a symmetric positive-definite coefficient matrix, and \( q^T \in \mathbb{R}^{1 \times n} \) is a coefficient vector (with superscript \( T \) denoting the transpose of a vector or matrix). As quadratic function (1) is strictly convex, it has a unique minimum solution \( x^* = -P^{-1} q \), satisfying \( \partial V(x)/\partial x|_{x=x^*} = P x + q|_{x=x^*} = 0 \), where \( \partial V(x)/\partial x = P x + q \) is the gradient of quadratic function (1).

As perhaps we know, there are two general types of methods for solving the kind of problems (e.g., quadratic minimization). One is the numerical algorithms performed on nowadays’ digital computers, of which the minimal arithmetic operations are usually proportional to the cube of the coefficient-matrix dimension \( n \), i.e., \( O(n^3) \) operations [9]. Thus, such serial-processing numerical algorithms may not be efficient enough for large-scale online applications.

Being another type of solution methods, parallel processing computational-methods (e.g., neural networks) have been extensively developed, analyzed and implemented [7][10]-[15]. Being one of the most important parallel and distributed processing methods, the neural-dynamic approach (e.g., Hopfield neural networks [16]) has been viewed as a powerful alternative for online quadratic minimization (1) owing to its convenience of hardware implementation [8].

Among different neural-network models [3][7][8][10]-[19], in this paper, we introduce a simple-structure BPNN which can be applied to solving the problem (1), and then compare it with the HNN model exploited in solving (1) as well. More importantly, although the network-architecture, physical-meaning and training pattern of BPNN are completely different from the HNN’s, we could show creatively that the weight-updating formula of the presented BPNN is even the same as the state-transition equation of the presented HNN, when applied to minimizing the quadratic function (1).

The remainder of this paper is arranged in four sections. Section II introduces the architecture of the presented BPNN for solving problem (1), from which the weight-updating formula can be derived. In Section III, a Hopfield-type neural network is presented for solving (1) as well, of which the discrete-time model is shown to have the same governing mathematical expression as that in BPNN! This tells us that the aforementioned BPNN and HNN models possess a common nature of learning. Section IV provides the computer-simulation results of the presented BPNN via MATLAB neural network toolbox [19] and via the common-learning formula of BPNN and HNN models. Section V concludes this paper with final remarks.

II. SIMPLIFIED BPNN

BPNN, with its learning algorithm presented in 1986, is one of the most widely-used neural-network models in computational-intelligence research area and engineering fields [15]-[18]. Its adaptation procedure can usually be summarized as the following two steps: 1) forward computation of working signal, and 2) backward propagation (BP) of training error.
For structure and implementation simplicity, linear activation function $f(v) = v$ is adopted to construct all neurons used in the BPNN with each threshold $b$ set zero. According to Fig. 1, the neural-network weights vector $w$ and the neural-network input vector $y$ could be defined respectively as

$$w = [w_1, w_2, \ldots, w_n]^T \in \mathbb{R}^{n \times 1},$$

$$y = [y_1, y_2, \ldots, y_n] \in \mathbb{R}^{1 \times n},$$

while $z \in \mathbb{R}$ is the corresponding neural-network output. Evidently, it follows that the relationship between network-input $y$ and output $z$ can be written as

$$z = f \left( \sum_{i=1}^{n} y_i w_i \right) = \sum_{i=1}^{n} y_i w_i = yw. \quad (3)$$

**B. Training Samples**

When the BPNN model depicted in Fig. 1 is applied to computing the minimum solution $x^* = -P^{-1}q$ of problem (1), the symmetric positive-definite coefficient matrix $P \in \mathbb{R}^{n \times n}$ can be rewritten as the following column-vector form:

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix} := [p_1, p_2, \ldots, p_n].$$

Then, the column-vector $p_i \in \mathbb{R}^{n \times 1}$ is used as the $i$th sample’s input $y^{(i)}$ for the presented BPNN, with its corresponding sample-output (or termed, target output, desired output) $z^{(i)}$ being $z^{(i)} = -q_i$. Note that $q_i$ denotes the $i$th element of coefficient vector $q$ mentioned in (1). In mathematics, the $i$th sample’s input $y^{(i)}$ and output $z^{(i)}$ can be expressed in pairs as $(p_i, -q_i)$, $i = 1, 2, \ldots, n$, where

$$y^{(i)} = [y_1^{(i)}, y_2^{(i)}, \ldots, y_n^{(i)}] := [p_{i1}, p_{i2}, \ldots, p_{in}] = Pw^T, \quad z^{(i)} := -q_i. \quad (4)$$

Evidently, based on the relation (3) between network input $y$ and output $z$, considering the training samples (4), we can readily define a scalar-valued positive (or at least lower-bounded) training-error function for the BPNN model depicted in Fig. 1 to perform the learning (or adaptation).

**C. Weights-Updating Formula**

In order to obtain the weights-updating formula, we could define the following training-error function $E(w)$ [which is actually equal to the objective quadratic function (1)] for this simplified BPNN (in neural-network terminology, the BPNN is trained in a batch-processing mode [15]-[18]):

$$E(w) = \frac{1}{2} \sum_{j=1}^{n} \left( w_j \sum_{i=1}^{n} (y_i^{(j)} w_i) - \sum_{i=1}^{n} (z^{(i)} w_i) \right)$$

$$= \frac{1}{2} \sum_{j=1}^{n} \left( w_j \sum_{i=1}^{n} (w_i p_{ij}) \right) + \sum_{i=1}^{n} (q_i w_i)$$

$$= \frac{1}{2} \left[ \sum_{i=1}^{n} w_i p_{i1}, \sum_{i=1}^{n} w_i p_{i2}, \ldots, \sum_{i=1}^{n} w_i p_{in} \right] w + q^T w$$

$$= \frac{1}{2} \left[ w^T p_1, w^T p_2, \ldots, w^T p_n \right] w + q^T w$$

$$= \frac{1}{2} w^T [p_1, p_2, \ldots, p_n] w + q^T w = w^T Pw/2 + q^T w.$$

Secondly, by following the back-propagation idea of training-error $E(w)$ via a gradient-descent method, this BPNN’s weights-updating formula could be set as

$$\Delta w = \frac{\partial E(w)}{\partial w} \bigg|_{w=w(k)} = -\eta \left( \frac{\partial E(w)}{\partial w} \right) \bigg|_{w=w(k)} = -\eta (Pw(k) + q),$$

which could be written explicitly as the following form:

$$w(k+1) = w(k) - \eta (Pw(k) + q), \quad (5)$$

In the ensuing subsections, a special-type simple-structure BP neural network (together with its architecture description and weights-updating formula) is proposed for solving online the quadratic-minimization problem (1).

**A. Network Architecture**

To minimize the convex quadratic function (1), we can construct a two-layer BPNN model as depicted in Fig. 1. For structure and implementation simplicity, linear activation function $f(v) = v$ is adopted to construct all neurons used in the BPNN with each threshold $b$ set zero. According to Fig. 1, the neural-network weights vector $w$ and the neural-network input vector $y$ could be defined respectively as

$$w = [w_1, w_2, \ldots, w_n]^T \in \mathbb{R}^{n \times 1},$$

$$y = [y_1, y_2, \ldots, y_n] \in \mathbb{R}^{1 \times n},$$

while $z \in \mathbb{R}$ is the corresponding neural-network output. Evidently, it follows that the relationship between network-input $y$ and output $z$ can be written as

$$z = f \left( \sum_{i=1}^{n} y_i w_i \right) = \sum_{i=1}^{n} y_i w_i = yw. \quad (3)$$

Fig. 1. A simple BPNN minimizing the function (1).

Fig. 2. BPNN block-diagram minimizing (1).
where the neural-network learning rate $\eta > 0$ should be sufficiently small, and iteration index $k = 0, 1, 2, \ldots$. Moreover, compared to conventional BP learning algorithms, the weights-updating formula may show a beautiful simplicity of the presented BPNN (in addition to its structural one).

As a result, after the simplified BPNN model in Fig. 1 being trained with weights-updating formula (5) in a sufficient number of iterations, we could have the following convergence result (which is also in view of coefficient matrix $P$ being symmetric and positive definite):

$$
\lim_{k \to +\infty} w(k) = w^* = -P^{-1}q = x^*.
$$

In other words, this simple-structure BPNN could solve problem (1) in the manner that its weights vector $w$ converges to the theoretical minimal solution $x^* = -P^{-1}q$ of quadratic-minimization problem (1) after a sufficient number of weights-updating iterations. The minimum value of quadratic function (1), i.e., $V(x^*) = V(w^*) = w^*TPw^*/2 + q^Tw^*$, could thus be achieved. Besides, the above-mentioned procedure of using the BPNN model to online minimize convex quadratic function (1) is represented via Fig. 2.

III. HOPFIELD-TYPE NEURAL NETWORK

Since [16], many gradient-based continuous-time Hopfield-type neural networks (HNN) have been developed and widely investigated for online problems’ solving. We can now generalize such an HNN approach to solving online the problem of minimizing quadratic function (1); i.e., to minimize $V(x) = x^TPx/2 + q^Tx$, compared to conventional neural-network models, this paper presents a simplified discrete-time HNN model, of which the design procedure is shown as below.

Firstly, different from the conventional definition of positive energy function for Hopfield neural networks, we can define the scalar-valued lower-bounded error function $\epsilon = x^TPx/2 + q^Tx$, in order to minimize the objective quadratic function (1) directly and also for the purpose of simplicity as well as the convenience in constructing such an HNN model.

Secondly, we can utilize the negative of the gradient $\partial \epsilon / \partial x = Px + q$ as the descent direction to minimize $\epsilon$. Then, we can have the following simplified continuous-time HNN model which minimizes (1) in real time $t$:

$$
\frac{dx}{dt} = \dot{x} = -\mu \frac{\partial \epsilon}{\partial x} = -\mu(Px + q),
$$

where design parameter $\mu > 0$, being the reciprocal of a capacitance parameter and setting as large as hardware permits [16], scales the convergence rate of the HNN model.

In order to compare with the aforementioned BPNN model (5) for the same problem-solving task, we could discretize HNN model (6) via Euler’s method (i.e., finite difference) so as to have the corresponding discrete-time HNN model:

$$
\frac{x(k + 1) - x(k)}{h} = -\mu(Px(k) + q),
$$

i.e., $x(k + 1) = x(k) - \eta(Px(k) + q),

where learning rate $\eta := h\mu > 0$ should be sufficiently small (with $h > 0$ denoting the sampling interval), and iteration index $k = 0, 1, 2, 3, \ldots$. Expression (7) is also termed neural-state transition equation, showing the neural-state changing from $x(k)$ to $x(k + 1)$ in the presented HNN model.

Moreover, by denoting $x_i$ as the $i$th element of neural-state vector $x$, discrete-time HNN (7) can be expressed in the following neuron-dynamics (for hardware-implementation):

$$
x_i(k + 1) = x_i(k) - \eta(p_{1i}x_1(k) + p_{2i}x_2(k) + \cdots + p_{ni}x_n(k)) - \eta q_i,
$$

where weight $p_{ij}$ is defined as the $ij$th entry of matrix $P$, and bias $q_i$ is the $i$th element of vector $q$, for $i, j = 1, 2, \ldots, n$.

Based on the above neuron-dynamics [i.e., (8)], we can obtain the circuit schematics and network structure of discrete HNN model (7) as shown in Figs. 3 and 4, which could be realized via digital circuits for minimizing quadratic function (1) iteratively and in parallel. Furthermore, in view of matrix $P$ being symmetric positive definite, the following convergence result could also be guaranteed for discrete HNN model (7) after a sufficient number of neural-state-vector transitions:

$$
\lim_{k \to +\infty} x(k) = x^* = -P^{-1}q.
$$

In other words, the presented HNN model can solve the problem of online minimizing quadratic function (1) as well, in the manner that its state vector $x(k)$ converges to the theoretical minimum solution $x^* = -P^{-1}q$ of problem (1). As a result, the minimum value of quadratic function (1), i.e., $V(x^*) = x^*TPx^*/2 + q^Tx^*$, is obtained [as $x(k) \to x^*$].

Now, let us stop at this point for a while. By comparing the BPNN weights-updating formula (5) and the HNN state-transition equation (7) carefully, we may find out a link (or to say, connection, relation); i.e., such two neural-network solvers of different types essentially possess the same mathematical expressions. Simply put, although the presented BPNN and
HNN models are deemed completely different (e.g., different concepts, definitions, physical meanings, structures, and training patterns), both of their learning schemes (or to say, computational-intelligence governing-equations) could finally be unified to be the same:

\[
u(k + 1) = u(k) - \eta(Pu(k) + q).
\] (9)

This makes us believe that a common nature of learning possibly exists inside many different kinds of neural networks; e.g., at least inside the presented BP and Hopfield neural networks. In addition, computer-simulation results in the ensuing section substantiate further the efficacy and commonness of governing-equation (9).

IV. COMPUTER SIMULATION AND VERIFICATION

In order to simulate and verify the presented BPNN and HNN models used in minimizing quadratic function (1) as well as their common nature of learning, we can firstly carry out the computer simulation of the BPNN model via the MATLAB neural network toolbox in Subsection IV-A. Then, for comparative purposes, we conduct the computer simulation of common-learning governing-equation (9) for both BPNN and HNN models in Subsection IV-B.

A. Via MATLAB Neural Network Toolbox

In this subsection, MATLAB neural network toolbox is used to simulate the simplified BPNN model (5) (i.e., in Fig.1) with the following MATLAB code.

```matlab
net = network(1,1); net.inputs{1}.size = 4;
net.layers{1}.size = 1;
net.biasConnect = [0];
net.inputConnect = [1];
net.layerConnect = [0];
net.outputConnect = [1];
net.targetConnect = [1];
net.layerWeights{1}.learn = 1;
net.layers{1}.transferFcn = 'purelin';
net.performFcn = 'mse';
net.trainParam.epochs = 500;
net.trainParam.lr = 1.2/trace(P'*P);
net.trainParam.goal = 1e-10;
net.trainParam.epochs = 500;
```

According to Subsection II-B, we can randomly choose coefficients \( P \in R^{4 \times 4} \) and \( q^T \in R^{1 \times 4} \), being an example, as follows (with \( P \) positive-definite and symmetric):

\[
P =
\begin{bmatrix}
0.1811 & 1.0974 & 0.2754 & 0.6674 \\
1.0974 & 1.6144 & 0.2754 & 0.1811 \\
0.2754 & 1.9904 & 0.4661 & 1.2601 \\
0.6674 & 0.1811 & 0.4661 & 1.2601
\end{bmatrix},
\]

\[
q =
\begin{bmatrix}
0.3659 \\
-0.1222 \\
0.2032 \\
0.2957
\end{bmatrix}.
\]

From the above-presented MATLAB code, we know that the learning-function “learngd” and the training-function “trainld” are employed to update the network weights based on the conventional gradient-descent method. In addition, “mse” in the MATLAB code stands for mean squared error and is set as the neural-network measurement index (with the goal-error being \( 10^{-10} \)), which is an illustration given in this paper intentionally showing that the model depicted in Fig. 1 does belong to the BP neural-network type. The simulation result of this simplified BPNN via MATLAB neural network toolbox is shown in Fig. 5(a), where the final training error (mse) is about \( 9.7567 \times 10^{-11} \), having met the prescribed error requirement. The neural-network output \( z \) and weights vector \( w \) at the 351st training epoch (or to say, iteration in our research terminology) are respectively

\[
\begin{bmatrix}
z^{(1)} \\
z^{(2)} \\
z^{(3)} \\
z^{(4)}
\end{bmatrix} =
\begin{bmatrix}
0.3659 \\
-0.1222 \\
0.2032 \\
0.2957
\end{bmatrix},
\begin{bmatrix}
w_1(351) \\
w_2(351) \\
w_3(351) \\
w_4(351)
\end{bmatrix} =
\begin{bmatrix}
-0.1603 \\
0.1226 \\
0.0093 \\
-0.1708
\end{bmatrix}.
\]

In addition, by substituting the weights-vector \( w(351) \) into strictly-convex quadratic function (1), we can get its minimum value as \( V(w(k)) = w^T(k)Pw(k)/2 + q^Tw(k) = -0.0611 \).

B. Via Common Learning Expression (9)

In comparison, we could also compute the neural-weights vector \( w \) in BPNN (or neural-state \( x \) in HNN) via learning formula (9) by using the following MATLAB code.
The values of coefficients $P \in \mathbb{R}^{4 \times 4}$ and $q \in \mathbb{R}^{4 \times 1}$ are specified to be the same as those in Subsection IV-A, and so is the error precision $10^{-10}$. Evidently, as seen from Fig. 5, the solution error synthesized by BPNN and HNN expression (9) could reach the goal-error (i.e., the error precision $10^{-10}$) just within 183 iterations, whereas, for the same goal-error, the neural network toolbox takes 351 epoches as mentioned in the previous subsection. Fig. 6 shows the convergence of $u(k)$ (i.e., the neural-weights vector $w$ in BPNN or the neural-state $x$ in HNN) to a unique steady-state solution $u^* = -P^{-1}q$:

$$u(183) = \begin{bmatrix} u_1(183) \\ u_2(183) \\ u_3(183) \\ u_4(183) \end{bmatrix} = \begin{bmatrix} -0.1604 \\ +0.1226 \\ +0.0094 \\ -0.1708 \end{bmatrix},$$

which is almost the same as the result obtained by the MATLAB neural network toolbox in Subsection IV-A. After substituting $u(k)$ into quadratic function (1), the $V(u(k))$-value solved by the presented BP and Hopfield neural networks [i.e., $V(u(k)) = u^T(k)Pu(k)/2 + q^T u(k)$] converges to the theoretical minimum value $V(u^*) = u^{*T} Pu^*/2 + q^T u^* = -0.0611$. This is also shown in Fig. 7.

In summary, comparing the simulation results obtained in Subsections IV-A and IV-B for the same purpose of minimizing quadratic function (1), we can see that the BPNN solver constructed by MATLAB neural network toolbox is feasible, convenient, effective and efficient to some extent. In terms of the number of iterations and accuracy, the common learning expression (9) appears to be superior to the neural network toolbox. Specifically speaking, with both under the goal-error requirement of $10^{-10}$ to meet, the training iterations of common learning expression (9) is 183, whereas the training epochs for MATLAB neural network toolbox is 351. More importantly and creatively, in addition to the theoretical analysis, the computer-simulation results in this section substantiate further the link and commonness of the BP and Hopfield neural networks. That is, such two neural-network solvers essentially possess the same mathematical expressions and computational results, though they, of different types, are deemed completely different (e.g., with different origins, concepts, definitions, physical meanings, structures, and training patterns).
Fig. 7. \( V(u(k)) \) convergence when solving (1) via common learning expression (9).

V. CONCLUSIONS

Through solving the problem of online minimization of strictly-convex quadratic function (1), two types of neural networks (i.e., the simplified BP and Hopfield neural networks) have been presented and investigated in this paper. Theoretical analysis shows that the BPNN weight-updating formula (5) and the HNN state-transition equation (7) could be governed by the same mathematical expression \([i.e., \text{the common learning expression (9)}]!\) This implies that, despite different network-structures, physical meanings and training patterns, the two types of neural-network models (supervised and unsupervised) can both possess and show a common nature of learning during the problem-solving process. Computer-simulation results have further substantiated the efficacy and commonness of both BPNN and HNN models for quadratic-function minimization purposes.

ACKNOWLEDGMENT

This research is funded by the National Natural Science Foundation of China under Grant 60775050, the Program for New Century Excellent Talents in University (NCET-07-0887), and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry, China. The first/corresponding author, Yunong Zhang, acknowledges here that all authors of this paper are jointly of the first authorship in view of their equally important contributions to this work. Besides, before joining SYSU in 2006, Yunong Zhang, had been with National University of Ireland at Maynooth, University of Strathclyde, National University of Singapore, Chinese University of Hong Kong, and South China University of Technology, since 1996. He has continued the line of this research, supported by the fellowships, assistantship and awards since then. His web-page is now available at http://www.ee.sysu.edu.cn/teacher/detail.asp?sn=129. Continuing the research and recalling the path, Yunong writes down: “Life may also be a paper, of which one paragraph takes ten years to draft”.

REFERENCES