1-5-2005

Efficient training algorithms for a class of shunting inhibitory convolutional neural networks

Fok Hing Chi Tivive
University of Wollongong, tivive@uow.edu.au

Abdesselam Bouzerdoum
University of Wollongong, bouzer@uow.edu.au

Efficient training algorithms for a class of shunting inhibitory convolutional neural networks

Abstract

This article presents some efficient training algorithms, based on first-order, second-order, and conjugate gradient optimization methods, for a class of convolutional neural networks (CoNNs), known as shunting inhibitory convolution neural networks. Furthermore, a new hybrid method is proposed, which is derived from the principles of Quickprop, Rprop, SuperSAB, and least squares (LS). Experimental results show that the new hybrid method can perform as well as the Levenberg-Marquardt (LM) algorithm, but at a much lower computational cost and less memory storage. For comparison sake, the visual pattern recognition task of face/nonface discrimination is chosen as a classification problem to evaluate the performance of the training algorithms. Sixteen training algorithms are implemented for the three different variants of the proposed CoNN architecture: binary-, Toeplitz- and fully connected architectures. All implemented algorithms can train the three network architectures successfully, but their convergence speed vary markedly. In particular, the combination of LS with the new hybrid method and LS with the LM method achieve the best convergence rates in terms of number of training epochs. In addition, the classification accuracies of all three architectures are assessed using ten-fold cross validation. The results show that the binary- and Toeplitz-connected architectures outperform slightly the fully connected architecture: the lowest error rates across all training algorithms are 1.95% for Toeplitz-connected, 2.10% for the binary-connected, and 2.20% for the fully connected network. In general, the modified Broyden-Fletcher-Goldfarb-Shanno (BFGS) methods, the three variants of LM algorithm, and the new hybrid/LS method perform consistently well, achieving error rates of less than 3% averaged across all three architectures.

Keywords

convolution, image processing, learning (artificial intelligence), least squares approximations, neural net architecture, pattern recognition

Disciplines

Physical Sciences and Mathematics

Publication Details

This article originally appeared as: Tivive, FHC and Bouzerdoum, A, Efficient training algorithms for a class of shunting inhibitory convolutional neural networks, IEEE Transactions on Neural Networks, May 2005, 16(3), 541-556. Copyright IEEE 2005.

Efficient Training Algorithms for a Class of Shunting Inhibitory Convolutional Neural Networks

Fok Hing Chi Tivive and Abdesselam Bouzerdoum, *Senior Member, IEEE*

***Abstract*—This article presents some efficient training algorithms, based on first-order, second-order, and conjugate gradient optimization methods, for a class of convolutional neural networks (CoNNs), known as shunting inhibitory convolution neural networks. Furthermore, a new hybrid method is proposed, which is derived from the principles of *Quickprop*, *Rprop*, *SuperSAB*, and least squares (LS). Experimental results show that the new hybrid method can perform as well as the Levenberg–Marquardt (LM) algorithm, but at a much lower computational cost and less memory storage. For comparison sake, the visual pattern recognition task of face/nonface discrimination is chosen as a classification problem to evaluate the performance of the training algorithms. Sixteen training algorithms are implemented for the three different variants of the proposed CoNN architecture: binary-, Toeplitz- and fully connected architectures. All implemented algorithms can train the three network architectures successfully, but their convergence speed vary markedly. In particular, the combination of LS with the new hybrid method and LS with the LM method achieve the best convergence rates in terms of number of training epochs. In addition, the classification accuracies of all three architectures are assessed using ten-fold cross validation. The results show that the binary- and Toeplitz-connected architectures outperform slightly the fully connected architecture: the lowest error rates across all training algorithms are 1.95% for Toeplitz-connected, 2.10% for the binary-connected, and 2.20% for the fully connected network. In general, the modified Broyden–Fletcher–Goldfarb–Shanno (BFGS) methods, the three variants of LM algorithm, and the new hybrid/LS method perform consistently well, achieving error rates of less than 3% averaged across all three architectures.**

***Index Terms*—Convolutional neural network (CoNN), first- and second-order training methods, shunting inhibitory neuron.**

I. Introduction

In RECENT years, there has been a shift toward the use of hierarchical two-dimensional (2-D) artificial neural networks (ANNs) for image processing, vision, and pattern recognition applications. These networks, commonly known as convolutional neural networks (CoNNs), are formed by one or more layers of 2-D filters, with possible nonlinear activation functions and subsampling. Formerly, they were developed as a model for the mechanism of pattern recognition in the mammalian visual system [1]. Subsequently, they have been successfully applied to handwritten digit and character recognition [2], [3], face recognition [4], automatic face analysis [5], face localization [6], among others. The key characteristics of CoNNs are local receptive fields[1] and translation invariant connection matrices. CoNNs possess several advantages over conventional ANNs:

1) the spatial topology of the input is well captured by the network structure;
2) the feature extraction stage is integrated with the classification stage, and both are generated by the learning process;
3) the concept of weight sharing (or weight replication) reduces the number of free (trainable) network parameters, hence, reducing the network complexity and improving generalization;
4) the hardware implementation of CoNNs is much simpler than that of fully connected ANNs of comparable input size.

Two well-known CoNN architectures have been developed in the past [2], [7]. Fukushima developed the Neocognitron [7], a biologically inspired multilayered neural network approach that models the mechanism of visual pattern recognition in the brain. It consists of a cascade of feature detection stages, each of which comprising two layers: a simple cell (S) layer and a complex cell (C) layer. LeCun and his colleagues, on the other hand, developed a series of CoNN architectures, dubbed LeNet (1–5), based on the perceptron neuron and the three architectural ideas of local receptive fields, weight sharing and subsampling [2], [8]. Their architectures consist of a cascade of convolutional and subsampling layers. The neurons in the convolutional layers are simple sigmoid type neurons; that is, the neuron output is a weighted sum of its inputs followed by a sigmoid squashing function.

Recently, we have proposed a new class of CoNNs, based on the physiologically plausible mechanism of shunting inhibition [9]. These networks, dubbed shunting inhibitory convolutional neural networks (SICoNNets), have a flexible "do-it-yourself" architecture, where the user only specifies the input size, the receptive field size, number of layers and/or number of feature maps, number of outputs, and the connection scheme between layers-there are three possible connection strategies. The processing elements of the hidden layers are shunting inhibitory neurons. There are two main reasons for such a choice. First, shunting inhibition has been extensively used to model some important visual and cognitive functions [10]–[15]. The second and more important reason is that when shunting neurons are used in a feedforward architecture for classification and nonlinear regression, they were found to be more powerful than the conventional multilayer perceptron (MLP) architecture; a single

Manuscript received December 12, 2003; revised October 23, 2004.

The authors are with the School of Electrical, Computer, and Telecommunications Engineering, University of Wollongong, Wollongong, NSW 2522, Australia (e-mail: fhct243@uow.edu.au; a.bouzerdoum@elec.uow.edu.au).

Digital Object Identifier 10.1109/TNN.2005.845144

[1]The receptive field is the area from which a neuron receives its inputs.

shunting neuron can form linear as well as nonlinear decision boundaries much more readily and, hence, it can solve linear nonseparable problems, such as the XOR problem [16], [17].

The implementation of training methods for CoNNs is much more complex than that of the MLPs due to weight sharing, the organization of the processing elements, and the connection strategies. Very few studies have been done to develop training algorithms for these 2-D architectures. Most of the existing CoNNs are either used with a simple training method such as steepest descent algorithm or they are trained layer by layer. This paper deals with the development of training algorithms for the proposed CoNN architecture. Training algorithms, based on first- and second-order gradient algorithms and hybrid optimization techniques, have been developed and tested on three different SICoNNet architectures. The rest of this article is organized as follows. The next section describes the proposed CoNN architecture and its connection strategies. The third section presents the various training algorithms that have been developed for SICoNNets. The experimental procedure and results are given in Section IV, followed by the conclusion in Section V. Finally, the Appendix presents the detailed derivation of the gradient terms required by the proposed training algorithms for the new network architecture.

II. Description of SICoNNet Architecture

This section describes in detail the proposed CoNN architecture and its various connection schemes. This is followed by a description of the mathematical model of the shunting neuron, the basic computing element of the feature maps in the proposed architecture. Finally, we highlight the major structural differences between the proposed architecture and existing CoNNs.

A. Network Architecture

The proposed CoNN architecture is a multilayer network where the input layer is a 2-D square array of arbitrary size. Each hidden layer consists of several planes of shunting inhibitory neurons. Each plane, also known as feature map, has a unique set of $N \times N$ incoming weights, where N is chosen to be an odd integer. The reasons why N is taken as an odd integer are: 1) to remove the ambiguity about the choice of the central weight among the set of incoming weights and 2) to facilitate the formulation of the output equation for the shunting neuron in the feature map and the derivation of the training algorithm. In the feature map, all the neurons share the same set of weights connecting to different locations in the input image (receptive field). This process allows the neurons in a feature map to extract elementary visual feature from the input image, and subsequent hidden layers. The same receptive field size is used to connect from one layer to the next layer throughout the network architecture. Subsampling is performed within each hidden layer by shifting the centres of receptive fields of neighboring neurons by two positions, horizontally and vertically, as shown in Fig. 1(a). In other words, this architecture merges the feature extraction layer and subsampling layer into one.

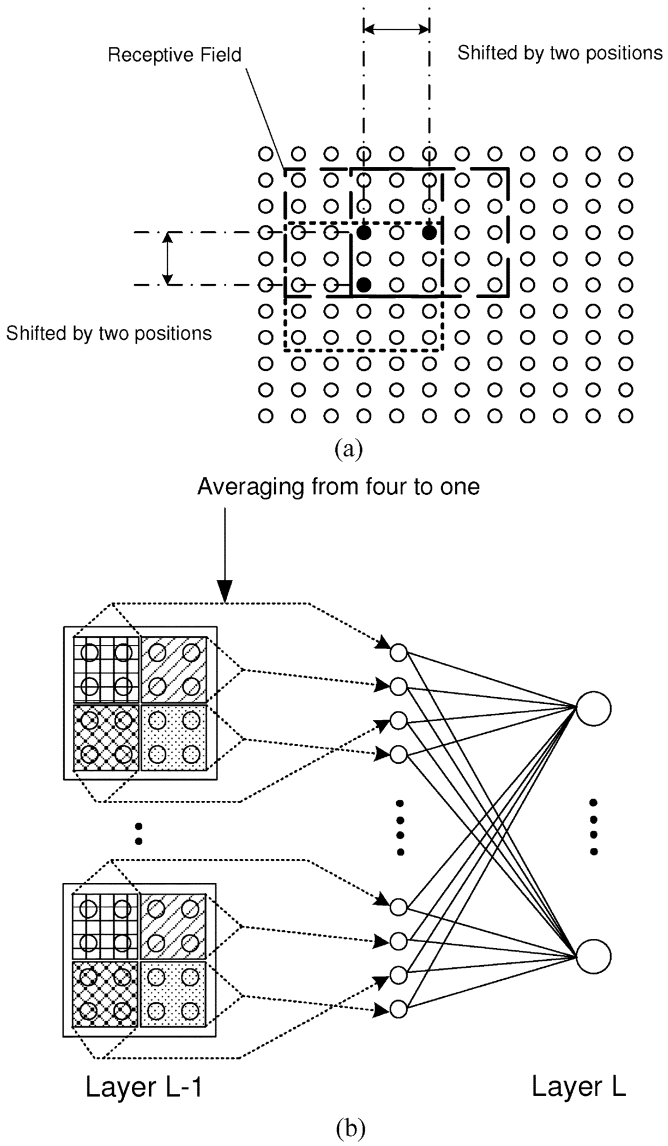


Fig. 1. (a) Movement of a receptive field in the input image or the feature map. (b) Local averaging of the feature maps of the $(N-1)$th layer.

Consequently, the size of the feature map is reduced by one quarter in successive layers.

The last hidden layer is fully connected to the output layer. However, to minimize the number of trainable weights, local averaging is performed on all feature maps of the last hidden layer: in the current implementation, small (2×2) nonoverlapping regions are averaged and the resulting signals are fed into the output layer [Fig. 1(b)]. In case where the feature maps in the last hidden layer consist of single neurons only, the outputs of these neurons serve as inputs directly to the output layer. The output layer consists of a set of linear or sigmoid neurons; that is, the response of an output neuron is a weighted sum of its input signals added to a bias term, and the result is passed through a linear or sigmoidal activation function. Mathematically, the response of an output neuron is given by

$$y = h\left(\sum_{i=1}^{S_N} w_i z_i + b\right) \tag{1}$$

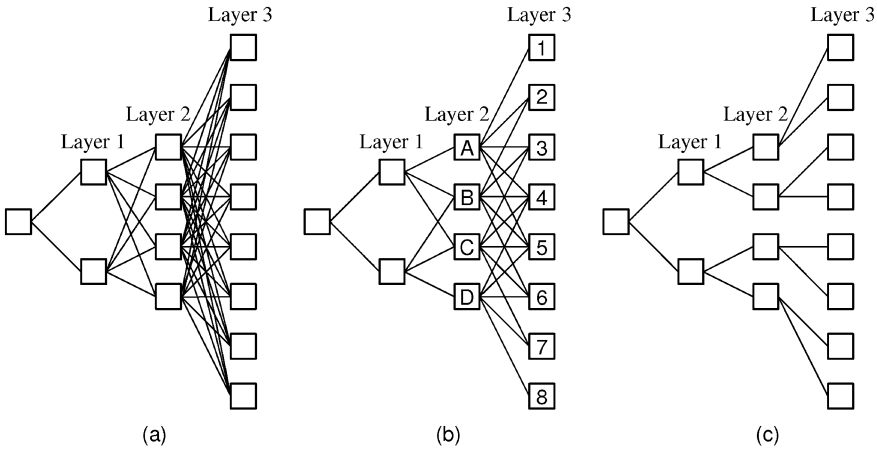


Fig. 2. Three schemes of SICoNNets. (a) Full-Connection. (b) Toeplitz-Connection. (c) Binary-connection.

where h is the activation function, w_i's are the connection weights, b is the bias term, and S_N is the number of inputs to the output layer N.

B. Connection Strategies

Even though the local receptive field is used as a medium to connect the feature maps in successive layers, a connection strategy between layers is required to construct the network topology. Therefore, three connection schemes, shown in Fig. 2, have been developed: full-connection, Toeplitz-connection and binary-connection. These connection strategies link successive layers in a systematic manner, without user interference. In the full-connection scheme, each feature map is fully connected to the feature maps in the succeeding layer, and each layer has an arbitrary number of feature maps, depending on the number of features to be extracted. This scheme is similar to the connection scheme of MLP, where the numbers of hidden layers and hidden units (equivalent to feature maps) can be altered arbitrarily. In the Toeplitz- and binary-connection schemes, the number of feature maps in each layer is constrained to an integer power of 2: the number of feature maps in the Lth hidden layer is equal to 2^L. Furthermore, in the binary-connection scheme, each feature map branches out to two feature maps in the succeeding layer, forming a binary tree [Fig. 2(c)].

In the Toeplitz-connection scheme, a feature map may have one-to-one or one-to-many links with feature maps in the preceding layer. As an example, Table I illustrates the connections between Layer 2 (L-2) and Layer 3 (L-3). Suppose that Layer 3 contains eight feature maps, labeled 1–8 (first column), and Layer 2 has four feature maps, labeled A, B, C, and D. Feature maps 1 and 8 have one-to-one connections with feature maps A and D, respectively. Feature map 2 has connections with feature maps B and A; the rest of the connections form a Toeplitz matrix, hence, the name. In other words, each feature map in Layer 2 connects to the same number of feature maps in Layer 3 (in this case five), and its connections appear along a diagonal of the connection matrix. The same principle is used to connect the other layers.

One can say that the two partial-connection strategies (Toeplitz and binary) are special cases of the full-connection scheme, whereby some of the connections are set to zero. These

TABLE I
LINKS BETWEEN FEATURE MAPS IN LAYER-2 AND LAYER-3 IN THE TOEPLITZ CONNECTION SCHEME

Layer-3 Feature Maps	Connections from L-2 to L-3				
1	A				
2	B	A			
3	C	B	A		
4	D	C	B	A	
5		D	C	B	A
6			D	C	B
7				D	C
8					D

two connection schemes are quite suitable for the proposed network architecture; in each layer, the size of the feature maps is reduced to one fourth of the size of the feature maps in the previous layer; to compensate for this loss, more feature maps are required. One advantage of these partial-connection schemes is that the network size depends only on the number of hidden layers and the number of neurons in the output layer. In the proposed architecture, however, the number of hidden layers is dependent on the size of the 2-D input.

C. Shunting Inhibitory Neuron Model

Shunting inhibition is a powerful computational mechanism that plays an important role in sensory information processing. From the time it was suggested as a plausible neuro-physiological mechanism in the early 1960s, shunting inhibition has been incorporated into several important neuro-information processing models. For example, Grossberg employed it extensively to model a number of visual and cognitive functions [10]. Pinter used it to model the adaptation phenomena in receptive field organizations and modulation transfer functions [11], [12]. A model of motion detection in insects, based on shunting inhibition, was developed by Bouzerdoum and Pinter [13], [14]. The same authors also developed a cellular neural network architecture, which has been applied to various vision and image processing tasks [15]. More recently, shunting inhibition has been used in a feedforward neural network architecture for supervised pattern classification and regression [17], [18]. In this architecture, the activation of the hidden neurons is governed

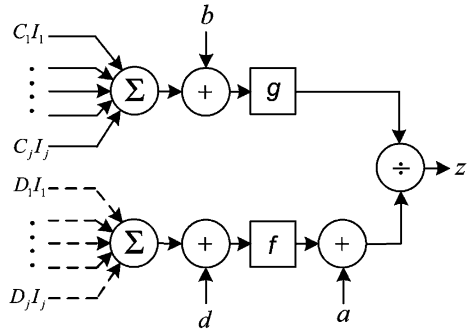


Fig. 3. Shunting neuron model.

by the steady-state response of the shunting inhibitory neuron model [16], [17], [19], [20], which has since been generalized to the following [18]:

$$z = \frac{g\left(\sum_{j=1}^{S_R} C_j I_j + b\right)}{a + f\left(\sum_{j=1}^{S_R} D_j I_j + d\right)} \tag{2}$$

where z is the activity of the shunting neuron, I_j is the jth input, a is the passive decay rate, C_j and D_j are the connection weights of the jth input, b and d are constant biases, f and g are activation functions, and S_R is the number of inputs inside the receptive field taken from the 2-D input, i.e., $S_R = N \times N$. A model of the shunting neuron is shown in Fig. 3.

In this paper, the neuron model (2) is used to build 2-D feature maps for the proposed CoNN architecture. Therefore, the output of the shunting neuron at location (i, j) in feature map L, k (kth feature map in the Lth layer) is shown in (3) at the bottom of the page where $Z_{L-1,v}$ represents the output of the vth feature map of the $(L-1)$th layer; S_{L-1} denotes the number of feature maps in the $(L-1)$th layer; S_F is the size of the feature map; $D_{L,k}$ and $C_{L,k}$ are the set of weights (convolution masks), and the "*" denotes the 2-D convolution operator. As stated earlier, all the shunting neurons in a feature map share the same weights, $D_{L,k}$ and $C_{L,k}$, but the biases and the passive decay rate constant can either be shared or each neuron can have its own $a_{L,k}(i,j)$, $b_{L,k}(i,j)$, and $d_{L,k}(i,j)$ terms. Furthermore, a constraint is imposed on (3) so as to avoid division by zero. More precisely, the denominator of (3) is kept always positive; in our case, it is bounded from the following inequality by a small positive constant:

$$a_{L,k}(i,j) + f_L\left(\sum_{v=1}^{S_{L-1}} [D_{L,k} * Z_{L-1,v}]_{(2i)(2j)} + d_{L,k}(i,j)\right) \geq 0.1. \tag{4}$$

For the previous inequality to hold, both the passive decay rate and the activation function f_L must be bounded from below. Therefore, given the lower bound of f_L, the lower bound $\tilde{a}_{L,k}(i,j)$ of the passive decay rate is set accordingly so that (4) is always satisfied

$$\tilde{a}_{L,k}(i,j) \geq 0.1 - \min(f_L). \tag{5}$$

This constraint is enforced during both initialization and training phases.

D. Structural Differences Between Our Network and the Existing CoNNs Structure

The similarity between the new class of CoNNs and the existing CoNN architectures [4], [21], [22] is that all of them are based on the same structural concepts of local receptive fields, weight sharing and subsampling. However, their implementations in the new architecture differ markedly. The CoNNs used in face detection [21] and face recognition [4] are based on the network architecture developed by LeCun *et al.* [2] for handwritten digit recognition. This network architecture was implemented in such way that each convolutional layer is followed by a subsampling layer. In [22], a blurring filter of size 3×3 has been used together with a subsampling operation. In our proposed network, the convolutional and subsampling layers are collapsed into one layer, which simplifies the network architecture. Furthermore, the receptive field size in the CoNN used for face detection [21] is different for each convolutional layer, whereas in our approach the same receptive field size is employed throughout the network architecture. In [2] and [21], the connections from the first subsampling layer to the second convolutional layer are manually specified by the user, depending on the specific task to be solved. In our proposed CoNN architecture, three types of systematic connection strategies have been developed: fully connected, Toeplitz-connected and binary-connected. However, the main difference between the CoNNs used in [4] and [21] and our network architecture is the use of the shunting neuron as the elementary processing unit in our architecture.

III. Training Algorithms

Very few studies have been devoted to the design of general training algorithms for CoNNs, barring some algorithms developed for specific architectures designed to solve specific tasks [2], [7]. This is due to the fact that the connection schemes used in existing architectures are quite complex, and very often need to be hand-coded for the task at hand. Nonetheless, the proposed CoNN architecture, with its systematic connection strategies, simplifies the development of general training algorithms.

$$Z_{L,k}(i,j) = \frac{g_L\left(\sum_{v=1}^{S_{L-1}} [C_{L,k} * Z_{L-1,v}]_{(2i)(2j)} + b_{L,k}(i,j)\right)}{a_{L,k}(i,j) + f_L\left(\sum_{v=1}^{S_{L-1}} [D_{L,k} * Z_{L-1,v}]_{(2i)(2j)} + d_{L,k}(i,j)\right)} \quad \text{for } i,j = 1,\ldots,S_F \tag{3}$$

Training a neural network with n free parameters (weights and biases) is equivalent to optimizing a function of n independent variables. In supervised learning, this function is usually taken as the mean squared error (MSE)

$$E(k) = \frac{1}{P \times M} \sum_{j=1}^{P} \sum_{i=1}^{M} \left(z_i^j - t_i^j\right)^2. \quad k = 1, \ldots, S_K \quad (6)$$

where M is the number of output neurons, P is the number of training patterns, S_K is the number of training iterations, z_i^j and t_i^j are, respectively, the actual and desired response of the ith output neuron due to the jth input pattern. Let $\mathbf{W}(k)$ be the n-dimensional column vector containing all n free parameters (i.e., adaptable weights) of the network at the kth iteration

$$\mathbf{W}(k) = [w_1(k), w_2(k), \ldots, w_n(k)]^T$$

where 'T' denotes the transpose operator. The weight vector is obtained by reshaping all the weights in the receptive fields and biases of the neurons in the feature maps, where elements are taken column-wise, from the first hidden layer to the last layer of the network, forming a large column vector. To optimize the error function in (6), the following update rule is applied iteratively, starting from an initial weight vector $\mathbf{W}(0)$

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \triangle\mathbf{W}(k) \quad (7)$$

with

$$\triangle\mathbf{W}(k) = \alpha(k)\mathbf{d}(k) \quad (8)$$

where $\triangle\mathbf{W}(k)$ is the weight-update vector, $\mathbf{d}(k)$ is a search direction, and $\alpha(k)$ is the step-length at the kth iteration. There are various ways for computing the search direction and the step-length, ranging from the simple gradient descent to the more efficient conjugate gradient and quasi-Newton methods. The simplest solution is to take a constant step-length, $\alpha(k) = \alpha$, and set the search direction to the negative gradient, which is the direction of the steepest descent from any given point on the error surface; that is

$$\triangle\mathbf{W}(k) = -\alpha(k)\mathbf{g}(k) \quad (9)$$

where $\mathbf{g}(k)$ is the gradient vector of the error function at the kth epoch. The gradient vector is an n-dimensional column vector given by

$$\mathbf{g}(k) = [g_1(k), g_2(k), \ldots, g_n(k)]^T$$

where $g_i(k) = \partial E(k)/\partial w_i (i = 1, \ldots, n)$ is the local gradient. This method is often called the steepest descent algorithm. In general, gradient information is required to effect a weight update. In a multilayer feedforward network, the gradient vector can be computed very efficiently using the error-backpropagation algorithm [23]; a derivation of error backpropagation for the proposed CoNN is given in the Appendix. Gradient descent methods usually work quite well during the early stages of the optimization process. However, as the error surface starts taking the form of a ravine or a very flat region, these methods behave poorly because only small steps are taken toward the bottom. Therefore, many optimization algorithms employ not only the gradient, but also the curvature of the error surface, to minimize the error function. This section presents a number of such techniques that have been implemented for the proposed CoNN architecture. They use batch training, in which any weight update is performed after the presentation of all input patterns.

A. First-Order Hybrid Training Method

In this paper we propose a hybrid training method based on the combination of *Rprop* [24], *Quickprop* [25], and *SuperSAB* [26]. It is a local adaptation strategy where the temporal behavior of the partial derivative of the weight is used in the computation of the weight-update. The weight update rule is given by

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \triangle\mathbf{W}(k) + \boldsymbol{\mu}(k) \cdot * \triangle\mathbf{W}(k-1) \quad (10)$$

where "$\cdot*$" is the element-by-element product of two column vectors. The weight-update vector $\triangle\mathbf{W}(k)$ is computed using the same principle as *Rprop* (more on this later). The vector of adaptive momentum rate $\boldsymbol{\mu}(k)$ is taken as the vector of magnitude of the *Quickprop*-step, as defined in (15). Furthermore, if there is a decrease in the current error with respect to the error in the previous iteration, a small percentage of the negative gradient is added to the weight

$$\mathbf{W}(k+1) = \mathbf{W}(k+1) - \boldsymbol{\alpha}(\boldsymbol{k}) \cdot * \mathbf{g}(k) \quad (11)$$

where $\boldsymbol{\alpha}(\boldsymbol{k})$ is the vector of step-length and is adapted in the same way as in the *SuperSAB* method, which is described in the following.

1) Calculating the Weight-Update $\triangle\mathbf{W}(k)$: From the strategy proposed by Riedmiller and Braun [24], the adaptation of the weight is divided into two parts. In the first part, the ith weight $w_i(k)$ of the weight vector $\mathbf{W}(k)$ is allowed to have its own step-size $\gamma_i(k)$, which is adjusted based on the observation of the behavior of the local gradient during two successive iterations

$$\gamma_i(k) = \begin{cases} \min(1.2\gamma_i(k-1), \gamma_{\max}), & \text{if } \triangle_c > 0 \\ \max(0.5\gamma_i(k-1), \gamma_{\min}), & \text{if } \triangle_c < 0 \\ \gamma_i(k-1), & \text{otherwise} \end{cases} \quad (12)$$

where $\triangle_c = g_i(k)g_i(k-1)$, $\gamma_{\max}$, and $\gamma_{\min}$ are the upper and lower limits of the step-size, respectively; the initial value $\gamma_i(0)$ is set to 0.001. In the second part, the weight-update of the ith weight $\triangle w_i(k)$ is determined using the "Manhattan-learning" rule

$$\Delta w_i(k+1) = -\text{sign}(g_i(k))\gamma_i(k) \quad (13)$$

where $\text{sgn}(\cdot)$ is the signum function. In addition, when the current local gradient has a change of sign with respect to the previous local gradient of the same weight, the stored local gradient is set to zero to avoid an update in that weight in the next iteration. In case of a change in the sign of the local gradient and an increase in the network error, a backtracking process is also included to revert back to the previous weight, which is multiplied by an adaptive momentum rate

$$\begin{aligned} &\text{if } \triangle_c < 0 \text{ and } E(k) > E(k-1) \\ &\text{then } \Delta w_i(k) = -\mu_i(k)\Delta w_i(k-1). \end{aligned} \quad (14)$$

2) Calculating the Adaptive Momentum Rate $\boldsymbol{\mu}(k)$: The adaptive momentum rate for the ith weight $\mu_i(k)$ is derived

from the *Quickprop* algorithm by taking the magnitude of the *Quickprop*-step, and is given by

$$\tilde{\mu}_i(k) = \left| \frac{g_i(k)}{g_i(k-1) - g_i(k)} \right|. \tag{15}$$

It is then constrained in the interval [0.5, 1.5] by (16) and (17)

$$\mu_i(k) = \min(\tilde{\mu}_i(k), 1.5) \tag{16}$$

$$\mu_i(k) = \begin{cases} \max(\tilde{\mu}_i(k), 0.5), & \text{if } \triangle_c < 0 \\ 0, & \text{if } \triangle_c = 0 \end{cases}. \tag{17}$$

3) Calculating the Adaptive Step Length $\boldsymbol{\alpha}(k)$: Tollenaere [26] proposed the use of a separate step-length for each weight; and the adaptation process is performed by (18)

$$\tilde{\alpha}_i(k) = \begin{cases} 1.2\alpha_i(k), & \text{if } \triangle_c > 0 \\ 0.5\alpha_i(k), & \text{if} \triangle_c < 0 \end{cases}. \tag{18}$$

To prevent the learning rate from increasing indefinitely, it is bounded as follows:

$$\alpha_i(k) = \min(\tilde{\alpha}_i(k), \alpha_{\max}) \text{ and } \alpha_{\max} = 0.9. \tag{19}$$

Hence, combining the functions use to compute the weight-update, the adaptive momentum rate and the adaptive step length, the proposed hybrid algorithm can be described by the following pseudocode.

Input: Initialize $\gamma_i = 0.001, \mu_i = 0.01$, and $\alpha_i = 0.01$.
Calculate the local gradient.
While stopping criterion is not met **do**
Calculate the adaptive momentum rate $\tilde{\mu}_i(k)$, according to (15) and bound it above by (16).
Calculate $\triangle_c = g_i(k)g_i(k-1)$.
if $\triangle_c > 0$ **then**
$\gamma_i(k) = \min(1.2\gamma_i(k-1), \gamma_{\max})$,
$\tilde{\alpha}_i(k) = 1.2\alpha_i(k-1)$.
else if $\triangle_c < 0$ **then**
$\gamma_i(k) = \max(0.5\gamma_i(k-1), \gamma_{\min})$,
$\tilde{\alpha}_i(k) = 0.5\alpha_i(k-1)$,
$\mu_i(k) = \max(\tilde{\mu}_i(k), 0.5)$,
$g_i(k) = 0$.
else if $\triangle_c == 0$ **then**
$\mu_i(k) = 0$.
end if
$\alpha_i(k) = \min(\tilde{\alpha}_i(k), \alpha_{\max})$.
$\Delta w_i(k) = -sgn(g_i(k))\gamma_i(k)$.
if $\triangle_c < 0$ and $E(k) > E(k-1)$ **then**
$\Delta w_i(k) = -\mu_i(k)\Delta w_i(k-1)$.
end if
$w_i(k+1) = w_i(k) + \Delta w_i(k) + \mu_i(k)\Delta w_i(k-1)$.
if $E(k) < E(k-1)$ **then**
$w_i(k+1) = w_i(k+1) - \alpha_i(k)g_i(k)$.
end if
end while

The limits for the step-size $\gamma_i(k)$ of the proposed hybrid training method are obtained after several training trials; they were set to $\gamma_{\max} = 10$ and $\gamma_{\min} = 10^{-10}$, respectively. In most of the trials, the convergence of the training was insensitive to the upper bound; nonetheless, the step-size was bounded by $\gamma_{\max}$ to prevent the training algorithm from diverging in some cases, and bounded by $\gamma_{\min}$ to reduce the risk of getting stuck in local minima by allowing a small change in the weights. The thresholds for the adaptive momentum rate given in (16) and (17) were selected by training the proposed networks while varying the adaptive momentum rate in the range [0, 2] with steps of 0.1. Most of the training trials show better convergence when $\mu_i(k)$ is limited within the range [0.5, 1.5]. For the upper limit of the learning rate $\alpha_{\max}$, a set of values in the interval [0, 1] were tested and a value of 0.9 was found to work well in most of the experiments.

B. Conjugate Gradient Methods

The conjugate gradient method is another efficient optimization technique; it can minimize a quadratic error function of n variables in n steps. This method generates a search direction that is mutually conjugate to the previous search directions, with respect to a given positive definite matrix $\mathbf{A}$, and finds the optimal point in that direction, using a line-search technique. Two search directions $\mathbf{d}(i)$ and $\mathbf{d}(j)$ are said to be mutually conjugate with respect to $\mathbf{A}$ if the following condition is satisfied:

$$\mathbf{d}^T(i)\mathbf{A}\mathbf{d}(j) = 0 \quad \text{where } i \neq j. \tag{20}$$

In other words, the next search direction is calculated as a linear combination of the previous direction and the current gradient, in such a way that the minimization steps in all previous directions are not interfered with. The next search direction can be determined as follows:

$$\mathbf{d}(k) = \begin{cases} -\mathbf{g}(k), & \text{for } k = 1 \\ -\mathbf{g}(k) + \beta(k)\mathbf{d}(k-1), & \text{for } k \geq 2 \end{cases}. \tag{21}$$

The variable $\beta(k)$ is a scalar chosen so that $\mathbf{d}(k)$ becomes the kth conjugate direction. There are various ways for computing the scalar $\beta(k)$: each one generates a distinct nonlinear conjugate gradient method which has its own convergence property and numerical performance. Several formulae for computing $\beta(k)$ have been proposed; the most notable ones are the following.

- Fletcher–Reeves (FR) [27]

$$\beta(k)^{\mathrm{FR}} = \frac{\|\mathbf{g}(k)\|^2}{\|\mathbf{g}(k-1)\|^2}. \tag{22}$$

- Polak–Ribière (PR) [28]

$$\beta(k)^{\mathrm{PR}} = \frac{\mathbf{g}^T(k)(\mathbf{g}(k) - \mathbf{g}(k-1))}{\|\mathbf{g}(k)\|^2}. \tag{23}$$

- Hestenes–Stiefel (HS) [29]

$$\beta(k)^{\mathrm{HS}} = \frac{\mathbf{g}^T(k)(\mathbf{g}(k) - \mathbf{g}(k-1))}{\mathbf{d}^T(k-1)(\mathbf{g}(k) - \mathbf{g}(k-1))}. \tag{24}$$

- Dai–Yuan (DY) [30]

$$\beta(k)^{\mathrm{DY}} = \frac{\|\mathbf{g}(k)\|^2}{\mathbf{d}^T(k-1)(\mathbf{g}(k) - \mathbf{g}(k-1))} \tag{25}$$

where $\|\cdot\|$ denotes the Euclidean norm.

In [31], Hu and Storey proposed a hybrid conjugate gradient method (HY-HuSt) by combining the good numerical performance of Polak–Ribière (PR) method and the nice global con-

vergence properties of Fletcher–Reeves (FR) method. Subsequently, Gilbert and Nocedal [32] suggested a modified version of Hu–Storey method (HY–GN). The respective parameter of these two methods is given by

$$\beta(k)^{\mathrm{HY-HuSt}} = \max\{0, \min\{\beta(k)^{\mathrm{PR}}, \beta(k)^{\mathrm{FR}}\}\} \tag{26}$$

and

$$\beta(k)^{\mathrm{HY-GN}} = \max\{-\beta(k)^{\mathrm{FR}}, \min\{\beta(k)^{\mathrm{PR}}, \beta(k)^{\mathrm{FR}}\}\}. \tag{27}$$

Dai and Yuan [33] developed two other hybrid conjugate gradient techniques (HY-DY, HY-DY-V1), and their experimental results showed that their methods outperform the Polak–Ribière method

$$\beta(k)^{\mathrm{HY-DY}} = \max\{-c\beta(k)^{\mathrm{DY}}, \min\{\beta(k)^{\mathrm{HS}}, \beta(k)^{\mathrm{DY}}\}\} \tag{28}$$

$$\beta(k)^{\mathrm{HY-DY-V1}} = \max\{0, \min\{\beta(k)^{\mathrm{HS}}, \beta(k)^{\mathrm{DY}}\}\} \tag{29}$$

where $c = (1-\sigma)/(1+\sigma) > 0$ (here σ is set to 0.1). Furthermore, it was suggested in [33] that (29) has better performance than (28) as it relates to the restart strategy proposed in [34] and prevents two consecutive search directions from being almost opposite.

C. Quasi-Newton Methods

Quasi-Newton methods were developed based on Newton's optimization method in which the Hessian matrix $\mathbf{H}$ is substituted by a Hessian approximation $\mathbf{B}(k)$ to avoid the calculation of the exact Hessian matrix. This method generates the search direction $\mathbf{d}(k)$ by solving (30), and finds the step-length using a backtracking line search technique

$$\mathbf{B}(k)\mathbf{d}(k) + \mathbf{g}(k) = 0. \tag{30}$$

Currently, the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method is considered one of the most popular quasi-Newton methods. It uses information from the changes in weights and gradient in the process of computing iteratively the matrix $\mathbf{B}(k)$. To solve (30) for $\mathbf{d}(k)$, the inverse of $\mathbf{B}(k)$ needs to be computed. A common method to compute a matrix inverse is to apply the Sherman–Morrison formula, (31), twice [35]

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} - \frac{(\mathbf{A}^{-1}\mathbf{u})(\mathbf{v}^T\mathbf{A}^{-1})}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}. \tag{31}$$

The matrix $\mathbf{A}$ is considered to be nonsingular, and $\mathbf{u}$ and $\mathbf{v}$ are column vectors. For $(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1}$ to be nonsingular, the denominator of (31) is assumed to be nonzero. Hence, applying the previous formula twice yields an inverse BFGS formula [36]

$$\mathbf{B}^{-1}(k+1) = \mathbf{B}^{-1}(k) + \left(1 + \frac{\mathbf{y}^T(k)\mathbf{B}^{-1}(k)\mathbf{y}(k)}{\mathbf{y}^T(k)\mathbf{s}(k)}\right) \times \frac{\mathbf{s}(k)\mathbf{s}^T(k)}{\mathbf{s}^T(k)\mathbf{y}(k)} - \frac{\mathbf{B}^{-1}\mathbf{y}(k)\mathbf{s}^T(k) + (\mathbf{B}^{-1}(k)\mathbf{y}(k)\mathbf{s}^T(k))^T}{\mathbf{y}^T(k)\mathbf{s}(k)} \tag{32}$$

where $\mathbf{s}(k) = \mathbf{W}(k+1) - \mathbf{W}(k)$, and $\mathbf{y}(k) = \mathbf{g}(k+1) - \mathbf{g}(k)$. The initial estimate $\mathbf{B}(0)$ of the Hessian is usually taken to be the identity matrix $\mathbf{I}$.

Most of the studies on global convergence of the BFGS method are focused on convex minimization problems. Recently, variants of BFGS have been proposed for convex/nonconvex unconstrained optimization problems, some of which have been shown to be better than the original algorithm. Li [37] proposed a modification of the BFGS method, which he claims to possess a global convergence property, even without the convexity assumption, and under certain conditions has super-linear convergence. The modification proposed by Li is to compute $\mathbf{y}(k)$ as

$$\mathbf{y}(k) = \mathbf{g}(k+1) - \mathbf{g}(k) + t(k)\|\mathbf{g}(k)\|\mathbf{s}(k) \tag{33}$$

where

$$t(k) = 1 + \max\left\{-\frac{(\mathbf{g}(k+1) - \mathbf{g}(k))^T\mathbf{s}(k)}{\|\mathbf{s}(k)\|^2}, 0\right\}. \tag{34}$$

A backtracking line-search is used to compute the optimal step-length. On the other hand, Liao [38] proposed another modified version of the BFGS update. In his experimental results, Liao showed that his version of BFGS corrects for large eigenvalues more efficiently than the original BFGS in certain cases. His BFGS update is given by (35)–(38)

$$\mathbf{B}(k+1) = \mathbf{B}(k) - \delta(k)\frac{\mathbf{B}(k)\mathbf{s}(k)\mathbf{s}^T(k)\mathbf{B}(k)}{\mathbf{s}^T(k)\mathbf{B}(k)\mathbf{s}(k)} + \gamma(k)\frac{\mathbf{y}(k)\mathbf{y}^T(k)}{\mathbf{s}^T(k)\mathbf{y}(k)} \tag{35}$$

where

$$\rho(k) = \frac{\mathbf{s}^T(k)\mathbf{B}(k)\mathbf{s}(k)}{\mathbf{s}^T(k)\mathbf{B}(k)\mathbf{s}(k) + \mathbf{s}^T(k)\mathbf{y}(k)}$$

$$\sigma(k) = \frac{\mathbf{s}^T(k)\mathbf{y}(k)}{\mathbf{s}^T(k)\mathbf{B}(k)\mathbf{s}(k) + \mathbf{s}^T(k)\mathbf{y}(k)}$$

$$\delta(k) = \begin{cases} \rho(k), & \text{if } \rho(k) \geq \tau(k) \\ \tau(k), & \text{otherwise} \end{cases} \tag{36}$$

$$\gamma(k) = \begin{cases} \sigma(k), & \text{if } \rho(k) \geq \tau(k) \\ 1, & \text{otherwise} \end{cases} \tag{37}$$

and

$$\tau(k) = \exp\left(-\frac{10}{k^{1.02}}\right). \tag{38}$$

D. Levenberg–Marquardt Method and Its Variants

Levenberg–Marquardt (LM) optimization technique is one of the most popular and effective second-order algorithms for training feedforward neural networks. One of the characteristics of this method is that it combines the stability of gradient descent with the speed of Newton's algorithm. The LM learning expression is formulated as follows:

$$\triangle\mathbf{W}(k) = \mathbf{G}(k)^{-1}\mathbf{J}(k)\mathbf{e}(k) \tag{39}$$

where

$$\mathbf{G}(k) = \mathbf{J}^T(k)\mathbf{J}(k) + \lambda(k)\mathbf{I}. \tag{40}$$

Here $\mathbf{J}(k)$ denotes the Jacobian matrix, $\mathbf{G}(k)$ is an approximation to the Hessian matrix, $\lambda(k)$ is a regularization parameter which prevents $\mathbf{G}(k)$ from becoming an ill-conditioned matrix, $\mathbf{I}$ is the identity matrix, and $\mathbf{e}(k)$ is the error vector (details of how the Jacobian matrix is calculated with the standard backpropagation algorithm can be found in [39]). Equation (40) shows that when $\lambda(k) = 0$, the LM method behaves as the Gauss–Newton method, whereas for very large values of $\lambda(k)$ it tends to the gradient descent with a very small step-size. During the training process, the regularization parameter is incremented or decremented by a factor of ten; that is, whenever a computed step-size results in an increase in the network error, $\lambda(k)$ is divided by ten, and vice-versa. However, even with this adaptation of the regularization parameter, there are situations where the matrix $\mathbf{G}(k)$ becomes ill-conditioned. One possible way to avoid such situations from happening, in our case, is to constrain the regularization parameter to the range $[10^{10}, 10^{-10}]$; experiments show that keeping the regularization parameter in this range reduces the risk of running into an ill-conditioned $\mathbf{G}(k)$ matrix.

As the LM algorithm is a local optimization method, it is not guaranteed to converge to the global minimum of the objective function. Ampazis and Perantonis [40] proposed to add an adaptive momentum term, by formulating the training task as a constrained optimization problem whose solution effectively offers the necessary framework for incorporating the momentum term into the learning rule. The resulting weight update rule is given by

$$\Delta\mathbf{W}(k) = -\frac{\lambda_1}{2\lambda_2}[\mathbf{G}(k)]^{-1}\mathbf{g}(k) + \frac{1}{2\lambda_2}\Delta\mathbf{W}(k-1). \quad (41)$$

The constants λ_1 and λ_2 are Lagrange multipliers defined as

$$\lambda_1 = \frac{-2\lambda_2 Q(k) + I_{GF}}{I_{GG}}$$

and

$$\lambda_2 = \frac{1}{2}\left[\frac{I_{FF}I_{GG} - I_{GF}^2}{I_{GG}P(k)^2 - Q(k)^2}\right]^{\frac{1}{2}}$$

where

$$\mathbf{g}(k) = \mathbf{J}(k)\mathbf{e}(k) \quad (42)$$
$$I_{FF} = \Delta\mathbf{W}^T(k-1)\mathbf{G}(k)\Delta\mathbf{W}(k-1) \quad (43)$$
$$I_{GG} = \mathbf{g}^T(k)[\mathbf{G}(k)]^{-1}\mathbf{g}(k) \quad (44)$$
$$I_{GF} = \mathbf{g}^T(k)\Delta\mathbf{W}(k-1) \quad (45)$$
$$P(k) = 2\sqrt{I_{GG}} \quad (46)$$
$$Q(k) = -P(k)\sqrt{I_{GG} - \frac{I_{GF}^2}{I_{FF}}}. \quad (47)$$

The regularization parameter $\lambda(k)$ in (40) is decreased by a factor of ten when the following Wolfe condition holds:

$$f(\mathbf{W}(k) + \alpha(k)\mathbf{d}(k)) \le f(\mathbf{W}(k)) + c_1\alpha(k)\mathbf{g}(k)\mathbf{d}(k) \quad (48)$$

where c_1 is a constant in the interval $[0, 1]$, $f(\cdot)$ is the objective function. The previous inequality can be simplified as follows:

$$E(k+1) < E(k+1) + 0.1\mathbf{g}^T(k)\Delta\mathbf{W}(k). \quad (49)$$

In this case the constant c_1 is set to 0.1. However, if the previous condition does not hold the parameter $\lambda(k)$ is increased by the same factor until there is a reduction in the error function.

E. Least Squares Method

The least squares (LS) method has been used to train multilayer neural networks and was shown to converge faster than the gradient descent method [41], [42]. It is based on a direct determination of the matrices of weights by solving, in the LS sense, a set of systems of linear equations [43]. As the inputs and desired output are available for each output neuron, it is possible to calculate the weights $\mathbf{W}_{\mathrm{LH}}$ from the last hidden layer to the output layer using LS technique for linear equations. In this paper, this technique is combined with the proposed hybrid method as well as the LM algorithm.

1) Integration of LS Method With the Hybrid Method (QRPROPLS): The QRPROPLS method consists of two training modules. The first module updates all the weights in the network by applying (10) at every epoch. Then a network evaluation is conducted on the training set with the updated weights. The evaluation results are used by the hybrid method to compute the next step-size. The second training module is based on updating the weights in the final layer of the network. Having the actual outputs of the last hidden layer and the desired outputs, a LS method is used to further tune the weights in the final layer starting from the second epoch. One assumption that has to be made is that the number of training patterns must be greater than the number of weights in $\mathbf{W}_{\mathrm{LH}}$, which is the case for most practical pattern recognition problems. Another assumption is that the activation function of the output layer f is invertible. Under these assumptions, the problem becomes an over-determined system of linear equations

$$\mathbf{X}\mathbf{W}_{\mathrm{LH}} = f^{-1}(\mathbf{T}). \quad (50)$$

To simplify the derivation of the solution of (50), the final layer of the network is assumed to have a single neuron. In this case, $\mathbf{T}$ is a column vector of desired outputs, where each element corresponds to an input pattern, $\mathbf{X}$ is a matrix whose columns contain the actual outputs of the last hidden layer, and $\mathbf{W}_{\mathrm{LH}}$ is a column vector of weights to be computed. The general LS solution of (50) is given by the Moore–Penrose pseudo-inverse

$$\begin{aligned}\mathbf{W}_{\mathrm{LH}} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T f^{-1}(\mathbf{T}) \\ &= \mathbf{X}^+ f^{-1}(\mathbf{T}) \end{aligned} \quad (51)$$

where $\mathbf{X}^+ = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is the pseudoinverse of $\mathbf{X}$. As $\mathbf{X}$ is an $m \times n$ matrix where $m > n$, it can be factored, using singular value decomposition, as

$$\mathbf{X} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T \quad (52)$$

where $\mathbf{V}$ is $m \times m$ orthogonal matrix, $\mathbf{U}$ is $n \times n$ orthogonal matrix, and $\mathbf{\Sigma}$ is $m \times n$ diagonal matrix of the special form

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{D} & 0 \\ 0 & 0 \end{bmatrix} \text{ where } \mathbf{D} = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_v \end{bmatrix}.$$

The diagonal entries of $\mathbf{D}, \sigma_1, \ldots, \sigma_v$, are the singular values of $\mathbf{X}$, arranged in descending order $(\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_v \geq 0)$. The pseudoinverse of $\mathbf{X}, \mathbf{X}^+$, is given by

$$\mathbf{X}^+ = \mathbf{U}\boldsymbol{\Sigma}^+\mathbf{V}^T \tag{53}$$

where $\boldsymbol{\Sigma}^+$ is the $n \times m$ matrix

$$\boldsymbol{\Sigma}^+ = \begin{bmatrix} \mathbf{D}^{-1} & 0 \\ 0 & 0 \end{bmatrix}.$$

In certain cases, some columns of $\mathbf{X}$ are almost linearly dependent, which results in a matrix $\boldsymbol{\Sigma}$ that is almost, if not, rank deficient; that is some singular values of $\mathbf{X}$ become very small, resulting in a large disparity between singular values and, hence, ill-conditioning. One way to circumvent this problem is to use the truncated singular value decomposition method to generate a new matrix $\mathbf{X_k}$ by removing the contribution to the solution of the smallest singular values; that is all singular values smaller than a threshold are treated as zeroes

$$\mathbf{X_k} = \sum_{i=1}^{k} \mathbf{v}_i \sigma_i \mathbf{u}_i^T \text{ and } \mathbf{X_k^+} = \sum_{i=1}^{k} \mathbf{u}_i \sigma_i^{-1} \mathbf{v}_i^T \tag{54}$$

where $\mathbf{v}_i$ and $\mathbf{u}_i$ are the corresponding singular vectors of $\mathbf{V}$ and $\mathbf{U}$, respectively, and $k, (k < v)$, is the number of singular values of matrix $\mathbf{X_k}$ that are considered. The solution of $\mathbf{W}_{\text{LH}}$ is then given by

$$\mathbf{W}_{\text{LH}} = \sum_{i=1}^{k} \frac{\mathbf{u}_i \mathbf{v}_i^T}{\sigma_i} f^{-1}(\mathbf{T}_i). \tag{55}$$

Once the weights in the final layer are further adjusted, the output of the network is computed to monitor the training progress. The gradient vector is computed and used by the hybrid method to calculate the next step-size.

2) Integration of LS Method With the LM Method: The same strategy as explained previously is used to implement the LS method with the LM algorithm, except that now the weights are updated by the LM method in each epoch. From the second iteration onwards, the weights in the output layer are further updated by either (51) or (55) depending on the rank of X. After applying the LS method to the final layer, the output of the network is computed to monitor the training progress.

IV. Experimental Results

A. Classification Problem

The problem that has been chosen for this experiment is the visual pattern recognition task of face/nonface discrimination. This 2-D problem is much harder to solve in comparison to digit recognition; it is a stepping-stone for various applications, such as security access control, model-based video coding, content-based video indexing, and advanced human-computer interactions. Two experiments have been performed to investigate the classification accuracy and the convergence speed of the training algorithms described in the previous section. A small training set containing 50 face patterns and 50 nonface patterns (of size 24×24) has been used to test the convergence speed of the developed training algorithms. Some examples of the face and nonface patterns used in the training and test sets are shown in Fig. 4. The face patterns are taken from ECU[2] face and skin detection database [44], and the nonface patterns are collected using a bootstrap procedure. The desired output values of the network are 1 for a face and -1 for a nonface pattern. All images in the dataset were converted to gray-scale, histogram equalized to improve the image contrast, and finally scaled to the range $[-1, 1]$.

[2]ECU stands for Edith Cowan University, Perth, Australia.

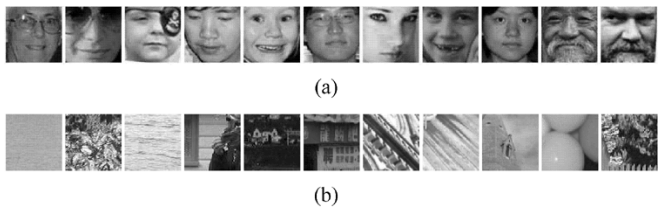


Fig. 4. Examples of face and nonface patterns.

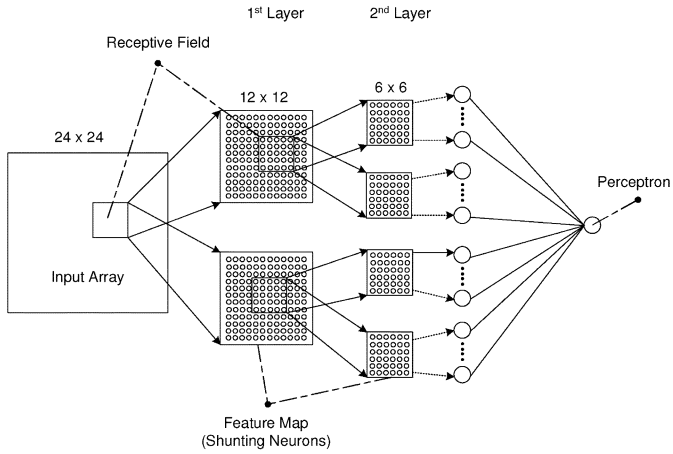


Fig. 5. Three-layer binary connected network.

B. Network Structure and Initialization Process

The network architecture that has been used for comparing the different training algorithms is shown in Fig. 5. It is a three-layer network with two feature maps in the first hidden layer, four feature maps in the second hidden layer, and one output neuron. After preliminary tests on different combinations of activation functions and a comparison of performance based on classification rate, the activation functions in the first hidden layer were chosen to be hyperbolic-tangent/exponential functions, whereas the logarithmic sigmoid/exponential were the activation functions in the second hidden layer, and a linear function in the output layer. Receptive fields of size 5×5 are used throughout the network. All the shunting neurons in the feature maps have their own bias parameters as well as their own passive decay rate constants; the network has a total of 1633 free parameters. In the initialization phase, the weights of each feature map were initialized to uniformly distributed random value in the range $[-1, 1]$; they were then divided by the width of the receptive field, i.e., 5. The bias parameters b and d were initialized similarly, but without scaling. Furthermore, the passive decay rate parameter a was initialized in the range [0, 1] and was constrained to satisfy the condition given in (5).

TABLE II
SUMMARY OF NETWORK CONVERGENCE RESULTS: S.T. SUCCESSFUL TRIALS OUT OF 50, MAX. MAXIMUM TRAINING EPOCHS, MIN. MINIMUM TRAINING EPOCHS, AVER. AVERAGE TRAINING EPOCHS, STD. STANDARD DEVIATION, n NUMBER OF WEIGHTS, P NUMBER OF TRAINING PATTERNS AND N_L NUMBER OF INPUTS TO THE OUTPUT LAYER

Training	Approximate	Binary-Connection					Toeplitz-Connection					Full-Connection				
Algorithms	Memory Usage	S.T	Max.	Min.	Aver.	Std.	S.T	Max.	Min.	Aver.	Std.	S.T	Max.	Min.	Aver.	Std.
QRPROP	$6n$	50	42	17	25	5	50	43	19	26	5	50	41	18	25	5
QRPROPLS	$6n + P \times N_L$	50	8	3	5	1	50	9	3	6	1	50	9	4	6	1
FR	$3n$	50	59	15	25	7	50	212	17	29	27	50	55	13	25	9
PR	$3n$	50	35	11	20	5	50	34	12	22	6	50	85	13	24	11
HS	$3n$	50	163	12	31	34	50	258	12	39	45	50	100	15	29	17
DY	$3n$	50	64	15	30	10	50	114	15	34	18	50	77	15	38	18
HY-HuSt	$3n$	50	30	12	20	4	50	56	14	22	7	50	41	13	22	6
HY-GN	$3n$	50	45	14	22	7	50	62	12	25	12	50	44	13	22	6
HY-DY	$3n$	50	180	12	30	35	50	207	13	37	42	50	137	13	36	28
HY-DY-V1	$3n$	50	69	12	23	12	50	170	14	25	23	50	140	13	31	24
BFGS	$3n + n^2$	50	26	11	14	3	50	46	11	17	6	50	77	11	24	15
BFGS-Li	$3n + n^2$	50	71	9	16	9	50	41	11	15	4	50	73	11	20	13
BFGS-Liao	$3n + n^2$	50	31	11	14	3	50	49	11	16	6	50	59	11	18	9
LM	$P \times n + n^2 + n$	50	46	5	9	6	50	43	5	9	7	50	42	5	10	7
OLMAM	$P \times n + n^2 + 2n$	50	19	3	6	3	50	12	3	5	1	50	12	3	5	1
LMLS	$P \times n + n^2 + P \times N_L$	50	4	2	3	1	50	4	2	3	0	50	4	2	3	1

C. Evaluation Procedure for Training Algorithms

To evaluate the convergence speed of the aforementioned training algorithms, 50 networks of the same structure, but different initial weights, were generated for each algorithm. In this particular classification problem, a training process is considered to be successful when the MSE reaches 0.1. On the other hand, a training process fails to converge when it reaches the maximum training time before reaching the desired MSE. The training time of an algorithm is defined as the number of epochs required to meet the stopping criterion. In these experiments, the maximum training time was set to 500 epochs. To compare the performances of the various algorithms, the following variables have been recorded for each training algorithm: the maximum training time Max., the minimum training time Min., the average training time Aver., the standard deviation of the training time Std., and the number of successful trials out of 50 runs, S.T. If a training trial fails, it is not included in the statistics; however, the result is reflected in the number of successes out of 50 trials. In addition, information about the memory storage required for each training algorithm is also included. These expressions are based on the parameters which required by the training algorithm to compute a weight update, for example, the gradient vector, the Hessian approximation, named a few.

D. Convergence Speed

Sixteen different training algorithms were implemented and tested; they are listed in the following:

- two hybrid methods based on *Quickprop*, *Rprop*, and *SuperSAB* with or without LS method: QRPROP and QRPROPLS;
- eight variants of the conjugate gradient methods: FR, PR, HS, DY, HY-HuSt, HY-GN, HY-DY, and HY-DY-V1;
- three quasi-Newton methods: BFGS, MBFGS-Li [37], and MBFGS-Liao [38];
- three versions of the LM algorithm: standard LM algorithm, optimized LM with adaptive momentum (OLMAM), and LM with LS method (LMLS).

Table II presents the statistical data for the convergence speed of the 16 training algorithms tested on SICoNNets with their three connection schemes. The second column of the table illustrates an approximation of the amount of memory used by each training algorithm. All the sixteen training algorithms met the stopping criterion $(\text{MSE} = 0.1)$, at different training times, in all 50 training trials, and none of them failed to converge. In this experiment, the line search method proposed by Charalambous [45] is used in the previous conjugate gradient methods as it is the most widely used in the Neural Network Matlab Toolbox and has produced excellent results for many different problems. Although, it requires an additional computation of the gradient, it achieves adequate reductions in the error function with fewer number of iterations. The criteria that are used to stop the line search are based on Wolfe condition (48) and the curvature condition

$$|\mathbf{g}(X(k) + \alpha(k)\mathbf{d}(k))^T\mathbf{d}(k)| \leq c_2|\mathbf{g}(k)^T\mathbf{d}(k)|. \qquad (56)$$

These two conditions are known as the strong Wolfe conditions. The values of c_1 and c_2 are set to 0.001 and 0.1, respectively. Among the four standard conjugate gradient methods, the PR method has the best performance; those that have the poorest results are DY and HS methods. The hybrid conjugate gradient methods such as HY-HuSt and HY-GN produce better results compared to HY-DY and HY-DY-V1 methods. However, the experimental results of the hybrid method HY-DY-V1 confirm these authors' claim that (29) has better convergence speed than (28). Among the eight conjugate gradient variants, HY-HuSt has the best convergence speed as it inherits the properties of PR and FR methods. The second best conjugate gradient is the PR method with a training time of 22 epochs on average.

In quasi-Newton methods, the modified BFGS algorithms perform slightly better than the original in terms of average training time. This finding supports the claims made in [37] and [38]. As the BFGS update proposed by Liao has a better "self-correcting" property by correcting for large eigenvalues more efficiently, it produces better result in comparison to the original BFGS and the modified BFGS proposed by Li. The quasi-Newton methods require slightly more operations to calculate an iterate and require more memory to store the matrix $\mathbf{B}(k)$, but these additional costs are outweighed by the advantage of better convergence, compared to conjugate gradient methods; on average there is a reduction of 10 epochs between these two groups of training algorithms. The variant of LM algorithm developed by Ampazis and Perantonis performs better than the original LM algorithm at the expense of additional memory for storing the previous weight update. The maximum number of epochs reached by LM algorithm is 46, whereas the maximum training time used by OLMAM method is 19 epochs.

During the experiment when the hybrid technique with and without LS method were used to train the networks in which the shunting neurons in a feature map have their own biases and passive decay rate constants, we found that the denominator of (3) became very large after a certain number of training iterations because of the exponential activation function. Consequently, the training process stopped. Therefore, for these particular training methods, the bias parameter d has been bounded above by 10 during the training. Nevertheless, our experiments based on a small training set show that the proposed hybrid technique, QRPROPLS, converges slightly faster than some second-order training methods, such as the LM algorithm, and it is definitely more efficient than the LM algorithm in terms of computation load and memory storage. For example, the module that requires the most memory storage in QRPROPLS is the matrix $\mathbf{X}$ of size $P \times N_L$, where N_L is the number of inputs to the final layer. However, the LM algorithm requires the computation of the Jacobian matrix $\mathbf{J}(k)$ and the approximated Hessian matrix $\mathbf{G}(k)$, which are of size $N_J \times n$ and $n \times n$ (where $N_J = P \times M$, P is the number of training patterns, M is the number of output neurons, and n is the total number of weights), respectively. As a numerical example, for a three-layer network with an output neuron having 37 inputs and a training set of 100 samples, the matrix $\mathbf{X}$ contains 3700 elements, whereas the matrices $\mathbf{J}(k)$ and $\mathbf{G}(k)$ have, respectively, 163300 and 2666689 elements. It is clear that QRPROPLS uses much less memory storage compared to the LM algorithm. In terms of computation cost, the QRPROPLS method needs only two network evaluations to update the weights in each epoch, whereas the LM method requires two or more network evaluations to determine the regularization parameter. However, the QRPRPLS method requires several thresholds to be tuned, such as the limits for the step-size, the adaptive momentum rate and the adaptive learning rate. In general, the threshold values used in our experiments produce good results. Even though the LMLS method has the fastest convergence rate, it requires a huge amount of memory storage to compute the Jacobian matrix of the error function, and requires the inversion of matrices with dimensions equal to the number of free network parameters.

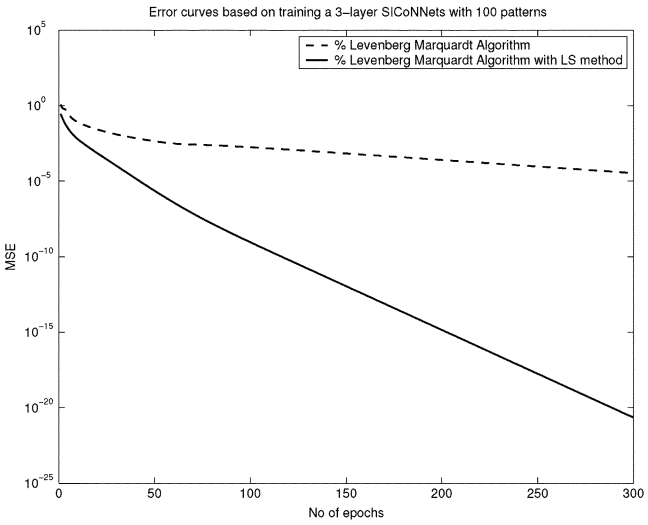


Fig. 6. (a) Convergence speed between LM algorithms with/without a LS method. (b) Convergence speed between the first-order hybrid techniques with/without a LS method.

Further adapting the weights in the output layer of the proposed network with a LS method enhances the convergence speed of the training method. For example, the difference between LM and LMLS algorithms is six epochs on average across all three networks. The results obtained with QRPROPLS also supports the same claim, whereas in this case there is a significant improvement in the training method, i.e., 19 epochs. Fig. 6(a) and (b) shows the time evolution of the MSE, as a function of the training epochs, for four training algorithms: QRPROP, QRPROPLS, LM, and LMLS. Note that all four algorithms used the same initialized network and the same training dataset. The curves in these figures illustrate that there is an improvement in convergence speed of both training algorithms when combined with the LS method.

E. Classification Accuracy and Generalization Ability

Another experiment was performed to demonstrate the classification accuracy and generalization ability of the proposed network architectures. The ten-fold cross-validation technique,

TABLE III
AVERAGE CLASSIFICATION RATES AND ERROR RATES OF THE THREE CONNECTION SCHEMES OBTAINED USING TEN-FOLD CROSS-VALIDATION

Training Algorithm	Binary connection			Toeplitz connection			Full connection			Aver. E rate of 3 Nets
	Aver. Classif. rates (%)		Av. E	Aver. Classif. rate (%)		Aver. E	Aver. Classif. rate (%)		Aver. E	
	Face	Nonface	rate (%)	Face	Nonface	rate (%)	Face	Nonface	rate (%)	
QRPROP	95.70	95.20	4.55	94.90	96.20	4.45	95.50	96.40	4.05	**4.35**
QRPROPLS	97.40	97.30	2.65	96.90	97.90	2.60	97.10	97.00	2.95	**2.73**
FR	95.30	96.70	4.00	94.90	95.20	4.95	88.40	91.60	10.00	**6.32**
PR	92.70	93.00	7.15	90.30	92.60	8.55	90.60	87.80	10.80	**8.83**
HS	95.90	97.10	3.50	95.00	96.10	4.45	96.50	95.70	3.70	**3.95**
DY	96.80	96.10	3.55	95.30	96.10	4.30	95.20	96.10	4.35	**4.07**
HY-HuSt	91.50	93.20	7.65	91.20	92.60	8.10	91.80	90.70	8.75	**8.17**
HY-GN	92.90	95.40	5.85	91.20	93.40	7.70	90.80	91.10	9.05	**7.53**
HY-DY	96.40	96.70	3.45	95.50	96.80	3.85	94.90	97.10	4.00	**3.77**
HY-DY-V1	95.90	96.80	3.65	95.50	97.40	3.55	96.00	97.20	3.40	**3.53**
BFGS	95.20	96.30	4.25	96.30	95.50	4.10	95.10	94.10	5.40	**4.58**
BFGS-Li	96.10	97.50	3.20	97.70	97.70	2.30	97.60	97.20	2.60	**2.70**
BFGS-Liao	97.30	97.50	2.60	97.30	97.40	2.65	96.70	97.40	2.95	**2.73**
LM	97.80	97.70	2.25	97.30	96.90	2.90	96.70	97.80	2.75	**2.63**
OLMAM	98.20	97.60	2.10	97.60	98.50	1.95	97.30	98.30	2.20	**2.08**
LMLS	97.20	98.00	2.40	97.50	97.80	2.35	96.30	97.60	3.05	**2.60**
Average	**95.77**	**96.38**	**3.93**	**95.28**	**96.13**	**4.30**	**94.78**	**95.19**	**5.01**	

which is usually used to estimate the generalization error of a given model, has been used to train and test the three SICoNNet architectures with all 16 training algorithms. A dataset consists of 2000 face patterns and 2000 nonface patterns was used, and in each fold 3600 patterns were used for training and 400 patterns for testing. The input patterns (of size 24×24 pixels) were preprocessed with histogram equalization and linearly scaled to the range $[-1, 1]$. Furthermore, the bias parameters and the passive decay rate constant are shared by the neurons in the feature map. In other words, all the shunting neurons in the feature map share the same set of incoming weights as well as the bias parameters and the passive decay rate constant. This means that the number of trainable weights in the network is now reduced to 355, which could improve the generalization. Training is terminated if either the number of iterations reaches 200 or the MSE is smaller than or equal to 0.1. The threshold value to separate face and nonface patterns at the network output is chosen so that the total classification error is at a minimum.

Table III presents the classification rates and error rates of the three proposed network architectures. The first column lists the training algorithms, and the remaining columns list the classification rates for face and nonface patterns and the error rates of the three architectures: binary-, Toeplitz- and fully connected networks. The last column gives the average error rates for each training algorithm, across all three networks, and the last row presents the average classification and error rates of each network, across all training algorithms. It is clear from the results presented in the last column of Table III that the best training algorithms, in terms of average error rates, are the OLMAM, LMLS, LM, BFGS-Li, BFGS-Liao, and QRPROPLS; they all achieve error rate of less than 3% averaged across all three networks. The OLMAM, LM, LMLS, BFGS-Liao, and QRPROPLS algorithms perform consistently well with all networks (their error rates are less than 3%), whereas the performance of the BFGS-Li algorithm is lightly worse for the binary-connected network (error rate $= 3.2\%$). While the proposed hybrid method QRPROP is not among the best performing training algorithms, it nevertheless outperforms some of the conjugate gradient methods and the standard BFGS quasi-Newton method. However, the performance of the hybrid method improves significantly when combined with the LS method to the point where it becomes competitive with LM algorithms; there is a reduction of 1.62% in average error rate across all network architectures. The results in Table III also show that the binary- and Toeplitz-connected networks outperform slightly the fully connected network: their average error rates across all training algorithms are 3.93% for binary-connected, 4.30% for the Toeplitz-connected, and 5.01% for the fully connected network. However, all three network architectures achieve good classification accuracy, with different training algorithms: the lowest error rates for each architecture are 1.95% for Toeplitz-connected, 2.1% for the binary-connected, and 2.2% for the fully connected network.

In general, all the developed training algorithms work well with less than 10% of false alarm rate. Overall, OLMAM method is the best training algorithm in terms of classification accuracy, but it requires a lot of computation and memory storage. Our proposed hybrid training method is considered among the best five algorithms, and it uses much less memory storage compared to the OLMAM algorithm.

V. CONCLUSION

In this paper, several optimization methods with necessary modifications have been implemented for training a new class of convolutional neural networks (SICoNNets). Three network architectures based on three different connection strategies (binary-, Toeplitz- and fully connected) have been trained and tested. Training algorithms for this type of networks are quite easy to implement due to the systematic connection strategies and their simple network architectures. Experimental results, based on a face discrimination task, show that all implemented

algorithms can be used to train the proposed CoNN architectures. A correct face classification rate of 98.20% (with 2.10% false alarm) is achieved with a two-layer binary-connected network trained using the OLMAM algorithm. Furthermore, a new hybrid training method has been proposed, based on the combination of three existing first-order methods (*Quickprop*, *Rprop*, *SuperSAB*) and the LS method. Experimental results indicate that the new hybrid training method has similar performance to the LM algorithm, but much lower computational cost and memory storage. As the new hybrid method requires only gradient information, it is very suitable for training networks with large number of free parameters. In general, we can conclude that when the weights of the output layer are further adapted with the LS method, there is improvements in both classification accuracy and convergence speed of training.

APPENDIX
DERIVATION OF ERROR BACKPROPAGATION

Nomenclature

$Z_{L,k}$	Output of the kth feature map of the Lth layer.
S_{L-1}	Number of feature maps of the $(L-1)$th layer.
S_F	Size of the feature map.
f_L, g_L	Activation functions.
$a_{L,k}(i,j)$	Passive decay rate for a neuron at the position (i,j) in the kth feature map of the Lth layer.
$b_{L,k}(i,j), d_{L,k}(i,j)$	Bias parameters.
$C_{L,k}(x,y), D_{L,k}(x,y)$	Set of weights at position (x,y)
ℓ	Size of the matrix weights $C_{L,k}(x,y)$ and $D_{L,k}(x,y)$.

This section presents the derivation of the error-backpropagation for the proposed CoNN architecture with a full-connection scheme.

Let $Z_{L,k}(i,j)$ be the output of the kth feature map of the Lth layer

$$Z_{L,k}(i,j) = \frac{g_L(Y_{L,k}(i,j))}{a_{L,k}(i,j) + f_L(X_{L,k}(i,j))} \qquad \forall i,j = 1,\ldots,S_F. \tag{A.1}$$

$$Y_{L,k}(i,j) = \sum_{v=1}^{S_{L-1}} \sum_{x,y=-\frac{\ell-1}{2}}^{\frac{\ell-1}{2}} C_{L,k} Z_{L-1,v}(2i+x, 2j+y) + b_{L,k}(i,j)$$

and

$$X_{L,k}(i,j) = \sum_{v=1}^{S_{L-1}} \sum_{x,y=-\frac{\ell-1}{2}}^{\frac{\ell-1}{2}} D_{L,k} Z_{L-1,v}(2i+x, 2j+y) + d_{L,k}(i,j).$$

Note that (A.1) is obtained from (3) by rotating the convolution masks $C_{L,k}$ and $D_{L,k}$ by 180° before performing convolution. This is done for the sake of implementation efficiency. The computed kth feature map of the Lth layer has a quarter the size of those of the previous layer. The output of the ith perceptron of the output layer N is calculated as

$$Z_{N,i} = h\left(\sum_{r=1}^{S_{N-1}} w_{N,ir} Z_{N-1,r} + b_{N,i}\right). \tag{A.2}$$

VI. CALCULATING THE UNNORMALIZED ERROR SENSITIVITIES OF THE ENTIRE NETWORK

Assuming the error function to be the sum squared error, the network to have N layers (including the output layer), and the output layer to consist of S_N neurons, then for each input pattern we have the error given by

$$E = \frac{1}{2}\sum_{i=1}^{S_N}(e_{N,i})^2 = \frac{1}{2}\sum_{i=1}^{S_N}(Z_{N,i} - t_{N,i})^2 \tag{A.3}$$

where $t_{N,i}$ is the target output, $Z_{N,i}$ is the actual output of the ith perceptron, and $e_{N,i}$ is the error. The partial derivative of the sum squared error with respect to the output (un-normalized error sensitivity) is

$$\delta_{N,i} = \frac{\partial E}{\partial Z_{N,i}} = Z_{N,i} - t_{N,i}. \tag{A.4}$$

In the last hidden layer, the un-normalized error sensitivities are given by

$$\delta_{N-1,i} = \sum_{j=1}^{S_N} \delta_{N,j} h'\left(\sum_{r=1}^{S_{N-1}} w_{N,jr} Z_{N-1,r} + b_{N,j}\right) w_{N,ji} \qquad \text{for } i = 1,\ldots,S_{N-1} \tag{A.5}$$

where $w_{N,jr}$ is the synaptic weight from the rth neuron of the $(N-1)$th layer to the jth neuron of the Nth layer, and S_{N-1} is the number of outputs generated after the local averaging at the $(N-1)$th layer. The prime on h function signifies differentiation with respect to the argument. As the $(N-1)$th layer is made up of feature maps, the computed error sensitivities $\delta_{N-1,i}$ have to be rearranged into a matrix form, by duplicating each error sensitivity into four, as shown in Fig. 7. For other layers $L < N-1$, the error sensitivity for the vth feature map at position (m,n) is computed as

$$\delta_{L,v}(m,n) = \frac{\partial E}{\partial Z_{L,v}(m,n)} = \sum_{k=1}^{S_{L+1}} \sum_{x,y=-\frac{(\ell-1)}{2}}^{\frac{(\ell-1)}{2}} \delta_{L+1,k}(i,j) \frac{\partial Z_{L+1,k}(i,j)}{\partial Z_{L,v}(m,n)} \qquad \forall m,n = 1,\ldots,S_F. \tag{A.6}$$

As the error sensitivity $\delta_{L+1,k}(i,j)$ and the computed output $Z_{L+1,k}(i,j)$ of the $(L+1)$th layer have a quarter the size of the feature map of the preceding layer, they have to be up-sampled by adding odd columns and rows that contain zeros until they have the same size as the feature map of the Lth layer. So, we now have

$$\delta_{L,v}(m,n) = \sum_{k=1}^{S_{L+1}} \sum_{x,y=-\frac{(\ell-1)}{2}}^{\frac{(\ell-1)}{2}} \tilde{\delta}_{L+1,k}(2i,2j) \frac{\partial \tilde{Z}_{L+1,k}(2i,2j)}{\partial Z_{L,v}(m,n)} \tag{A.7}$$

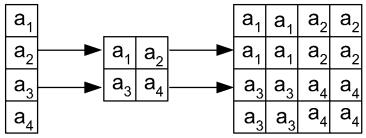


Fig. 7. Arrangement of un-normalized error sensitivities into a matrix.

where the first and second terms on the right-hand side (RHS) of (A.7) are given by

$$\tilde{\delta}_{L+1,k}(2i,2j) = \begin{cases} 0, & \text{if } i \text{ or } j \text{ is odd} \\ \delta_{L+1,k}(i,j), & \text{if } i \text{ and } j \text{ are even} \end{cases} \tag{A.8}$$

$$\tilde{Z}_{L+1,k}(2i,2j) = \begin{cases} 0, & \text{if } i \text{ or } j \text{ is odd} \\ Z_{L+1,k}(i,j), & \text{if } i \text{ and } j \text{ are even} \end{cases}. \tag{A.9}$$

The arguments i and x are related by $2i + x = m$, and the arguments j and y are related by $2j + y = n$, which implies that $i = (m-x)/2$ and $j = (n-y)/2$. Hence, we can write

$$\tilde{\delta}_{L+1,k}(2i,2j) = \tilde{\delta}_{L+1,k}(m-x,n-y) \tag{A.10}$$

and

$$\frac{\partial \tilde{\delta}_{L+1,k}(2i,2j)}{\partial Z_{L,v}(m,n)} = \frac{\partial \tilde{Z}_{L+1,k}(m-x,n-y)}{\partial Z_{L,v}(m,n)}. \tag{A.11}$$

Then, (A.7) becomes

$$\delta_{L,v}(m,n) = \sum_{k=1}^{S_{L+1}} \sum_{x,y=\frac{-(\ell-1)}{2}}^{\frac{(\ell-1)}{2}} \tilde{\delta}_{L+1,k}(m-x,n-y) \times \frac{\partial \tilde{Z}_{L+1,k}(m-x,n-y)}{\partial Z_{L,v}(m,n)}. \tag{A.12}$$

Now, the second term on the RHS of (A.12) has to be simplified. From the definition of (A.1), the partial derivative of $Z_{L+1,k}(i,j)$ with respect to $Z_{L,v}(2i+x,2j+y)$ is

$$\frac{\partial Z_{L+1,k}(i,j)}{\partial Z_{L,v}(2i+x,2j+y)} = \frac{C_{L+1,k}(x,y) g'_{L+1}(Y_{L+1,k}(i,j))}{a_{L+1,k}(i,j) + f_{L+1}(X_{L+1,k}(i,j))} - \frac{f'_{L+1}(X_{L+1,k}(i,j)) D_{L+1,k}(x,y) Z_{L+1,k}(i,j)}{a_{L+1,k}(i,j) + f_{L+1}(X_{L+1,k}(i,j))}. \tag{A.13}$$

Up-sampling the results generated in layer $L+1$th in (A.13) by two, and replacing i and j in terms of m, x and n, y, respectively, we compute the error sensitivity in the vth feature map of the Lth layer at (m,n) as

$$\delta_{L,v}(m,n) = \sum_{k=1}^{S_{L+1}} [C_{L+1,k} * A_{L+1,k}]_{(m,n)} - [D_{L+1,k} * B_{L+1,k}]_{(m,n)} \tag{A.14}$$

where

$$A_{L+1,k}(m,n) = \frac{g'_{L+1}(Y_{L+1,k}(m,n)) \delta_{L+1,k}(m,n)}{a_{L+1,k}(m,n) + f_{L+1}(X_{L+1,k}(m,n))} \tag{A.15}$$

$$B_{L+1,k}(m,n) = \frac{f'_{L+1}(X_{L+1,k}(m,n)) \delta_{L+1,k}(m,n) Z_{L+1,k}(m,n)}{a_{L+1,k}(m,n) + f_{L+1}(X_{L+1,k}(m,n))}. \tag{A.16}$$

Calculating the Gradient of E With Respect to the Arguments

At output layer N, the partial derivative of E with respect to $w_{N,ij}$ is

$$\frac{\partial E}{\partial w_{N,ij}} = \delta_{N,i} \frac{\partial Z_{N,i}}{\partial w_{N,ij}} = \delta_{N,i} h' \left(\sum_{r=1}^{S_{N-1}} w_{N,ir} Z_{N-1,r} + b_{N,i} \right) Z_{N-1,j}. \tag{A.17}$$

For other layers $L < N$, the partial derivative of E with respect to $C_{L,k}(x,y)$ is

$$\frac{\partial E}{\partial C_{L,k}(x,y)} = \sum_{i,j} \delta_{L,k}(i,j) \frac{\partial Z_{L,k}(i,j)}{\partial C_{L,k}(x,y)} \tag{A.18}$$

$$\frac{\partial E}{\partial C_{L,k}(x,y)} = \sum_{i,j} \sum_{v=1}^{S_{L-1}} Z_{L-1,v}(2i+x,2j+y) A_{L,k}(i,j) \tag{A.19}$$

where $A_{L,k}(i,j)$ is given in (A.15) and $-(\ell-1)/2 \le x,y \le (\ell-1)/2$. The partial derivative of E with respect to $D_{L,k}(x,y)$ is

$$\frac{\partial E}{\partial D_{L,k}(x,y)} = \sum_{i,j} \delta_{L,k}(i,j) \frac{\partial Z_{L,k}(i,j)}{\partial D_{L,k}(x,y)} \tag{A.20}$$

$$\frac{\partial E}{\partial D_{L,k}(x,y)} = -\sum_{i,j} \sum_{v=1}^{S_{L-1}} Z_{L-1,v}(2i+x,2j+y) B_{L,k}(i,j) \tag{A.21}$$

where $B_{L,k}(i,j)$ is given in (A.16). At the output layer N, the partial derivative of E with respect to $b_{N,i}$ is

$$\frac{\partial E}{\partial b_{N,i}} = \delta_{N,i} \frac{\partial Z_{N,i}}{\partial b_{N,i}}, = \delta_{N,i} h' \left(\sum_{r=1}^{S_{N-1}} w_{N,ir} Z_{N-1,r} + b_{N,i} \right). \tag{A.22}$$

For other layers $L < N$, the partial derivative of E with respect to $b_{L,k}(i,j)$ is computed as

$$\frac{\partial E}{\partial b_{L,k}(i,j)} = \delta_{L,k}(i,j)\frac{\partial Z_{L,k}(i,j)}{\partial b_{L,k}(i,j)} = A_{L,k}(i,j). \quad \text{(A.23)}$$

The partial derivative of E with respect to $d_{L,k}(i,j)$ is

$$\frac{\partial E}{\partial d_{L,k}(i,j)} = \delta_{L,k}(i,j)\frac{\partial Z_{L,k}(i,j)}{\partial d_{L,k}(i,j)} = -B_{L,k}(i,j). \quad \text{(A.24)}$$

The partial derivative of E with respect to $a_{L,k}(i,j)$ is expressed as

$$\frac{\partial E}{\partial a_{L,k}(i,j)} = \delta_{L,k}(i,j)\frac{\partial Z_{L,k}(i,j)}{\partial a_{L,k}(i,j)} = \delta_{L,k}(i,j)\frac{-Z_{L,k}(i,j)}{a_{L,k}(i,j) + f_L\left(X_{L,k}(i,j)\right)}. \quad \text{(A.25)}$$

REFERENCES

[1] K. Fukushima, "Cognitron: A self organizing multilayered neural network," *Biol. Cybern.*, pp. 121–136, 1975.

[2] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jacket, "Handwritten digit recognition with a back-propa-gation neural network," in *Advances in Neural Information Processing Systems*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufman, 1990, vol. 2, pp. 396–404.

[3] D. R. Lovell, "The neocognitron as a system for handwritten character recognition limitations and improvements," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Queensland, Australia, 1994.

[4] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural network approach," *IEEE Trans. Neural Netw.*, vol. 8, no. 1, pp. 98–113, Jan. 1997.

[5] B. Fasel, "Robust face analysis using convolutional neural networks," in *Proc. 16th Int. Conf. Pattern Recognition*, vol. 2, 2002, pp. 11–15.

[6] S. Behnke, "Face localization in the neural abstraction pyramid," in *Proc. 7th Int. Conf. Knowledge-Based Intelligent Information and Engineering Systems (KES'03)*, vol. 2, pp. 139–145.

[7] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 826–834, 1983.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[9] F. Tivive and A. Bouzerdoum, "A new class of convolutional neural network (siconnets) and their application to face detection," in *Proc. Int. Joint Conf. Neural Networks*, vol. 3, 2003, pp. 2157–2162.

[10] S. Grossberg, Ed., *Neural Networks and Natural Intelligence*. Cambridge, MA: MIT Press, 1988.

[11] R. B. Pinter, "Adaptation of spatial modulation transfer function via nonlinear lateral inhibition," *Biol. Cybern.*, vol. 51, pp. 285–291, 1985.

[12] ——, "Adaptation of receptive field organization by multiplicative lateral inhibition," *J. Theor. Biol.*, vol. 110, pp. 435–444, 1984.

[13] A. Bouzerdoum, "The elementary movement detection mechanism in insect vision," *Phil. Tram. R. Soc. Lond*, vol. B-339, pp. 375–384, 1993.

[14] A. Bouzerdoum and R. B. Pinter, "Nonlinear lateral inhibition applied to motion detection in the fly visual system," in *Nonlinear Vision*, R. B. Pinter and B. Nabet, Eds. Boca Raton, FL: CRC, 1992, pp. 423–450.

[15] ——, "Shunting inhibitory cellular neural networks: Derivation and stability analysis," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 40, no. 3, pp. 215–221, Mar. 1993.

[16] A. Bouzerdoum, "A new class of high-order neural networks with nonlinear decision boundaries," in *Proc. 6th Int. Conf. Neural Information Processing*, vol. 3, Perth, Australia, 1999, pp. 1004–1009.

[17] ——, "Classification and function approximation using feed-forward shunting inhibitory artificial neural networks," in *Proc. Int. Joint Conf. Neural Networks*, 2000, pp. 613–618.

[18] G. Arulampalam and A. Bouzerdoum, "A generalized feedforward neural network architecture for classification and regression," *Neural Netw.*, vol. 16, pp. 561–568, 2003.

[19] ——, "Application of shunting inhibitory artificial neural networks to medical diagnosis," in *Proc. 7th Australian and New Zealand Intelligent Information Systems Conf. (ANZIIS'01)*, Perth, Western Australia, 2001, pp. 89–94.

[20] ——, "Expanding the structure of shunting inhibitory artificial neural network classifiers," in *Proc. Int. Joint Conf. Neural Networks*, vol. 3, 2002, pp. 2855–2860.

[21] C. Garcia and M. Delakis, "A neural architecture for fast and robust face detection," in *Proc. 16th Int. Conf. Pattern Recognition (ICPR'02)*, vol. 2, Quebec, Canada, 2002, pp. 20044–20048.

[22] C. Neubauer, "Evaluation of convolutional neural network for visual recognition," *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 685–696, Jul. 1998.

[23] D. E. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing-Explorations in the Microstructure of Cognition*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, vol. 1, pp. 318–362. Foundations.

[24] M. Riedmiller, "Advanced supervised learning in multilayer perceptions—From backpropagation to adaptive learning algorithms," *Comput. Standards Interfaces*, no. 5, pp. 265–278, 1994.

[25] S. Fahlman, "An empirical study of learning speed in backpropagation networks," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS 88-162, 1988.

[26] T. Tollenaere, "Supersab: Fast adaptive bp with good scaling properties," *Neural Netw.*, vol. 3, pp. 561–573, 1990.

[27] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *Comput. J.*, vol. 7, pp. 149–154, 1964.

[28] E. Polak and G. Ribiére, "Note sur la convergence de methodes de directions conjugees," *Revue Francaise d'Informatique et de Recherche Operationnelle*, vol. 3, pp. 35–43, 1969.

[29] M. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *J. Res. Natl. Bur. Stand.*, vol. 49, pp. 409–436, 1952.

[30] Y. H. Dai and Y. Yuan, "A nonlinear conjugate gradient method with a strong global convergence property," in *SIAM J. Optim.*, vol. 10, 1999, pp. 177–182.

[31] Y. F. Hu and C. Storey, "Global convergence result for conjugate gradients methods," *J. Optim. Theory Appl.*, vol. 71, pp. 399–405, 1991.

[32] J. C. Gilbert and J. Nocedal, "Global convergence properties of conjugate gradient methods for optimization," in *SIAM J. Optim.*, vol. 2, 1992, pp. 21–42.

[33] Y. H. Dai and Y. Yuan, "An efficient hybrid conjugate gradient method for unconstrained optimization," *Ann. Oper. Res.*, no. 103, pp. 33–47, 2001.

[34] M. J. D. Powell, "Nonconvex minimization calculations and the conjugate gradient method," in *Lecture Notes in Mathematics*. Berlin, Germany: Springer-Verlag, 1984, pp. 122–141.

[35] A. S. Householder, *The Theory of Matrices in Numerical Analysis*. New York: Dover, 1975.

[36] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization*. New York: Wiley, 1996.

[37] D. H. Li and M. Fukushima, "A modified BFGS method and its global convergence in nonconvex minimization," Dept. Appl. Math. Phys., Kyoto Univ., Tech. Rep. 98003, 1998.

[38] A. Liao, "Modifying BFGS method," Advance Comput. Res. Inst., Cornell Theory Center, Cornell Univ., Tech. Rep., Jan. 31, 1994.

[39] M. T. Hagan and M. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.

[40] N. Ampazis and S. J. Perantonis, "Two highly efficient second-order algorithms for training feedforward networks," *IEEE Trans. Neural Netw.*, vol. 13, no. 5, pp. 1064–1074, Sep. 2002.

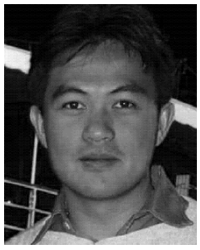
[41] F. Bärmann and F. Biegler-König, "A learning algorithm for multilayered neural networks based on linear least squares problems," *Neural Netw.*, vol. 6, pp. 127–131, 1993.

[42] B. Verma, "Fast training of multilayer perceptions," *IEEE Trans. Neural Netw.*, vol. 8, no. 6, pp. 1314–1321, Nov. 1997.

[43] M. D. Martino, S. Fanelli, and M. Protasi, "Exploring and comparing the best "direct methods" for the efficient training of MLP-networks," *IEEE Trans. Neural Netw.*, vol. 7, no. 6, pp. 1497–1502, Nov. 1996.

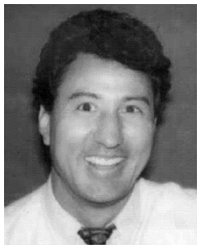
[44] S. L. Phung, A. Bouzerdoum, and D. Chai, "Skin segmentation using color pixel classification: Analysis and comparison," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 1, pp. 148–154, Jan. 2005.

[45] C. Charalambous, "Conjugate gradient algorithm for efficient training of artificial neural networks," in *Proc. Inst. Elect. Eng. G*, vol. 139, 1992, pp. 301–310.



Fok Hing Chi Tivive received the B.Eng. (Hons.) degree in communication systems from Edith Cowan University, Perth, Australia. He is currently working toward the Ph.D. degree at the University of Wollongong.

His research interests include image processing, pattern recognition, machine learning, and neural networks.



Abdesselam Bouzerdoum (M'89–SM'03) received the M.S. and Ph.D. degrees in electrical engineering from the University of Washington, Seattle.

He joined the University of Adelaide, Australia, in 1991, and in 1998, he was appointed Associate Professor at Edith Cowan University, Perth, Australia. Since 2004, he has been with the University of Wollongong, Australia, where he is currently a Professor of Computer Engineering and Head of School of Electrical, Computer, and Telecommunications Engineering. In 2004, he was a Visiting Professor at Institut Galile, University of Paris-13, Paris, France. He has published over 200 technical articles and graduated 13 Ph.D. and six Masters students. His research interests include signal/image processing, pattern recognition, machine learning/neural networks, and VLSI implementation of smart vision microsensors.

Dr. Bouzerdoum has received several fellowships and distinguished awards; among them are the Vice Chancellor's Distinguished Researcher Award in 1998 and 1999, and the Awards for Excellence in Research Leadership and Excellence in Postgraduate Supervision. In 2001, he was the recipient of a Distinguished Researcher (Chercheur de Haut Niveau) Fellowship from the French Ministry of Research. He served as Chair of the IEEE WA Section Signal Processing Chapter in 2004, and was Chair of the IEEE SA Section NN RIG from 1995 to 1997. Currently, he is serving as an Associate Editor for the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS PART C. He is a member of the International Neural Network Society (INNS), and a Member of the International Association of Science and Technology for Development (IASTED) Technical Committee on Neural Networks.