Optimal In-Place Self-Organization for Cortical Development: Limited Cells, Sparse Coding and Cortical Topography

Juyang Weng and Matthew D. Luciw
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824 USA

***Abstract*— Cortical self-organization during open-ended development is a core issue for perceptual development. Traditionally, unsupervised learning and supervised learning are two different types of learning conducted by different networks. However, there is no evidence that the biological nervous system treats them in a disintegrated way. The computational model presented here integrates both types of learning using a new biologically inspired network whose learning is in-place. By in-place learning, we mean that each neuron in the network learns on its own while interacting with other neurons. There is no need for a separate learning network. We present in this paper the Multi-layer In-place Learning Network (MILN) for regression and classification. This work concentrates on its two-layer version for global pattern detection (without incorporating an attention selection mechanism). It reports properties about limited cells, sparse coding and cortical topography. The network enables both unsupervised and supervised learning to occur concurrently. Within each layer, the adaptation of each neuron is nearly-optimal in the sense of the least possible estimation error given the observations. Experimental results are presented to show the effects of the properties investigated.**

***Index Terms*— Biological cortical learning, statistical efficiency, minimum error, self-organization, incremental learning**

I. Introduction

What are the possible mechanisms that lead to the emergence of the orientation cells in V1? Since V1 takes input from the retina, LGN, and other cortical areas, the issue points to the developmental mechanisms for the formation and adaptation of the multi-layer pathways of visual processing.

Well known unsupervised learning algorithms include Self-Organizing Map (SOM), vector quantization, PCA, Independent Component Analysis (ICA), Isomap, and Non-negative Matrix Factorization (NMF). Only a few of these algorithms have been expressed by in-place versions (e.g., SOM and PCA [11]).

Supervised learning networks include feed-forward networks with back-propagation learning, radial-basis functions with iterative model fitting (based on gradient or similar principles), Cascade-Correlation Learning Architecture [2], support vector machines (SVM), and Hierarchical Discriminant Regression (HDR) [3].

However, it is not convincing that biological networks use two different types of networks for unsupervised and supervised learning, which occur in an intertwined way in the process of development. When a child learns to draw, his parent can hold his hand during some periods to guide his hand movement (i.e., supervised) but leave him practicing on his own during other periods (i.e., unsupervised). Does the brain switch between two totally different networks, one for supervised moments and the other for unsupervised moments? The answer to this type of question is not clear at the current stage of knowledge. However, there is evidence that the cortex has wide-spread projections both bottom-up and top-down [8] (pages 99-103). For example, cells in layer 6 in V1 project back to the lateral geniculate nucleus [5] (page 533). Can projections from later cortical areas be used as supervision signals?

Currently there is a lack of biologically inspired networks that integrate these two different learning modes using a single learning network. The network model proposed here enables unsupervised and supervised learning to take place at the same time throughout the network.

One of the major advantages of supervised learning is the development of certain invariant representations. Some networks have built-in (programmed-in) invariance, either spatial, temporal or some other signal properties. Other networks do not have built-in invariance. The required global invariance then must be learned object-by-object. However, they cannot share invariance of subparts (or locally invariant features) for different objects. Consequently, the number of samples needed to reach the desired global invariance in object recognition is very large.

This paper proposes a new, general-purpose, multi-layer network, which learns invariance from experience. The network is biologically inspired. The network has multiple layers; later layers take the response from early layers as their input. This work concentrates on two layers. The network enables supervision from two types of projections: (a) supervision from the succeeding layer; (b) supervision from other cortical regions (e.g., as attention selection signals). The network is self-organized with unsupervised signals (input data) from bottom-up and supervised signals (motor signals, attention selection, etc.) from top-down.

From a mathematical point of view, in each layer of the network, unsupervised learning enables nodes (neurons) to

generate a self-organized map that approximates the statistical distribution of the bottom-up signals (input vector space), while supervised learning adjusts the node density in such a map so that those areas in the input space that are not related (or weakly related) to the output from this layer receive no (or fewer) nodes. Therefore, more nodes in each layer will respond to output-relevant input components. This property leads to increasing invariance from one layer to the next in a multi-layer network. Finally, global invariance emerges at the last (motor) layer.

Furthermore, we require in-place learning. By in-place learning, we mean that the signal processing network itself deals with its own adaptation through its own internal physiological mechanisms and interactions with other connected networks and, thus, there is no need to have an extra network that accomplishes the leaning (adaptation). It is apparent that a design of an in-place learning, biologically inspired network that integrates both unsupervised and supervised learning is not trivial.

In what follows, we first present the network structure in Section II. Then, in Section III, we explain the in-place learning mechanism within each layer. Experimental examples that demonstrate the effects of the discussed principles are presented in Section IV. Section V provides some concluding remarks.

II. The Multi-Layer In-place Learning Network

This section presents the architecture of the new Multi-layer In-place Learning Networks (MILN), whose architecture is shown in Fig. 1. For biological plausibility, assume that the signals through the lines are *non-negative* signals that indicate the firing rate. Two types of synaptic connections are possible, excitatory and inhibitory.

This is a recurrent network. For each neuron i, at layer l, there are three types of weights:

1) bottom-up (excitatory) weight vector $\mathbf{w}_b$ that links input lines from the previous layer $l-1$ to this neuron;
2) lateral (inhibitory) weight $\mathbf{w}_h$ that links other neurons in the same layer to this neuron.
3) top-down (excitatory or inhibitory) weight $\mathbf{w}_t$. It consists of two parts: (a) the part that links the output from the neurons in the next layer $l+1$ to this neuron. (b) The part that links the output of other layer processing areas (e.g., other sensing modality) or layers (e.g., the motor layer) to this neuron i. For notational simplicity, we only consider excitatory top-down weight, which selects neurons selected to increase their potential values. Inhibitory top-down connection can be used if the primary purpose is inhibition (e.g., inhibition of neurons that have not been selected by attention selection signals).

Assume that this network computes in discrete times, $t = 0, 1, 2, ...$, as a series of open-ended developmental experience after the birth at time $t = 0$. This network incorporates unsupervised learning and supervised learning. For unsupervised learning, the network produces an output vector at the output layer based on this recurrent computation. For supervised learning, the desired output at the output layer at time t is set (imposed) by the external teacher at time t.

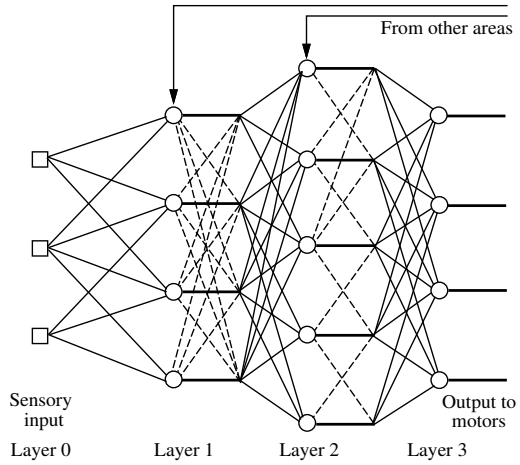


Fig. 1. The architecture of the Multi-layer In-place Learning Networks. A circle indicates a cell (neuron). The thick segment from each cell indicates its axon. The connection between a solid signal line and a cell indicates an excitatory connection. The connection between a dashed signal line and a cell indicates an inhibitory connection. Projection from other areas indicates excitatory or inhibitory supervision signals (e.g., excitatory attention selection).

III. In-Place Learning

To better understand the nature of learning algorithms, we define five types of learning algorithms:

Type-1 batch: A batch learning algorithm L_1 computes g and $\mathbf{w}$ using a batch of vector inputs $B = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_b\}$, where b is the batch size. **Type-2 block-incremental**: A type-2 learning algorithm, L_2, breaks a series of input vectors into blocks of certain size b ($b > 1$) and computes updates incrementally between blocks. Within each block, the processing by L_2 is in a batch fashion. **Type-3 incremental**: Type-3 is the extreme case of Type-2 in the sense that block size $b = 1$. **Type-4 covariance-free incremental**: A Type-4 learning algorithm L_4 is a Type-3 algorithm, but, it is not allowed to compute the 2nd or higher order statistics of the input $\mathbf{x}$. The CCI PCA algorithm [11] is Type-4 algorithm. **Type-5 in-place neuron learning**: A Type-5 learning algorithm L_5 is a Type-4 algorithm, but further, the learner L_5 must be implemented by the signal processing neuron. It is desirable that a developmental system uses an in-place developmental program due to its simplicity and biological plausibility. Further, biological in-place learning mechanisms can facilitate our understanding of biological systems since there is no evidence that each biological network has a separate network to handle its learning.

The five types of algorithms have progressively more restrictive conditions, with batch (Type-1) being the most general and in-place (Type-5) being the most restrictive.

A. Three types of projections

Consider a more detailed computational model of a layer within a multi-layer network. Suppose the input to the cortical layer is $\mathbf{y} \in \mathcal{Y}$, the output from early neuronal processing. However, for a recurrent network, $\mathbf{y}$ is not the only input. All the input to a neuron can be divided into three parts: bottom-up input from the previous layer $\mathbf{y}$ which is weighted by the neuron's bottom-up weight vector $\mathbf{w_b}$, lateral inhibition $\mathbf{h}$ from other neurons of the same layer corresponding which is weighted by the neuron's lateral weight vector $\mathbf{w_h}$, and the top-down input vector $\mathbf{a}$ which is weighted by the neuron's top-down weight vector $\mathbf{w_t}$. Therefore, the response z from this neuron can be written as

$$z = g(\mathbf{w}_b \cdot \mathbf{y} - \mathbf{w}_h \cdot \mathbf{h} + \mathbf{w}_t \cdot \mathbf{a}). \quad (1)$$

where g is its nonlinear sigmoidal function, taking into account under-saturation (noise suppression), transition, and over-saturation, and $\cdot$ denotes the dot production.

For digital computer, we simulate this analogue network through discrete times $t = 0, 1, 2, ...$ If the discrete sampling rate is much faster than the change of inputs, such a discrete simulation is expected to be a good approximation of the network behavior.

B. Lateral projection

Lateral inhibition is a mechanism of competition among neurons in the same layer. The output of A is used to inhibit the output of neuron B which shares a part of the receptive field, totally or partially, with A.

The net effect of lateral inhibition is that fewer winners can fire. We use a computationally more effective scheme to simulate lateral inhibition without resorting to iterations: Sort all the responses. Keep top-k responding neurons to have non-zero response. All other neurons have zero response (i.e., does not go beyond the under-saturation point).

C. Bottom-up projections

A neuron is updated using an input vector only when the (absolute) response of the neuron to the input is high. This is called Hebbian's rule.

The rule of Hebbian learning is to update weights when output is strong. And the rule of lateral inhibition is to suppress neighbors when the neuron output is high.

D. Lobe components

Atick and coworkers [1] proposed that early sensory processing decorrelates inputs. Weng et al. [11] proposed an in-place algorithm that develops a network that whitens the input. Therefore, we can assume that prior processing has been done so that its output vector $\mathbf{y}$ is roughly white. By white, we mean its components have unit variance and are pairwise uncorrelated. The sample space of a k-dimensional white input random vector $\mathbf{y}$ can be illustrated by a k-dimensional hypersphere.

A concentration of the probability density of the input space is called a lobe, which may have its own finer structure (e.g., sublobes). The shape of a lobe can be of any type, depending on the distribution. For non-negative input components, the lobe components lie in the section of the hypersphere where every component is non-negative (corresponding to the first octant in 3-D).

Given a limited cortical resource, c cells fully connected to input $\mathbf{y}$, the developing cells divide the sample space $\mathcal{Y}$ into c mutually nonoverlapping regions, called *lobe regions*:

$$\mathcal{Y} = R_1 \cup R_2 \cup ... \cup R_c, \quad (2)$$

(where $\cup$ denotes the union of two spaces). Each region R_i is represented by a single unit feature vector $\mathbf{v}_i$, called the *lobe component*. Given an input $\mathbf{y}$, many cells, not only $\mathbf{v}_i$, will respond. The response pattern forms a new population representation of $\mathbf{y}$.

Suppose that a unit vector (neuron) $\mathbf{v}_i$ represents a lobe region R_i. If $\mathbf{y}$ belongs to R_i, $\mathbf{y}$ can be approximated by $\mathbf{v}_i$ as the projection onto $\mathbf{v}_i$: $\mathbf{y} \approx \hat{\mathbf{y}} = (\mathbf{y} \cdot \mathbf{v}_i)\mathbf{v}_i$. Suppose the neuron $\mathbf{v}_i$ minimizes the mean square error $E\|\mathbf{y} - \hat{\mathbf{y}}\|^2$ of this representation when $\mathbf{y}$ belongs to R_i.

According to the theory of Principal Component Analysis (PCA) (e.g., see [4]), we know that the best solution of column vector $\mathbf{v}_i$ is the principal component of the conditional covariance matrix $\Sigma_{y,i}$, conditioned on $\mathbf{y}$ belonging to R_i. That is $\mathbf{v}_i$ satisfies $\lambda_{i,1}\mathbf{v}_i = \Sigma_{y,i}\mathbf{v}_i$.

Replacing $\Sigma_{y,i}$ by the estimated sample covariance matrix of column vector y, we have

$$\lambda_{i,1}\mathbf{v}_i \approx \frac{1}{n}\sum_{t=1}^{n} \mathbf{y}(t)\mathbf{y}(t)^\top \mathbf{v}_i = \frac{1}{n}\sum_{t=1}^{n} (\mathbf{y}(t) \cdot \mathbf{v}_i)\mathbf{y}(t). \quad (3)$$

We can see that the best lobe component vector $\mathbf{v}_i$, scaled by "energy estimate" eigenvalue $\lambda_{i,1}$, can be estimated by the *average* of the input vector $\mathbf{y}(t)$ weighted by the linearized (without g) response $\mathbf{y}(t) \cdot \mathbf{v}_i$ whenever $\mathbf{y}(t)$ belongs to R_i. This average expression is crucial for the concept of optimal statistical efficiency discussed below.

E. Optimality

Suppose that there are two estimators Γ_1 and Γ_2, for a vector parameter (i.e., synapses or a feature vector) $\theta = (\theta_1, ..., \theta_k)$, which are based on the same set of observations $S = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$. If the expected square error of Γ_1 is smaller than that of Γ_2 (i.e., $E\|\Gamma_1 - \theta\|^2 < E\|\Gamma_2 - \theta\|^2$), the estimator Γ_1 is more statistically efficient than Γ_2. Given the same observations, among all possible estimators, the optimally efficient estimator has the smallest possible error. The challenge is how to convert a nonlinear search problem

into an optimal estimation problem using the concept of statistical efficiency.

For in-place development, each neuron does not have extra space to store all the training samples $\mathbf{y}(t)$. Instead, it uses its physiological mechanisms to update synapses incrementally. If the i-th neuron $\mathbf{v}_i(t-1)$ at time $t-1$ has already been computed using previous $t-1$ inputs $\mathbf{y}(1), \mathbf{y}(2), ..., \mathbf{y}(t-1)$, the neuron can be updated into $\mathbf{v}_i(t)$ using the current sample defined from $\mathbf{y}(t)$ as:

$$\mathbf{x}_t = \frac{\mathbf{y}(t) \cdot \mathbf{v}_i(t-1)}{\|\mathbf{v}_i(t-1)\|} \mathbf{y}(t). \tag{4}$$

Then Eq. (3) states that the lobe component vector is estimated by the average:

$$\lambda_{i,1} \mathbf{v}_i \approx \frac{1}{n} \sum_{t=1}^{n} \mathbf{x}_t. \tag{5}$$

Statistical estimation theory reveals that for many distributions (e.g., Gaussian and exponential distributions), the sample mean is the most efficient estimator of the population mean (see, e.g., Theorem 4.1, p. 429-430 of Lehmann [7]). In other words, the estimator in Eq. (5) has nearly the minimum error given the observations.

F. Lobe components for nonstationary processes

The sensory environment of a developing brain is not stationary. That is the distribution of the environment changes over time. Therefore, the sensory input process is a nonstationary process too. We use the amnesic mean technique below which gradually "forgets" old "observations" (which use bad $\mathbf{x}_t$ when t is small) while keeping the estimator quasi-optimally efficient.

The mean in Eq. (5) is a batch method. For incremental estimation, we use what is called an amnesic mean [11].

$$\bar{x}^{(t)} = \frac{t-1-\mu(t)}{t} \bar{x}^{(t-1)} + \frac{1+\mu(t)}{t} x_t \tag{6}$$

where $\mu(t)$ is the amnesic function depending on t. If $\mu \equiv 0$, the above gives the straight incremental mean. We adopt a profile of $\mu(t)$:

$$\mu(t) = \begin{cases} 0 & \text{if } t \le t_1, \\ c(t-t_1)/(t_2-t_1) & \text{if } t_1 < t \le t_2, \\ c + (t-t_2)/r & \text{if } t_2 < t, \end{cases} \tag{7}$$

in which, e.g., $c = 2, r = 10000$. As can be seen above, $\mu(t)$ has three intervals. When t is small, straight incremental average is computed.

G. Single-layer updating algorithm

We model the development (adaptation) of an area of cortical cells (e.g., a cortical column) connected by a common input column vector $\mathbf{y}$ by the following Candid Covariance-free Incremental Lobe Component Analysis (CCI LCA) (Type-5) algorithm, which incrementally updates c such cells (neurons) represented by the column vectors $\mathbf{v}_1^{(t)}, \mathbf{v}_2^{(t)}, ..., \mathbf{v}_c^{(t)}$ from input samples $\mathbf{y}(1), \mathbf{y}(2), ...$ of dimension k without computing the $k \times k$ covariance matrix of $\mathbf{y}$. The length of the estimated $\mathbf{v}_i$, its eigenvalue, is the variance of projections of the vectors $\mathbf{y}(t)$ onto $\mathbf{v}_i$. The output of the layer is the response vector $\mathbf{z} = (z_1, z_2, ..., z_c)$. The quasi-optimally efficient, in-place learning, single layer CCI LCA algorithm $\mathbf{z} = \text{LCA}(\mathbf{y})$ is as follows:

1) Sequentially initialize c cells using first c observations: $\mathbf{v}_t^{(c)} = \mathbf{y}(t)$ and set cell-update age $n(t) = 1$, for $t = 1, 2, ..., c$.
2) For $t = c+1, c+2, ...$, do
   a) If the output is not given, compute output (response) for all neurons: For all i with $1 \le i \le c$, compute response:
   $$z_i = g_i \left(\frac{\mathbf{y}(t) \cdot \mathbf{v}_i^{(t-1)}}{\|\mathbf{v}_i^{(t-1)}\|} \right), \tag{8}$$
   where g_i is a neuron-specific sigmoidal function. A simple version of g_i is a linear function with under- and over-saturation points at a distance of a few standard deviations away from the mean of the input.
   b) Simulating lateral inhibition, decide the winner: $j = \arg\max_{1 \le i \le c} \{z_i\}$, using z_i as the belongingness of $y(t)$ to R_i.
   c) Update only the winner neuron v_j using its temporally scheduled plasticity:
   $$\mathbf{v}_j^{(t)} = w_1 \mathbf{v}_j^{(t-1)} + w_2 l_j \mathbf{y}(t),$$
   where the scheduled plasticity is determined by its two age-dependent weights:
   $$w_1 = \frac{n(j) - 1 - \mu(n(j))}{n(j)}, w_2 = \frac{1 + \mu(n(j))}{n(j)},$$
   with $w_1 + w_2 \equiv 1$. Update the number of hits (cell age) $n(j)$ only for the winner: $n(j) \leftarrow n(j) + 1$.
   d) All other neurons keep their ages and weight unchanged: For all $1 \le i \le c$, $i \ne j$, $\mathbf{v}_i^{(t)} = \mathbf{v}_i^{(t-1)}$.

The neuron winning mechanism corresponds to the well known mechanism called lateral inhibition (see, e.g., Kandel et al. [5] p. 4623). The winner updating rule is a computer simulation of the Hebbian rule (see, e.g., Kandel et al. [5] p.1262). Assuming the plasticity scheduling by w_1 and w_2 are realized by the genetic and physiologic mechanisms of the cell, this algorithm is in-place. Alternatively, we use "soft winners" where multiple top (e.g., top-k) winners update.

H. Multiple-layer in-place learning

The following is the multi-layer in-place learning algorithm $\mathbf{z}(t) = \text{MILN}(\mathbf{x}(t))$. Suppose that the network has l layers.

MILN Learning: Initialize the time $t = 0$. Do the following forever (development), until power is off.

1) Grab the current input frame $\mathbf{x}(t)$. Let $\mathbf{y}_0 = \mathbf{x}(t)$.
2) If the current desired output frame is given, set the output at layer l, $\mathbf{y}_l \leftarrow \mathbf{z}(t)$, as given.
3) For $j = 1, 2, ..., l$, run the LCA algorithm on layer j, $\mathbf{y}_j = \text{LCA}(\mathbf{y}_{j-1})$, where the layer j is also updated.
4) Produce output $\mathbf{z}(t) = \mathbf{y}_l$; $t \leftarrow t + 1$.

IV. Experimental Examples

The network is intended to serve as a new engine for cortical development using natural images as input. However, to show the effect of the discussed properties more clearly, we use the MNIST database of handwritten digits available at http://yann.lecun.com/exdb/mnist/ so that the image of each weight vector intuitively shows samples that contributed to it.

An attention selection mechanism, such as that of Zhang & Weng [12] will be incorporated into this network in our future study. Without an attention mechanism, we should not expect that the global match performed by the network will outperform other methods that perform local analysis (e.g., convolution using small templates [6]) given the same set of training samples. This is the first framework that we know of for in-place incremental cortical development. It is important to understand the properties of this network before applying attention selection mechanisms.

The MNIST data set consists of 70,000 images of the handwritten digits from 0 to 9, with 60,000 samples for training and 10,000 for testing. The 250 writers represented in the training set are disjoint from the 250 writers represented in the testing set (i.e., writer-disjoint). Each image is composed of $28 \times 28 = 784$ pixel intensity values, with a grey and white digit over a black background. All images have already been translation-normalized, so that each digit resides at the center of the image.

Networks can be constructed with any number of neurons on any number of layers. In the following experiments, networks with only two-layers are used. For those with more layers, learning the weights of the hidden layers is different than in the traditional multi-layer feed-forward networks with back-propagation learning and are discussed in [10].

A. Limited cells

In the first experiment, we test global prototyping capabilities of lobe components when the number of cells is limited. All the networks have n layer-1 neurons and 10 layer-2 neurons (for 10 classes of digits). During the supervised training session, we set the layer-2 output vector in the following way: When a sample of class i is given, $i = 0, 1, ..., 9$, the layer-2 output is such that the i-th neuron gives 1 and all other neurons give zero outputs. Therefore, lobe components in layer-1 serve as "prototypes" of a class. We used top-1 response in that only a single winner at each layer can fire. Thus, the layer-2 neurons determine which class the firing layer-1 neuron belongs to. We constructed networks with different numbers of n, to study the effect of varying number of "prototypes" in layer-1.

To compare with the effect of LCA updating, we also tested a simpler network call "Pure Initialization," where the weights of n layer-1 neurons were initialized by the sequentially arriving samples as in the LCA algorithm, but they are not updated any further when later samples arrive. The training set was pre-arranged so that the number of neurons representing each class were nearly equal. In the second network type "updated lobe components," the layer-1 lobe-components were further updated as specified by the LCA algorithm. When $n = 60,000$, both approaches will give the same result, since the number of layer-1 neurons is the same as the number of training samples. Both layers are trained at once using a single cycle through the samples.

A test sample can then be classified using the neuron index with the highest layer-2 response. Performance is based on the error rate for all 10,000 test samples. Results are summarized in Fig. 2. It can be proved that both types of networks perform nearest-neighbor match in the inner-product space, using the lobe components in layer-1 as prototypes. The plot shows that the larger the number of prototypes, the smaller the error rate, in general. When $n = 60,000$, the number of training samples is the same as the number of prototypes, which is excessive for most applications. When n is reduced, the error rate also increases. However, "updated lobe components" give smaller errors, which shows that the initialization followed by updating is more effective than the pure initialization alone when the number of prototypes is smaller than the number of training samples. When n is reduced to $1,800$, about 33 samples per prototype, the error rate only increased slightly from $n = 60,000$.

B. Sparse coding

In this experiment, we study whether it will help to allow more neurons to fire. We used the same network structure as the first experiment, but all the lobe components in layer-1 are updated. In the first network type "top one nonzero response," only the top-one winner in layer-1 is allowed to fire. In the second type called "top three nonzero responses," the top-three responses from layer-1 were multiplied by a factor of 1, 0.66 and 0.33, respectively, and all other layer-1 neurons give zero response. The error rates for different numbers of layer-1 neurons are shown in Fig. 3. We know that top-3 responses give more information about the position of the input in relation with the top-three winning neurons. However, Fig. 3 showed that multiple responses did not help when the density of the prototypes is low. This is reasonable because bringing in far-away prototypes in decision making may add noise. This situation changed when the number of

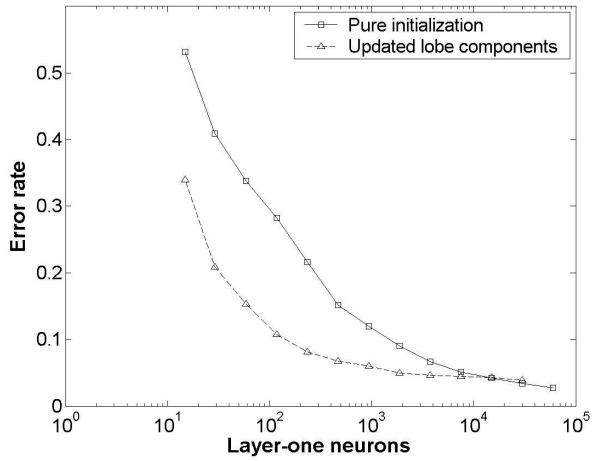


Fig. 2. The effects of the limited number of layer-1 cells and the update of lobe components. In "Pure initialization," every weight vector in layer-1 is purely initialized by a single training sample and is not updated.

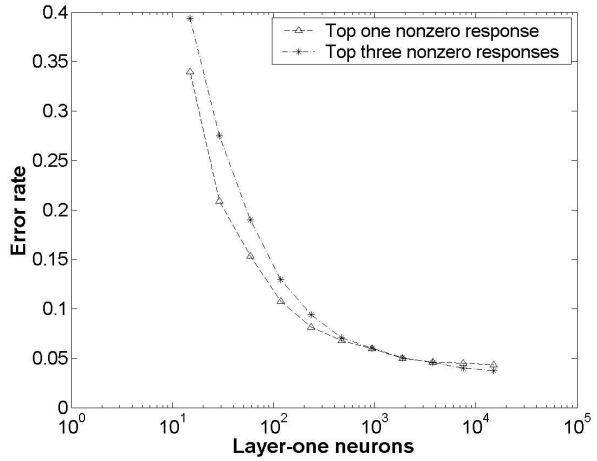


Fig. 3. The effect of multiple responses versus the neuronal density.

prototypes is sufficiently large so that the positions of nearby prototypes better predict the shape of the class boundaries indicated by the crossing point in Fig. 3.

C. Topography and invariance

The next experiments focus on topographical cortical self-organization and its implication to within-class invariance. As observed in V1 and other cortical areas, nearby neurons have similar representations. This means nearby neurons represent nearby inputs or concepts. We place all layer-1 neurons in a two-dimensional, square grid, simulating a "cortical sheet." The winner neuron that updates for each input will also cause its nearby neurons within a distance of 2 from the winner to update as well, with their gain w_2 in the algorithm weighted by their distance to the winner, but $w_1 + w_2 = 1$ still holds.

Fig. 4. The topographic map of layer-1 neurons for the hand-written digit 8.

Fig. 4 shows an example with a 10×10 grid and all of the training samples were from the class "8." We can see that within-class variation of the digit 8 is represented by this cortical area, sampled by the discrete number of prototypes. The invariance for such within-class variation is achieved by the positive weights from all of these neurons to the 8-th layer-2 neuron.

The above case does not include between-class competition for resources (neurons). To show the self-organization of resource for all 10 classes, Fig. 5(a) shows the layer-1 neurons in a 40×40 grid trained with all training samples. There are areas which can be seen to correspond to a particular class, since there is no guarantee that the area of a single class is connected in the topographic map, which is also the case in biological networks (see, e.g., [9]). It depends on the size of neighborhood update, the inputs and degree of input variations.

Fig. 5(b) shows the learned weights of the layer-2 neuron corresponding to the digit-1 class. The darker the intensity, the larger the weight. As shown, invariance is achieved by the positive weights from the corresponding digit "1" neurons to the corresponding output neuron. Therefore, the within-class invariance shown at the output layer can be learned from multiple regions in the previous layer.

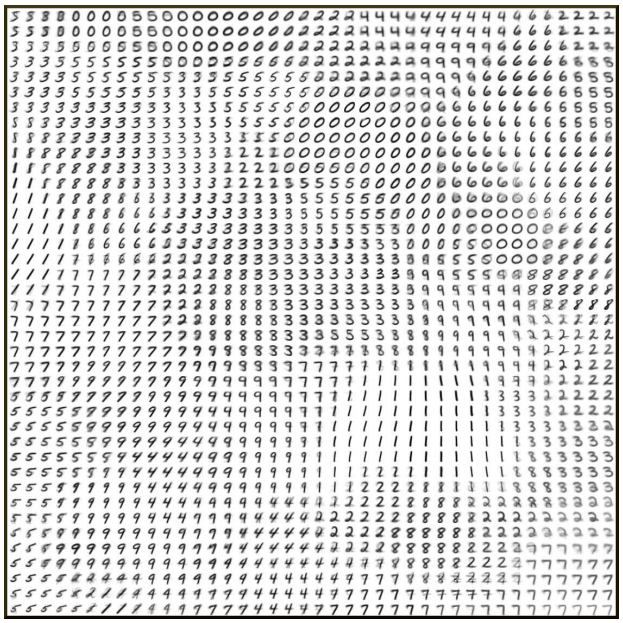
V. Conclusions

This is mainly a theoretical paper about a multi-layer network that integrates bottom-up unsupervised learning and top-down supervised learning. This is the first multi-layer in-place learning network for general-purpose regression and classification with a near-optimal within-layer statistical efficiency. In other words, within each layer all neurons incrementally maintain a near-optimal representation while receiving sensory inputs from developmental experience. The experimental networks showed that those lobe components can learn the complex shape of the class manifolds in the layer's input (inner product) space.

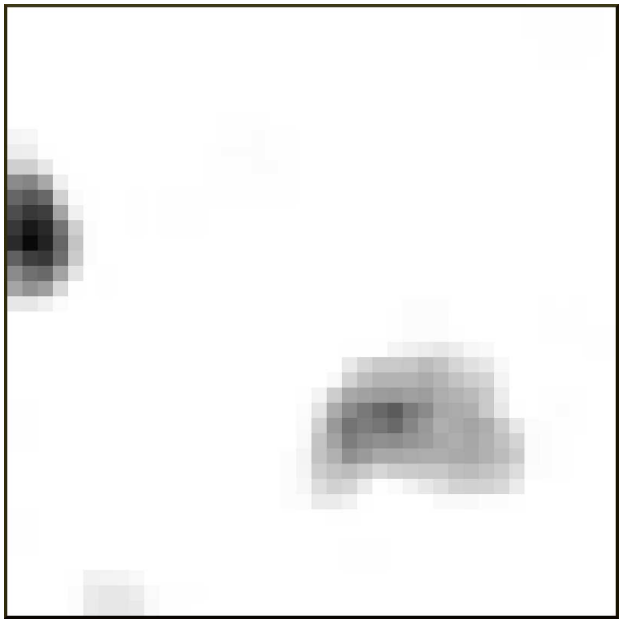
References

[1] J. J. Atick and A. N. Redlich. Towards a theory of early visual processing. *Neural Computation*, 2:308–320, 1990.

[2] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Feb. 1990.
[3] W. S. Hwang and J. Weng. Hierarchical discriminant regression. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(11):1277–1293, 2000.
[4] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.
[5] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, editors. *Principles of Neural Science*. McGraw-Hill, New York, 4th edition, 2000.
[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of IEEE*, 86(11):2278–2324, 1998.
[7] E. L. Lehmann. *Theory of Point Estimation*. John Wiley and Sons, Inc., New York, 1983.
[8] M. I. Posner and M. E. Raichle. *Images of Mind*. Scientific American Library, New York, 1994.
[9] X. Wang, M. M. Merzenich, K. Sameshima, and W. M. Jenkins. Remodeling of hand representation in adult cortex determined by timing of tactile stimulation. *Nature*, 378(2):13–14, 1995.
[10] J. Weng, H. Lu, T. Luwang, and X. Xue. In-place learning for positional and scale invariance. In *Proc. IEEE World Congress on Computational Intelligence*, Vancouver, BC, Canada, July 16-21 2006.
[11] J. Weng, Y. Zhang, and W. Hwang. Candid covariance-free incremental principal component analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 25(8):1034–1040, 2003.
[12] N. Zhang and J. Weng. A developing sensory mapping for robots. In *Proc. IEEE 2nd International Conf. on Development and Learning (ICDL 2002)*, pages 13–20, MIT, Cambridge, Massachusetts, June 12-15 2002.



(a)



(b)

Fig. 5. (a) The topographic map of layer-1 neurons. (b) Learned weights of the layer-2 neuron for the digit "1" class.