Available online at www.sciencedirect.com

ScienceDirect

Neural Networks 21 (2008) 150–159

Neural Networks

www.elsevier.com/locate/neunet

2008 Special Issue

Multilayer in-place learning networks for modeling functional layers in the laminar cortex[☆]

Juyang Weng[a,b,1], Tianyu Luwang[a], Hong Lu[a,*], Xiangyang Xue[a]

[a] *Department of Computer Science and Engineering, Fudan University, Shanghai, China*
[b] *Michigan State University, East Lansing, MI 48824, USA*

Received 8 August 2007; received in revised form 18 December 2007; accepted 29 December 2007

Abstract

Currently, there is a lack of general-purpose in-place learning networks that model feature layers in the cortex. By "general-purpose" we mean a general yet adaptive high-dimensional function approximator. In-place learning is a biological concept rooted in the genomic equivalence principle, meaning that each neuron is fully responsible for its own learning in its environment and there is no need for an external learner. Presented in this paper is the Multilayer In-place Learning Network (MILN) for this ambitious goal. Computationally, in-place learning provides unusually efficient learning algorithms whose simplicity, low computational complexity, and generality are set apart from typical conventional learning algorithms. Based on the neuroscience literature, we model the layer 4 and layer 2/3 as the feature layers in the 6-layer laminar cortex, with layer 4 using unsupervised learning and layer 2/3 using supervised learning. As a necessary requirement for autonomous mental development, MILN generates invariant neurons in different layers, with increasing invariance from earlier to later layers and the total invariance in the last motor layer. Such self-generated invariant representation is enabled mainly by descending (top-down) connections. The self-generated invariant representation is used as intermediate representations for learning later tasks in open-ended development.
© 2008 Elsevier Ltd. All rights reserved.

Keywords: Laminar cortex; Cortex structure; Feature extraction; Regression; Unsupervised learning; Supervised learning; Representation; Abstraction; Multi-task learning

1. Introduction

The *genomic equivalence* principle (Purves, Sadava, Orians, & Heller, 2004) states that the set of genes in the nuclei of every cell is functionally complete — sufficient to guide the development from a single cell into an entire adult life. There are no genes that are devoted to more than one cell as a whole. Therefore, development guided by the genome is *cell-centered*. Carrying a complete set of genes and acting as an autonomous machine, each cell must handle its own learning while interacting with its external environment (e.g., other cells). Each neuron (a single cell) develops and learns in place. It does not need any dedicated learner outside the neuron. For example, it does not need an extra-cellular learner to compute the covariance matrix (or any other moment matrices or partial derivatives) of its input lines and store extra-celullarly.

Popular artificial networks include Feed-Forward Networks (FFN) with back-propagation learning, Radial Basis Functions (RBF), Self-Organization Maps (SOM), Cascade-Correlation Learning Architecture (CCLA) (Fahlman & Lebiere, 1990), Support Vector Machines (SVM), and Incremental Hierarchical Discriminant Regression (IHDR) (Weng & Hwang, 2007). However, adaptation and long-term memory are two conflicting criteria. Realizing both concurrently is a great challenge. Gradient-based methods used by FFN, RBF and SOM update the network along the greedy gradient direction computed by the last input data, without properly taking into account the observations along the past nonlinear search trajectory. CCLA addresses the loss of memory problem by freezing all old internal nodes in FFN and adding new nodes for new observations, which is a simple yet crude way of growing a

[☆] An abbreviated version of some portions of this article appeared in Weng, Luwang, Lu, and Xue (2007) as part of the IJCNN 2007 Conference Proceedings, published under IEE copyright.

[*] Corresponding author. Tel.: +86 2165643922; fax: +86 2165643922.
E-mail addresses: weng@cse.msu.edu (J. Weng), honglu@fudan.edu.cn (H. Lu).

[1] Tel.: +1 517 353 4388; fax: +1 517 353 4388.

0893-6080/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2007.12.048

Table 1
A comparison of major network models

Network	HD	Inc	NLE	LTM	In-place	Sup	Soft-Inv	Complete
FFN	N	Y	N	N	N	Y	N	N
RBF	N	Y	N	N	N	Y	N	N
SOM	Y	Y	Y	Y	Y	N	N	Y
CCLA	N	Y	N	Y	N	Y	N	N
SVM	N	N	Y	Y	N	Y	N	N
IHDR	Y	Y	Y	Y	N	Y	N	N
MILN	Y	Y	Y	Y	Y	Y	Y	Y

HD: High-dimensional; Inc: Incremental; NLE: No local extrema; LTM: Long-term memory; Sup: Supervision; Soft-Inv: Soft-invariance.

network, inappropriate for online autonomous development for an open length of time period. IHDR overcomes both problems by allowing dynamic spawning nodes from a growing tree, while shallow nodes and unmatched leaf nodes serving as the long term memory. However, IHDR is not an in-place learner (e.g., computing covariance matrix for each neuron).

The multi-layer in-place learning network proposed here is an in-place learning network whose architecture is biologically inspired. The weight vector of each neuron is not computed based on gradient. Instead, it is the amnesic average (called the lobe component) with properly scheduled, experience-dependent plasticity. The lobe component represents the most efficient statistic (i.e., minimum error given observations) for a stationary process and is almost the most efficient statistic for a nonstationary process. In other words, all the observations along the nonlinear search path of every neuron are taken into account in the best way for each update. Detailed properties of the lobe component analysis and the performance comparison with ICA are available in Weng and Zhang (2006).

It has been well recognized that all cortical areas have 6 layers, from L1 at the most superficial layer to L6 at the deepest layer (see, e.g., Kandel, Schwartz, and Jessell (1991), pages 327–331). Each MILN models a sensorimotor pathway through a series of cortical areas, or a subset thereof.

The importance of MILN is indicated by its conjunctive consideration of 8 challenging properties that are motivated by biological development and engineering applications: (1) high-dimensional input; (2) incremental learning; (3) without a significant local minima problem for regression; (4) without a significant loss of memory problem; (5) in-place learning; (6) enable supervised and unsupervised learning in any order suited for development; (7) local-to-global invariance from early to later processing; (8) a rich completeness of response at the scale corresponding to a layer and the given layer resource (e.g., the number of cells). FFN suffers problems in (1), (3), (4), (5), (7), and (8). RBF has major problems with (3), (4), (7), and (8). SOM does not meet (7) and (8). CCLA is problematic in (1), (3), (5), (7) and (8). SVM is significantly limited in (1), (2) and does not meet (5), (6), (7) and (8). IHDR does well in (1) through (6) but does not satisfy (7) and (8). MILN introduced here is the only computational network designed for all Property (1) through (8). A summary of these comparison is provided in Table 1.

In what follows, we first present the network structure in Section 2. Section 3 explains the in-place learning mechanism within each layer. Section 4 discusses how invariance is learned by the same in-place learning mechanism while supervision signals are incorporated. Experimental examples are presented in Section 5. Section 6 provides some concluding remarks.

2. Network architecture

The Multi-layer In-place Learning Network takes a vector as input. The vector can be considered as the entry lines to the sensorimotor pathway, e.g., the projection from LGN. For simulations, the input vector corresponds to an image, although the corresponding biological pathway may require the retina and LGN. Biologically, each component in the input vector corresponds to the firing rate of the source neuron. The output of the network corresponds to motor signals. For example, each node in the output layer corresponds to the control signal (e.g., voltage) of a motor. Mathematically, this kind of network that produces numerical vector outputs is called regressor. This network can also perform classification. For example, the number of nodes in the output layer is equal to the number of classes. Each neuron in the output layer corresponds to a different class. At each time instant t, the neuron in the output layer that has the highest output corresponds to the class that has been classified.

2.1. Biological cortex

Grossberg and Williamson (2001) were among the first to model laminar cortex. According to the cortex model by Callaway (1998), among the layers L1 to L6 of the cortex, L4 and L2/3 are two feature layers, while L5 assists L2/3 for inter-neuron competition and L6 assists L4 for inter-neuron competition. L1 contains mostly axons for signal transmission. Therefore, each layer of MILN corresponds to a feature layer in a cortical area, either L4 or L2/3, with inter-neuron competition imbedded into the layer itself.

Felleman and Van Essen (1991) reported that descending (top-down) projections from later cortical areas end in L2/3 but avoid L4 of the cortex. Therefore, in each cortex, L4 performs unsupervised learning and L2/3 performs supervised learning. Consequently, the sensorimotor pathway through different cortical areas are "paired", in the form of USUS from retina and LGN, where U representing unsupervised layer and S representing supervised layer. There might be many reasons for the cortex to ensure that L4 is unsupervised but L2/3 is supervised. It is clear that supervision is useful for developing discriminant features while neglecting irrelevant components.

Then, why does the cortex require a layer of unsupervised feature layer? Developmentally, an unsupervised layer is complete in that the representation does not neglect components that may be needed for discrimination for some future tasks, given the limited resource (the number of neurons) in the layer or the cortical area. In this paper, we will report our experimental results by comparing the performance of USUS structure and SSSS structure. By default, we will consider a layer of MILN to be either an unsupervised or supervised feature layer in the cortex. We will use layer-k to denote a generic layer of MILN and use Lk to denote the layer-k with the laminar cortex.

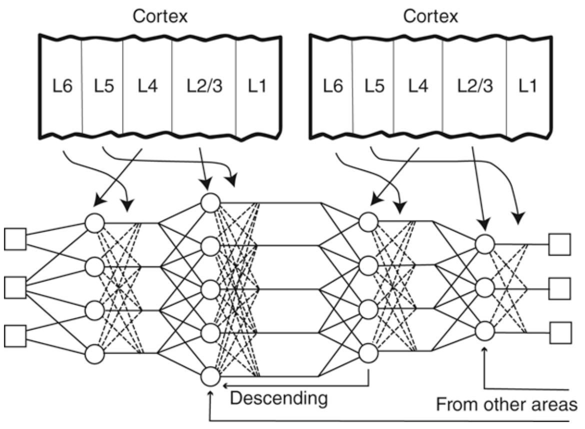

Fig. 1. The Multi-layer In-place Learning Networks and the 6-layer cortex. A circle indicates a cell (neuron). An arrow from a layer in the cortex to a part of MILN indicates the structural correspondence. The connection between a dashed signal line and a cell indicates lateral inhibition. Descending projection indicates supervision. Projection from other areas indicates excitatory or inhibitory supervision signals (e.g., attention selection).

2.2. *MILN architecture*

The architecture of MILN is shown in Fig. 1. Two types of synaptic connections are possible, excitatory and inhibitory.

This is a recurrent network. The output from each layer is not only used as input for the next layer, but is also fed back into other neurons in the same layer through lateral inhibition (dashed lines in the figure). 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 $\mathbf{w}_p$.

The top-down connection vector $\mathbf{w}_p$ 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 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).

3. In-place learning

Let us consider a simple computational model of a neuron. Consider a neuron which takes n inputs, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. The synaptic weight for x_i is w_i, $i = 1, 2, \ldots, n$. Denoting the weight vector as $\mathbf{w} = (w_1, w_2, \ldots, w_n)$, the response of a neuron has been modeled by:

$$y = g(w_1x_1 + w_2x_2 + \cdots + w_nx_n) = g(\mathbf{w} \cdot \mathbf{x}), \tag{1}$$

where g is its nonlinear sigmoidal function.

According to five types learning algorithms defined in Weng and Zhang (2006), an in-place learning algorithm is of Type-5 which means: (1) It is framewise incremental, (2) each neuron does not require the computation of the 2nd or higher statistics of its input, (3) each neuron adapts "in-place" through interactions with its environment and it does not need an extra dedicated learner (e.g., to compute partial derivatives). It is desirable that a developmental system uses an in-place learning due to its simplicity and biological plausibility.

3.1. *Recurrent network*

Recurrent connections, which are ubiquitous in biological networks, are necessary for developing invariant representation. For a recurrent network, all the inputs to a neuron can be divided into three parts: bottom-up input from the previous layer $\mathbf{y}$, lateral inhibition $\mathbf{h}$ from other neurons of the same layer, and the top-down input vector $\mathbf{a}$ from the next layer or other areas. Therefore, the response z from this neuron can be written as

$$z = g(\mathbf{w}_b \cdot \mathbf{y} - \mathbf{w}_h \cdot \mathbf{h} + \mathbf{w}_p \cdot \mathbf{a}), \tag{2}$$

where $(\mathbf{y}, \mathbf{a})$ corresponds to the excitatory input space. In the computation, we simulate the effect of $\mathbf{h}$ as lateral inhibition to allow only top-k winners in the layer to fire, where k is a positive number. Hebbian's rule is used to update the neuron. The weights of a neuron are updated only when the neuron is a winner.

3.2. *Lobe components*

The lobe component theory models the excitatory inputs of a neuron. A concentration of the probability density of the excitatory input space is called a lobe, which may have its own finer structure (e.g., sublobes). 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 excitatory input space $\mathcal{Y}$ into c mutually nonoverlapping regions, called *lobe regions*. $\mathcal{Y} = R_1 \cup R_2 \cup \cdots \cup R_c$. Each region R_i is represented by a single unit feature vector $\mathbf{v}_i$, called the *lobe component*, which is the weight vector of the corresponding neuron. Given an input $\mathbf{y} \in \mathcal{Y}$, many cells – and not only the neuron with $\mathbf{v}_i$ – 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. The projection $(\mathbf{y} \cdot \mathbf{v}_i)$ is a linear part of neuronal projection, or neuronal response. It indicates the "belongingness" of the input $\mathbf{y}$ to the region R_i represented by lobe component vector $(\mathbf{v}_i)$.

According to the theory of Principal Component Analysis (PCA) (e.g., see Jolliffe (1986)), 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 $\lambda_{i,1}\mathbf{v}_i = \Sigma_{y,i}\mathbf{v}_i$. However, in-place learning does not allow a neuron to compute the covariance matrix $\Sigma_{y,i}$, as it is not biologically plausible.

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

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

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.

The concept of statistical efficiency is very useful for minimizing the chance of false extrema and to optimize the speed to reach the desired feature solution in a nonlinear search problem. Suppose that there are two estimators Γ_1 and Γ_2, for a vector parameter (i.e., synapses or a feature vector) $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$, which are based on the same set of observations $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$. If the expected square error of Γ_1 is smaller than that of Γ_2 (i.e., $E\|\Gamma_1 - \boldsymbol{\theta}\|^2 < E\|\Gamma_2 - \boldsymbol{\theta}\|^2$), 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 $\mathbf{y}(1), \mathbf{y}(2), \ldots \mathbf{y}(t-1)$, 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 (1983)). In other words, the estimator in Eq. (5) is nearly optimal given the observations.

3.3. To deal with nonstationary processes

For averaging to be the most efficient estimator, the conditions on the distribution of $\mathbf{x}_t$ are mild. For example, $\mathbf{x}_t$ is a stationary process with exponential type of distribution. However, $\mathbf{x}_t$ depends on the currently estimated $\mathbf{v}_i$. That is, the observations $\mathbf{x}_t$ are from a nonstationary process. In general, the environments of a developing system change over time.

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 (Weng, Zhang, & Hwang, 2003) 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 (Weng et al., 2003).

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

where $\mu(t)$ is the amnesic function depending on t. If $\mu \equiv 0$, the above gives the straight incremental mean. The way to compute a mean incrementally is not new, but the way to use the amnesic function of t for computing a mean incrementally is.

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

in which, e.g., $c = 2, r = 10\,000$. As can be seen above, $\mu(t)$ has three intervals. When t is small, straight incremental average is computed. Then, $\mu(t)$ changes from 0 to 2 linearly in the second interval. Finally, t enters the third section where $\mu(t)$ increases at a rate of about $1/r$, meaning the second weight $(1+\mu(t))/t$ in Eq. (6) approaches a constant $1/r$, to slowly trace the slowly changing distribution.

In the next section $\bar{x}^{(t)}$ is the weight vector of a neuron with its "energy" as the length. It is clear from Eq. (6) that the increment of the "neuronal" synapse vector $\bar{x}^{(t)}$ can not be expressed by partial derivatives (or finite difference from $\bar{x}^{(t-1)}$ for the discrete version). This is because the weight for the previous estimate $(t-1-\mu(t))/t$ is not equal to 1. This is especially important for the early updates, as they greatly determine whether the self-organization is trapped into a local structure during the early development. The previous observations along the trajectory of "nonlinear trajectory" are optimally taken into account by the single mean vector estimate in above incremental mean formulation.

3.4. Single layer: In-place learning 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 first presented in Weng and Zhang (2006). As an alternative model for ICA, LCA is based on Hebbian learning. With LCA, each neuron is a feature detector based on Hebbian learning. Through competition between neurons of the same layer (e.g., lateral inhibition or top-k response rule), each neuron is assigned a region in the input space, where the synaptic weight vector corresponds to the principal component for the region to which the neuron is assigned through competition. Graphically, the direction of the lobe component is the direction of the principal component vector

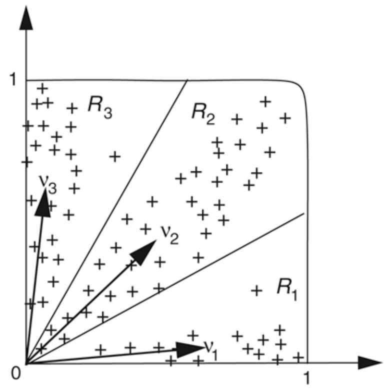

Fig. 2. The illustration of lobe components where all the input components are non-negative. The input space (2-D in the figure) is partitioned into a number of regions (R_1, R_2, and R_3 in the figure) where the number equals to the number of neurons (3 in the figure) available in a layer. The partition is the result of Hebbian learning through lateral competition (lateral inhibition or top-k response rule). The direction of the lobe component vector is the direction of the principal component vector and the length is the standard deviation of the projections in the region onto the direction of lobe component vector.

and the length of the lobe component is the standard deviation of the projections of the region onto the lobe component direction, as illustrated in Fig. 2.

With c neurons available from the layer, the CCI LCA algorithm incrementally updates c such cells (neurons) represented by the column vectors $\mathbf{v}_1^{(t)}, \mathbf{v}_2^{(t)}, \ldots, \mathbf{v}_c^{(t)}$ from input samples $\mathbf{y}(1), \mathbf{y}(2), \ldots$ 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 $\mathbf{z} = (z_1, z_2, \ldots, z_c)$. 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 $t = 1, 2, \ldots, c$.
2. For $t = c+1, c+2, \ldots,$ 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:
   $$z_i = g_i\left(\frac{\mathbf{y}(t) \cdot \mathbf{v}_i^{(t-1)}}{\|\mathbf{v}_i^{(t-1)}\|}\right), \tag{8}$$
   $g_i(x)$ can take the form of:
   $$g(x) = \frac{1}{1 + \mathrm{e}^{-(x/\sigma)}} \tag{9}$$
   where σ is the standard deviation of x.
   (b) Simulating lateral inhibition and sparse coding (Field, 1994) as a result of lateral inhibition, use the top-k response rules. Rank $k+1$ top winners only (to avoid high computational complexity) so that after ranking, $z_1 \ge z_2 \cdots \ge z_k \ge z_{k+1}$, as the belongingness of $y(t)$ to $k+1$ regions. Use a linear function to scale the response:
   $$z_i' = (z_i - z_{k+1})/(z_1 - z_{k+1}), \tag{10}$$
   for $i = 1, 2, \ldots, k$.
   (c) Simulate homeostasis. Let the total weight sum be $s = \sum_{i=1}^{k} z_i'$. The normalized nonzero responses sum to 1 across the layer:
   $$z_i = z_i'/s, \tag{11}$$
   for $i = 1, 2, \ldots, k$. This step normalizes the total "energy" of the response.
   (d) Update only the top k winner neurons $\mathbf{v}_j$, for all j in the set of top k winning neurons, using its temporally scheduled plasticity:
   $$\mathbf{v}_j^{(t)} = w_1 \mathbf{v}_j^{(t-1)} + w_2 z_j \mathbf{y}(t), \tag{12}$$
   where the scheduled plasticity is determined by its two age-dependent weights:
   $$w_1 = \frac{n(j) - 1 - \mu(n(j))}{n(j)}, \qquad w_2 = \frac{1 + \mu(n(j))}{n(j)}, \tag{13}$$
   with $w_1 + w_2 \equiv 1$. Update the number of hits (cell age) $n(j)$ only for the winner: $n(j) \leftarrow n(j) + 1$.
   (e) All other neurons keep their age and weights 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, Schwartz, and Jessell (2000) p. 463). The winner updating rule is a computer simulation of the Hebbian rule (see, e.g., Kandel et al. (2000) 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.

4. Multi-layer network

In theory, two layers of MILN are functionally sufficient to approximate any continuous regression from any multi-dimensional input space and any motor output space to any given desired output accuracy ϵ. For example, the first layer performs self-organizing partition of the input space. The partition is so fine that no two vectors in the input space require output vectors that differ more than ϵ. The second layer maps the prototype (lobe component vector) in every region of the partition to the desired output vector.

4.1. Overview of MILN

It is well known in neuroscience that almost all projections between two cortical areas are bi-directional (see, e.g., the comprehensive study of visual and motor cortices by Felleman and Van Essen (1991), and also Andersen, Asanuma, Essick, and Siegel (1990) and Boussaoud, Ungerleider, and Desimone (1990)). Further, between two consecutive cortical areas, there are as many top-down projections as bottom-up projections. Although the detailed computational principles of these top-down projections are not very clear, for MILN presented here, we model top-down projections as supervision from the next layer or other cortical areas.

The main ideas are as follows. Each MILN is a general-purpose regression network. It develops by taking a series of input-output pairs, whenever the desired output vector is available. Otherwise, given any input without desired output, it estimates the corresponding output vector based on what it has learned so far. This is basically how MILN interleaves the training phases and testing phases in any way that is needed for effective development.

Then, the key question is how to supervise an internal layer when the desired output vector is available. The values of the neurons in the output layer can be totally supervised when the desired output vector is available during supervised learning. The back-propagation learning for a feed-forward network suffers from the local minima problem. The gradient direction of the connection weights violates the biological in-place learning principle (no extra learner is allowed to compute the large matrix of partial derivatives), and it does not take into account the great need of unsupervised learning, in the sense of partition of the input space of each layer while being supervised by later layers. We use "soft supervision" imbedded into the optimal lobe component analysis, so that supervised learning and unsupervised learning are automatically taken into account. Consequently, early layers mainly perform unsupervised learning. While the computation proceeds towards later layers, more supervision is available, thus, more invariance emerges from later layers. Finally the output layer displays almost perfect invariance in a well trained MILN.

Furthermore, a developmental program must deal with different maturation stages of the network. During the very early developmental stage of the network, unsupervised learning should take a dominant role, as supervision is meaningless when the network is very immature. When the network has become more and more mature, supervision can gradually kick in, to gradually improve the invariance of the network. We use time dependent supervision scheduling profile to realize this developmental mechanism.

4.2. Soft supervision

Assuming that the network samples the world at a fast pace so that the response of each layer does not change fast enough compared to the temporal sampling rate of the network, we can then use the output from the later layers at $t-1$ as the supervision signal. Denoting the response vector at layer l at time t to be vector $y^{(l)(t)}$ and its i-th component to be $y_i^{(l)(t)}$, we have the model for computation of this supervised and unsupervised network, for each neuron i at layer l:

$$y_i^{(l)(t)} = g_i^{(l)}(\mathbf{w}_b \cdot \mathbf{y}^{(l-1)(t)} - \mathbf{w}_h \cdot \mathbf{y}^{(l)(t)} + \mathbf{w}_p \cdot \mathbf{a}^{(t-1)}). \tag{14}$$

Note that the i-th component in $\mathbf{w}_h$ is zero, meaning that neuron i does not use the response of itself to inhibit itself.

Using this simple model, the belongingness can take into account not only unsupervised belongingness of $\mathbf{y}^{(l-1)(t)}$ but also the supervised belongingness enabled by $\mathbf{a}^{(t-1)}$. Therefore, the region that belongs to a neuron can be a very complex nonlinear manifold determined by the dynamic supervision signal (e.g., attention), achieving quasi-invariance locally within every neuron of early layers and global invariance at the output layer.

With the top-down projection $\mathbf{a}_i^{(t-1)}$, the Eq. (8) should be changed to:

$$z_i = g_i\left(\frac{\mathbf{y}(t)\cdot\mathbf{v}_i^{(t-1)}}{\|\mathbf{v}_i^{(t-1)}\|} + \mathbf{w}_{pi}^{(t-1)}\cdot\mathbf{a}_i^{(t-1)}\right) \tag{15}$$

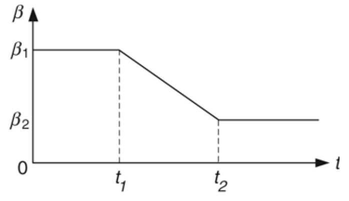

Fig. 3. The dynamic scheduling of supervision using the profile $\beta(t)$. The $\beta(t)$ function is non-increasing. It has three periods, high constant, linearly decreasing, and low constant, respectively.

where $\mathbf{w}_{pi}^{(t-1)}$ is the fan-out vector of neuron i (the weights connected from the neuron i to other neurons) and $\mathbf{a}_i^{(t-1)}$ is the top-down projection vector. When we compute the fan-out vector from the fan-in vector of the next layer, use the normalized versions of all fan-in vectors so that their lengths are all equal to one.

4.3. Dynamic scheduling of supervision

We define a multi-sectional function $\beta(t)$ as shown in Fig. 3 which changes over time. It has two transition points. From $t = 0$ to $t = t_1$, $\beta(t) = \beta_1$ (e.g., $\beta_1 = 1$). From $t = t_1$ to $t = t_2$, $\beta(t)$ linearly decreases to a small β_2 (e.g., $\beta_2 = 0.5$). After $t = t_2$, $\beta(t)$ remains constant.

In the non-iterative version used in CCI LCA, Eq. (15) is changed to the following Eq. (16).

$$z_i = g_i\left(\beta(t)\cdot\frac{\mathbf{y}(t)\cdot\mathbf{v}_i^{(t-1)}}{\|\mathbf{v}_i^{(t-1)}\|} + (1-\beta(t))\cdot\mathbf{w}_{pi}^{(t-1)}\cdot\mathbf{a}_i^{(t-1)}\right). \tag{16}$$

As shown above, the bottom-up (unsupervised) part is weighted by $\beta(t)$ and the top-down (supervision) part is weighted by $1-\beta(t)$. Since these two weights sum to one, the addition of these two weights does not change the total "energy" of the neuronal dendritic inputs. Based on the profile of $\beta(t)$ in Fig. 3, we can see that "soft supervision" is phased in gradually during the development.

The transition times t_1 and t_2 depend on the expected duration of developmental time and the expected time for normal performance for such a learning agent. The biological equivalents of these two numbers are likely coded in the genes as gene regulators, as a result of evolution. For engineering systems, these two numbers can be estimated based on the number of layers and the number of neurons in each layer.

4.4. Multiple layers

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 $j = 1, 2, \ldots, l$, run the LCA algorithm on layer j, $\mathbf{y}_j = \text{LCA}(\mathbf{y}_{j-1})$, where layer j is also updated.
4. Produce output $\mathbf{z}(t) = \mathbf{y}_l$; $t \leftarrow t + 1$.

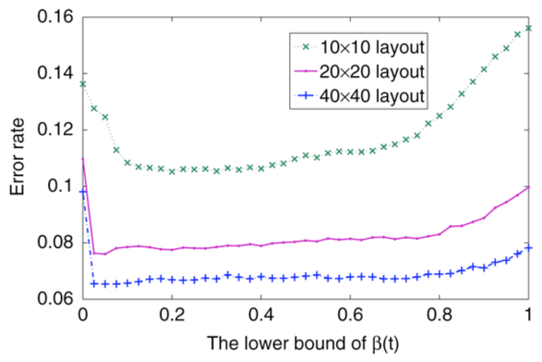

Fig. 4. Error rates versus different amount of supervision denoted by β_2.

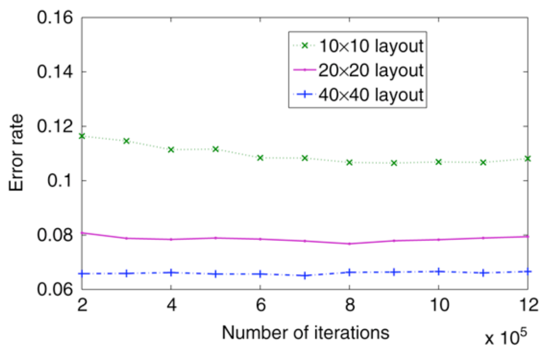

Fig. 5. Error rates versus different second transition point t_2.

5. Experimental examples

As we discussed earlier, a developmental program should be sufficiently general-purpose to enable the development of various skills for an open ended number of tasks that are unknown or not fully predictable during the programming time. At the current state of knowledge, the main purpose of the tasks that we present here concentrates on understanding how MILN works and its strengths and limitations. Tasks that involve complex motor and sensory interactions through autonomous development are in order when the new engine MILN is better understood.

MILN presented above is a general-purpose, biologically more plausible engine for incremental cortical development. For sensory processing, it requires an attention selection mechanism that restricts the connection range of each neuron so that the network can perform local feature detection from the retinal image. Without local analysis of this type, MILN performs global feature-based matching, which should not be expected to outperform other classifiers with local feature analysis in, e.g., character recognition. Here we used handwritten characters. They are good for visualization as their lobe components indicate the corresponding class clearly.

It is important to note that MILN does not assume that the inputs are handwritten characters, thanks to the general strategy of developing task-specific skills from a general-purpose developmental engine. In the last set of experiments reported here, we also present results of feature development from natural images.

5.1. *The combination of self-organization and supervision*

The MNIST database (LeCun, 2007) was used in this experiment. It has 10 classes, from “0” to “9”, with 60,000 training samples and 10,000 testing samples. The size of images in this database is 28×28 pixels. We conducted two sets of experiments for comparison, one with supervision (i.e., top-down connections) and the other without. Each MILN network has two layers. The setting of the experiments are different in the number of neurons in the first layer. Specifically, we have set the number of layer-1 neurons to $10 \times 10 = 100$, $20 \times 20 = 400$, and $40 \times 40 = 1600$, respectively. For each case, k is set to 10, 20, and 40, respectively. All the settings used 3×3 updating. In the training stage, the signal of the output layer was imposed by the given class information. In the testing stage, the neuron of the output layer with largest output gives the recognized class identification. As the network is for generating actions according to sensory inputs, instead of detection of a class of pre-defined patterns, we classify the actions (class labels) as correct or not according to input, instead of false alarm rates. In other words, no action is not a default. No action is an action.

According to the dynamic scheduling function $\beta(t)$, the supervised signals are gradually kicked in from t_1, until $\beta(t)$ reaches its lower bound: β_2. By adjusting the value of β_2, we could control the amount of supervision from later layers. Experiments were conducted to evaluate the effect of different β_2. 20 epochs were run over each layout. t_1 was set to 100,000 and t_2 was set to 600,000. The results are shown on Fig. 4. First, the figure clearly shows that supervision using top-down projection ($\beta_2 < 1$) is better than a totally unsupervised network ($\beta_2 = 1$). Fig. 4 further shows the best performance has been reached over a relatively large area, with a reduction of errors by about 50% compared to the total unsupervised counterpart. Therefore, the results showed that error rates are not very sensitive to β_2. Acceptable results can be obtained when β_2 is larger than 0.1 and smaller than 0.7. The “large basin” phenomenon of the amount of top-down connections is beyond the scope of this paper. The theoretical account will appear in a future paper.

Experiments over different t_2 in the amenesic function $\mu(t)$ were also conducted. In the experiments, β_2 is fixed to 0.1. And the experimental results are shown in Fig. 5. It can be observed from Fig. 5, the error rates vary slightly for different t_2 values. From the meaning of the three sections of amnesic function $\mu(t)$, the transition point t_2 is the transition of a moderate amnesic section toward a constant learning rate section, whose value should not be very sensitive as long as the value is reasonable to the large number of training samples. So, based on the study, we set β_2 to 0.1 and t_2 to 600,000 in later experiments.

5.2. *Comparison for paired laminar architecture*

In this section, we conducted experiment on paired layer settings in an MILN with 4 layers. As discussed earlier for biological laminar cortex, the settings from the bottom to top layers are “paired” – unsupervised–supervised–unsupervised–supervised

Table 2
Error rates: Paired setting helps to reduce error rates

β_2	Error rate (%)	
	Unpaired (SSSS)	Paired (USUS)
0	0.2069	0.2198
0.1	0.2248	0.2187
0.2	0.2247	0.2149
0.3	0.2349	0.213
0.4	0.2567	0.2119
0.5	0.2514	0.2081
0.6	0.2516	0.2066
0.7	0.2505	0.2057
0.8	0.2501	0.2058
0.9	0.2514	0.2055
1	0.2457	0.2069

(labeled as 'USUS') – and "unpaired" – all supervised (labeled as 'SSSS'). In the experiment, the number of layer-1 neurons is $10 \times 10 = 100$. In 'USUS' setting, experiments are conducted on β_2 value of layer-2 varying from 0.1 to 1.0. In 'SSSS' setting, β_2 values of layer-1 and layer-3 are all set to 0.1. And experiments were also conducted on β_2 value of layer-2 varying from 0.1 to 1.0. The experimental results are a summarized in Table 2. It can be observed from Table 2 that the paired architecture performed better than the unpaired architecture. At this point, we do not yet obtained a solid theoretical account for the reasons behind this performance difference. Here is only a hypothesis. Intuitively, an unsupervised layer results in a more even distribution of neuronal resource according to the response distribution of the response of the previous layer, which might be important for an earlier layer. The layer L4 of each cortex takes afferent input from many different brain regions, which is probably a reason for L4 to remain unsupervised, i.e., to maintain a relatively even distribution of neuronal resource before "abstraction" in L2/3.

5.3. *Internal invariance shared by multiple tasks*

The purpose of this experiment is to show how supervised signals can facilitate the development of intermediate "abstract" representations (class of digits in this example). In this experiment, the first task is to recognize digits, "0" to "9." The second task is to classify even numbers and odd numbers. When learning the second task, the intermediate representation for digits "0" to "9" with their "soft" invariances (such as style, positions and sizes) developed in the first task can be well utilized. In other words, the second task does not need to deal with the detailed variations that have been dealt with by the first task. Such multi-task learning is essential for autonomous mental development. Of course, the network is not meant to perform abstract reasoning to symbolically and explicitly explain what digits are even and what digits are odd, although the network connections clearly account for the learned behaviors.

The network has three layers. The number of neurons for each layer is $20 \times 20 = 400$, $5 \times 5 = 25$, and 12, respectively. The 12 nodes in the last motor layer represent digits "0" to "9" and the class "e" of even numbers and the class "o" of odd numbers, respectively. For each layer, k is set to 20, 5, and 1, respectively. We expect that the first layer mainly develops detectors for different variations. The second layer mainly develops an intermediate representation that is more "digits," although a single class of digits should be represented by multiple neurons. The last layer is the motor layer. The true overall invariance should only be reached by the last motor layer.

Table 3
The parameters for $\beta(t)$ profiles

Layer l	β_1	β_2	t_1	t_2
1	1.0	0.5	40,000	180,000
2	1.0	0.2	20,000	120,000

Table 3 provides the parameters of the $\beta(t)$ function used in the experiments. The last layer (motor) is fully supervised when the supervision is applied. As shown, supervision for layer-2 was relatively deeper and was kicked-in earlier than layer-1.

During the process of learning the second task, the first 10 nodes in the motor layer-3 are free to respond and the last two nodes "e" and "o" are supervised. It is true that layer-2 may still adapt during learning for the second task. However, since learning the second task can be readily accomplished using the developed representation in layer-2, only very few examples are sufficient to finish the learning process for task-2. This incremental learning of shared invariant representation facilitates the learning for task-2. In other words, task-2 does not need to learn from scratch.

Fig. 6 shows a visualization of neurons in layer-1, layer-2 and layer-3 as well as their composite weights. For layer-1, the composite weight vector of a neuron is its weight vector. For layer l with $l > 1$, the composite weight vector of a neuron is the sum of all the composite weights $\bar{\mathbf{x}}_i$ of the neurons i in the previous layer, weighted by the normalized synaptic weight w_i of the neuron:

$$\bar{\mathbf{x}} = \sum_{i=1}^{n} \frac{w_i}{\sum_{j=1}^{n} w_j} \bar{\mathbf{x}}_i \tag{17}$$

assuming that the input from the previous layer is n-dimensional. Note that the composite weight vector is only intended as a visualization of multilayer weight summation. It does not take into account the nonlinear effects such as neuronal competition and sigmoidal functions. For more detailed visualization, the broken-down weight vectors of a neuron are defined as the n terms under the summation that gives the composite weight vector in Eq. (17). They show the terms from which the composite vector is summed.

In layer-3, the first 10 neurons represent the single digits 0 to 9 in the first task while the last two neurons represent "even" and "odd" digits in the second task.

Fig. 7 shows the error rates of the multi-task learning (training task-1 followed by training task-2) and single-task learning (training task-2 only from beginning). From the figure, we can see that multi-task learning using MILN reaches comparable error rates faster and gives a smaller error rate at the

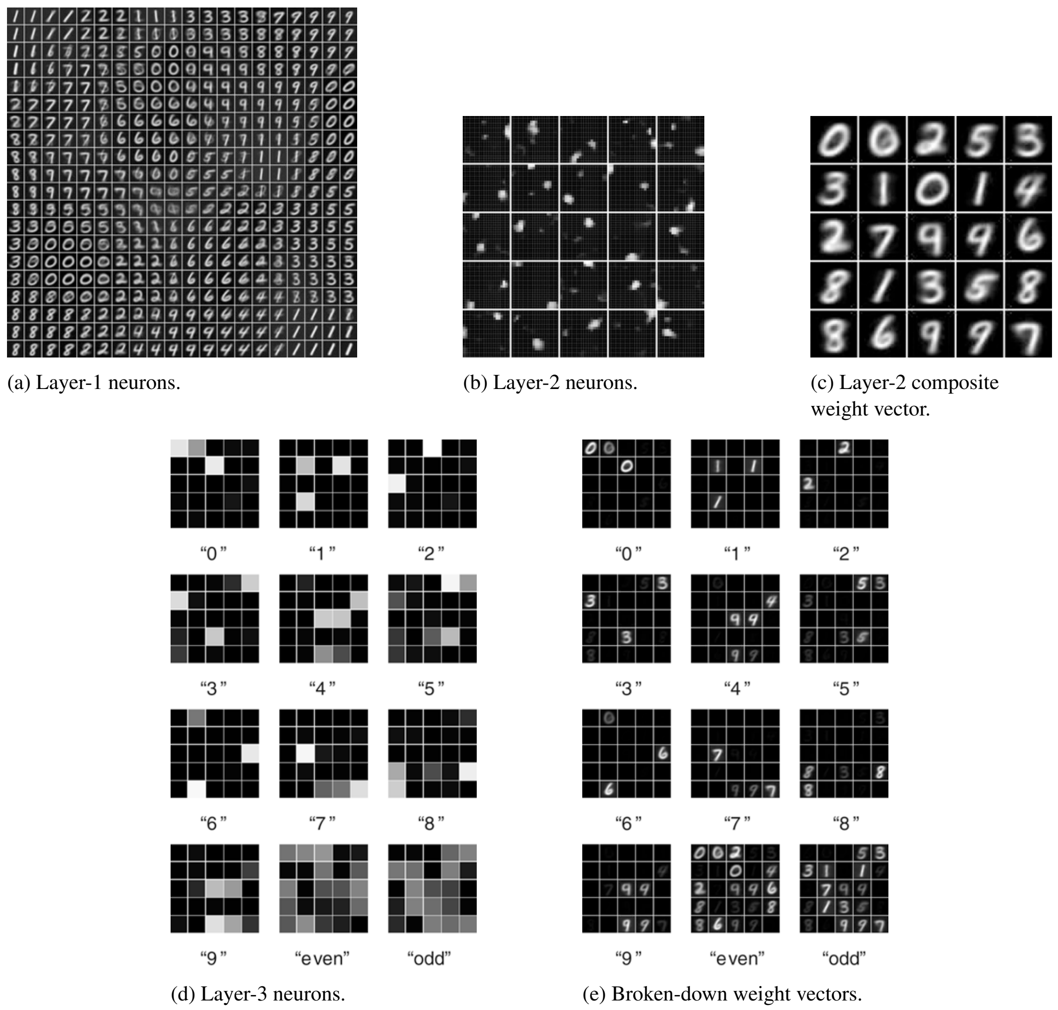
(a) Layer-1 neurons.
(b) Layer-2 neurons.
(c) Layer-2 composite weight vector.
"0" "1" "2" "3" "4" "5" "6" "7" "8" "9" "even" "odd"
(d) Layer-3 neurons.
(e) Broken-down weight vectors.

Fig. 6. A visualization of neurons developed from multi-task learning. (a) Bottom-up weight vectors of 20 × 20 layer-1 neurons. (b) Bottom-up weight vectors of 5 × 5 layer-2 neurons. (c) Composite weight vectors of 5 × 5 layer-2 neurons. (d) Bottom-up weight vectors of 12 (output) layer-3 neurons. (e) Broken-down weight vectors of 12 (output) layer-3 neurons.

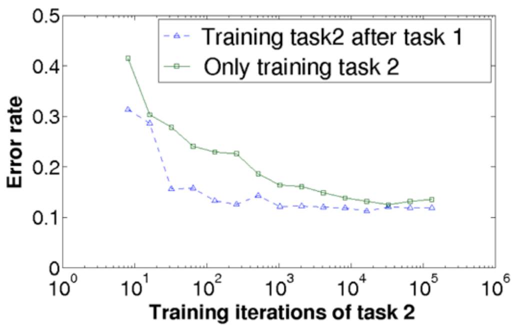

Fig. 7. Error rates of multi-task learning and single-task learning.

same time, indicating a benefit of using developmental learning by MILN.

5.4. Features and receptive fields from natural images

To simulate the process of developing MILN from natural scenes by a developmental robot, we extracted many image patches of random sizes and random positions from 13 natural images (Sinkkonen, 1997). Each image patch simulates an attention fixation on the corresponding image. Each image patch is scaled to the standard size of 16 × 16-pixel before being fed into MILN for development.

The developed feature vectors in the first layer are shown in Fig. 8. They resemble the oriented edge detectors. The smooth self-organization of the features across the 2-D neuronal surface is largely due to the 3 × 3 updates. As a comparison, we also show the developed feature vectors by just randomly selecting 16 × 16-pixel image patches from the same thirteen images, without taking different sizes of the image patches and then normalizing their sizes. The corresponding features developed in layer 1 are shown in Fig. 8 (b). In other words, the results in (a) take into account the size variation but those in (b) do not. The results in (a) and (b) are different, but statistically similar. These feature detectors could potentially be used for developing desired sensorimotor behaviors of a robot in the natural world, which is our planned future work.

As a newborn baby begins to see the natural environment at arbitrary positions and attentions, it is not absolutely certain at this stage of knowledge which scheme among the two is closer to sensorimotor development in the early visual pathways. The

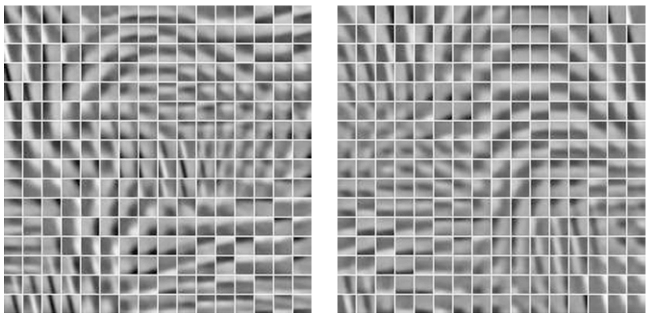
(a) With size variation. (b) Without size variation.

Fig. 8. Lobe components from natural images.

latter scheme (b) does not re-normalize attended images and might be closer to the biological development of neurons in a cortical area having neurons with similar sizes of receptive fields. But this is only a hypothesis that assumes that later cortical processing (e.g., prefrontal cortex) integrates responses from neurons with very different sizes of receptive fields.

6. Conclusions

Although it is known that all the cerebral cortices – visual cortex, parietal cortex, frontal cortex, or motor cortex – have a 6-layer architecture, their functional and computational rules have been illusive. The modeling and simulation work presented here suggest that the 6-layer laminar cortex has two functional layers, L4 and L2/3, while L4 performs mainly unsupervised learning and L2/3 performs supervised learning.

The result reported here has indicated that the top-down connections enable L2/3 to purposively "abstract", generating increasingly more abstract representations that the downstream cortex needs, or eventually what motor needs. Therefore, it is not true that cortex cannot "abstract" based on sub-symbolic (i.e., numerical) representations. Invariance and abstraction are two facets of the same function under complex sensory inputs: While within-class variation is disregarded by within-class invariance and the between-class variation is kept, an abstraction of the classes in the input is manifested in the representation. Such internal abstraction facilitates new task learning, if the later tasks can take advantage of the generated abstraction. The mechanism of abstraction in the MILN model of the laminar cortex involves the top-down connections from the down-stream motor layer and the lobe component analysis in the layer, but not through error back-propagation used in various existing error back-propagation learning algorithms.

Potentially, MILN can be used as a core technology for a general agent with multiple sensors and multiple effectors to incrementally develop perceptual and cognitive skills through real-time sensorimotor experiences. It is important to note that although we used handwritten characters as input stimuli, the system did not have any rule that uses this information. We plan to test the performance of MILN for more applications. A version of the MILN source code is available at the website of MSU EI Lab (Luciw & Weng, 2007).

Acknowledgments

This work was supported in part by MoE Foundation under contract 104075 and Shanghai Municipal R&D Foundation under contracts 065115017 and 06DZ15008.

References

Andersen, R. A., Asanuma, C., Essick, G., & Siegel, R. M. (1990). Cortico-cortical connections of anatomically and physiologically defined subdivisions within the inferior parietal lobule. *Journal of Comparative Neurology*, *296*, 65–113.

Boussaoud, D., Ungerleider, L. C., & Desimone, R. (1990). Pathways for motion analysis: Cortical connections of the medial superior temporal and fundus of the superior temporal visual areas in the macaque. *Journal of Comparative Neurology*, *296*, 462–495.

Callaway, E. M. (1998). Local circuits in primary visual cortex of the macaque moneky. *Annual Review of Neuroscience*, *21*, 47–74.

Fahlman, S.E., & Lebiere, C. (1990). *The cascade-correlation learning architecture*. *Tech. Rep. CMU-CS-90-100*. Pittsburgh, PA: School of Computer Science, Carnegie Mellon University.

Felleman, D. J., & Van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex*, *1*, 1–47.

Field, D. (1994). What is the goal of sensory coding? *Neural Computation*, *6*, 559–601.

Grossberg, S., & Williamson, J. R. (2001). A neural model of how horizontal and interlaminar connections of visual cortex develop into adult circuits that carry out perceptual grouping and learning. *Cerebral cortex*, *11*, 37–58.

Jolliffe, I. T. (1986). *Principal component analysis*. New York: Springer-Verlag.

Kandel, E. R., Schwartz, J. H., & Jessell, T. M. (Eds.) (1991). *Principles of Neural Science* (3rd ed.). Norwalk, Connecticut: Appleton & Lange.

Kandel, E. R., Schwartz, J. H., & Jessell, T. M. (Eds.) (2000). *Principles of neural science* (4th ed.). New York: McGraw-Hill.

LeCun, Y. (2007). The MNIST database of handwritten digits. [Online]. Available: http://yann.lecun.com/exdb/mnist.

Lehmann, E. L. (1983). *Theory of point estimation*. New York: John Wiley and Sons, Inc..

Luciw, M.D., & Weng, J. (2007). The MILN source code. [Online]. Available: http://www.cse.msu.edu/ei/software.htm.

Purves, W. K., Sadava, D., Orians, G. H., & Heller, H. C. (2004). *Life: The science of biology* (7th ed.). Sunderland, MA: Sinauer.

Sinkkonen, J. (1997). Natural image collection for ICA experiments. [Online]. Available: http://www.cis.hut.fi/projects/ica/data/images/.

Weng, J., & Hwang, W. (2007). Incremental hierarchical discriminant regression. *IEEE Transactions on Neural Networks*, *18*(2), 397–415.

Weng, J., & Zhang, N. (2006). In-place learning and the lobe component analysis. In *Proc. IEEE world congress on computational intelligence*.

Weng, J., Zhang, Y., & Hwang, W. (2003). Candid covariance-free incremental principal component analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *25*(8), 1034–1040.

Weng, J., Luwang, T., Lu, H., & Xue, X. (2007). The multilayer in-place learning network for the development of general invariances and multi-task learning. In *Proceedings of IJCNN '07*.