A MULTILAYER IN-PLACE LEARNING NETWORK FOR DEVELOPMENT OF GENERAL INVARIANCES

JUYANG WENG^{*,†,‡}, TIANYU LUWANG^{*,§}, HONG LU^{*,¶}
and XIANGYANG XUE^{*,||}

**Department of Computer Science and Engineering,
Fudan University, Shanghai, China*

*†Department of Computer Science and Engineering,
Michigan State University, East Lansing, MI 48824, USA*

‡weng@cse.msu.edu

§lwty@fudan.edu.cn

¶honglu@fudan.edu.cn

||xyxue@fudan.edu.cn

Received 22 September 2006

Accepted 9 February 2007

Currently, there is a lack of general-purpose, in-place learning engines that incrementally learn multiple tasks, to develop “soft” multi-task-shared invariances in the intermediate internal representation while a developmental robot interacts with its environment. In-place learning is a biologically inspired concept, rooted in the genomic equivalence principle, meaning that each neuron is responsible for its own development while interacting with its environment. With in-place learning, there is no need for a separate learning network. Computationally, biologically inspired, in-place learning provides unusually efficient learning algorithms whose simplicity, low computational complexity, and generality are set apart from typical conventional learning algorithms. We present in this paper the multiple-layer in-place learning network (MILN) for this ambitious goal. As a key requirement for autonomous mental development, the network enables both unsupervised and supervised learning to occur concurrently, depending on whether motor supervision signals are available or not at the motor end (the last layer) during the agent’s interactions with the environment. We present principles based on which MILN automatically develops invariant neurons in different layers and why such invariant neuronal clusters are important for learning later tasks in open-ended development. From sequentially sensed sensory streams, the proposed MILN incrementally develops a hierarchy of internal representations. The global invariance achieved through multi-layer invariances, with increasing invariance from early layers to the later layers. Experimental results with statistical performance measures are presented to show the effects of the principles.

Keywords: In-place learning; positional invariance; size invariance; style invariance; incremental learning; internal representation; biologically inspired network.

1. Introduction

Existing study in neural science has provided much knowledge about biological multi-layer networks. Examples of 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),⁹ support vector machines (SVM), and incremental hierarchical discriminant regression (IHDR).^{16,44} Much research already exists in the field of learning using supervised and unsupervised networks.

However, development has raised requirements that existing methods cannot meet. The genome (the biological developmental program of the species) is task-nonspecific⁴⁵ in the sense that the genome must enable the agent, through the process of epigenetic process of development, to perform an open-ended number of tasks that the sensors, effectors and the environment allow. For mental development, the biological brain performs online, real-time, incremental learning using sequentially arriving high-dimensional sensory and motoric signal streams through the life-long time axis without suffering from the local minima problem and the loss of memory problem. By the local minima problem, we mean that the performance of networks updated by gradient-based methods is moderately sensitive to the initial guess of their connection weights. By the loss of memory problem, we mean that a gradient-based method fits the network to the new data but also forgets the experience from the older data. The biological central nervous system does not appear to seriously suffer from these two problems — the mind of a normal newborn is expected to develop normally if she is raised in proper human environments. The inferior performance of traditional neural networks can be attributed to many factors. The large scale factors include the lack of large scale developmental plasticity that has been observed with biological nervous systems.^{36,37} The local scale factors include the weakness of traditional network adaptation models (such as the gradient-based updating) and violation of the biological in-place learning principle⁴⁶ as a consequence of the genomic equivalence principle.³¹

In fact, adaptation and long-term memory are two conflicting criteria. Striking a balance is a great challenge. Self-organization in cortical layers through life-time is further a highly nonlinear global search issue. 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 network, inappropriate for online autonomous development for an open time period. IHDR avoids 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. This optimality represents the best balance, best dealing with the local minima problem (free local minima) and the loss of memory problem (long term memory). Collectively, all the neurons in a layer represent a globally self-organized distributed representation of the probability density of the layer's high-dimensional input space. This work concentrates on invariance of the network. For detailed properties of the lobe component analysis and the performance comparison, the reader is referred to Ref. 46.

Before we present the new network that addresses the above challenges, we first discuss some important issues related to autonomous development. The following discussion is motivated by the development of biological systems but the points raised are not necessarily well accepted by many traditional engineering systems that do not deal with development.

(i) Innate orientated edge features? Orientation selective cells in cortical area V1 are well known,^{8,14,15,34} but the underlying computational principles that guide their emergence (i.e. development) are still elusive. Are cells in V1 totally genetically wired to detect orientations of visual stimuli? This issue is important since the later layers of network, with their larger receptive fields, need to detect features that are more complex than oriented edges, such as edge groupings (T, X, Y, cross, etc.), patterns, object parts, human faces, etc. A classical study by Blakemore and Cooper 1970⁵ reported that if kittens were raised in an environment with only vertical edges, only neurons that respond to vertical or nearly vertical edges were found in the primary visual cortex (also see, more recent studies, e.g. Ref. 33). From a computational point of view, hand-programmed (or genome-coded) edge detectors are not sufficient for all the possible sensory environments. In the model presented here, feature detectors are developed through interactions with cell's environment, instead of hand-programmed.

(ii) Developmental mechanisms. If the answer to the above is negative, 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 early multi-layer pathway of visual processing. In this project, we propose intra-layer developmental mechanisms as well as inter-layer developmental mechanisms. The within-layer mechanisms are based on two well-known biological mechanisms, *Hebbian learning* and *lateral inhibition*, but we further their optimality. The inter-layer developmental mechanisms specify how information between layers is passed and used, which enables both unsupervised and supervised learning to occur in the same network. We also propose a scheduling mechanism that deals with the amount of supervision at different maturation times of development. Of course, there are many other mechanisms that we do not examine in this work, such as internally generated spontaneous signals which assist the development of cortical networks before birth.³⁰

- (iii) **In-place development.** By in-place development, we mean that each neuron deals with its own development through its own internal physiological mechanisms and interactions with other neurons and, thus, there is no need to have an extra network that accomplishes the learning (adaptation). The in-place learning principle is a result of the well known principle called *genomic equivalence*, namely, each neuron carries the entire set of genes (i.e. the genome, the developmental program). In other words, the biological developmental program is cell-centered. There is no extra developmental program dedicated to any collection of more than one cell. It is desirable that a developmental system uses an in-place developmental program, due to not only the biological plausibility but also the high degree of plasticity that development requires. We further propose that biological in-place learning mechanisms can facilitate our understanding of biological systems. From computational point of view, in-place learning leads to unified, very simple and very computationally efficient learning algorithms for the very challenging problem of incremental learning from real-time sensory streams. No other known algorithms come close in terms of computational efficiency for such incremental learning problems.⁴⁶
- (iv) **Integration of unsupervised and supervised learning.** There exist many learning algorithms for self-organizing a multi-layer network that corresponds to a network of feature detectors. Well-known unsupervised learning algorithms include self-organizing maps by Kohonen,²³ vector quantization,¹ principal component analysis (PCA),²¹ independent component analysis (ICA),^{17,19} Isomap,³⁸ and non-negative matrix factorization (NMF).²⁵ Only a few of these algorithms have been expressed by in-place versions (e.g. SOM and PCA⁴⁷). 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,⁹ support vector machines,⁷ and hierarchical discriminant regression (HDR).¹⁶ However, it is not convincing that biological networks use two different types of networks for unsupervised and supervised learning, which may occur concurrently in the process of development. Currently, there is a lack of biologically inspired networks that integrate these two different learning modes using a single learning algorithm. Motivated by the well accepted fact that most connections between visual cortical areas are two-way (see Ref. 10 for an overview), this work proposes a biologically inspired general-purpose network that unifies both modes of learning, allowing them to be performed in any order desired by the developmental process.
- (v) **Rich completeness of representation at each layer.** Due to the need for representation of all possible receptive fields at any retinal position of any size, the sensory pathway of the human vision system adopts a scheme of layered-representation (e.g. V1, V2, V4, PIT), each corresponding to a different scale of receptive fields. The response from a layer provides a rich representation for all the possible receptive fields of the scale range represented by the layer.

The completeness for all the possible receptive fields is essential for attention selection (e.g. focus attention to a human face of a particular size and position on the retina). The biologically inspired candid covariance-free incremental lobe component analysis (CCI LCA), modeling some major mechanisms within a cortical area, is suited for high-dimensional incremental self-organization of such a rich representation, due to coarse-to-fine competition imbedded by its nearly optimal statistical efficiency (i.e. minimum error in the estimate of the lobe component). In the model presented here, synaptic weights that have significant non-zero value contribute to the receptive fields but the detailed properties receptive fields are beyond the scope of this work.

- (vi) **Invariant representation.** Hubel and Wiesel^{13,14} suggested early that a logic-OR type of learning for edges presented at different positions in the receptive field might partially explain how positional invariance may arise in V1. Studies have shown that a normally developed human visual system has some capabilities, but are not perfect, in translational invariance,²⁸ scale invariance,²⁴ or orientational invariance.³⁹ Therefore, it seems biologically incorrect, even though convenient to engineer, to impose strict invariance in position, scale, or orientation. From the developmental point of view, such an imposition will significantly restrict the system's ability to learn other perceptual skills. For example, when a square is rotated by 45° , the shape is called a diamond; and the number "6" rotated by 180° is called "9". A type of complete representation scheme that is able to learn all open-ended tasks as well as their embedded invariances is a great challenge that this work aims at.

Why invariance? For learning a single given task, invariant features are useful for generalization along the factors that are known, *a priori*, to be invariant, as we discussed above. However, for autonomous mental development, tasks that a robot will learn are unknown during the time of programing.⁴⁵ Task-specific invariance then must be learned from experience. However, it seems sufficient to display invariance at the output (motor) layer and only "soft invariance" is needed at earlier layers. Why is internal learned "soft" invariance useful? It is useful for tasks that share similar invariant properties, although any invariance is not always useful for all the possible tasks, as explained above. For example, consider two tasks. The first task is to learn to recognize handwritten digits from 0 to 9. The second is to learn to classify two classes, odd digits and even digits. An internal representation that has learned positional, scale, and various style invariances for the first task should be useful for speeding up the learning of the second task, because the internal "soft" invariant representation can be used for the later task. That is, early learned skills facilitate later learning.

It is also important to note that "hard" invariance (e.g. a single internal node represents a single class) is impossible and inappropriate because during autonomous mental development the human trainer is only allowed to directly

supervise motor nodes in the last layer. For example, a human teacher can hold a child's hand (motor) to teach him how to write, but the teacher is not allowed to directly wire into the child's brain. This inaccessibility of the brain is a major challenge for autonomous mental development.

Some networks have built-in (programed-in) invariance, either spatial, temporal or some other signal properties. Neocognitron by Fukushima¹² is a self-organizing multi-layer neural network of pattern recognition unaffected by *shift in position*. Cresceptron by Weng et al.⁴³ has an architecture similar to Fukushima's Neocognitron while its neural layers are dynamically generated from sensing experience and, thus, the architecture is a *function of sensory signals*. The above two networks have built-in shift-invariance in that weights are copied across neurons centered at different retinal positions. However, other types of spatial invariance, such as size, style, and lighting, have to be learned object-by-object (no sub-part invariance sharing). Detection of objects using motion information belongs to built-in invariance based on temporal information. Many other built-in signal properties have been used to develop quasi-invariant feature detectors, such as color (e.g. human skin tones), saliency, slow in response (e.g. slow feature analysis³²) and the independence between norms of projections on linear subspaces.¹⁸

Other networks do not have built-in invariance. The required invariance is learned object-by-object. The self-organizing maps proposed by Kohonen and many variants²³ belong to this category. The self-organizing hierarchical mapping by Zhang and Weng⁴⁸ was motivated by *representation completeness* using incremental principle component analysis and showed that the neural coding can reconstruct the original signal to a large degree. Miikkulainen et al.²⁷ have developed a multi-layer network with nearby *excitatory interactions* surrounded by *inhibitory interactions*. There have been many other studies on computational modeling of retinotopic networks^a (e.g. Refs. 11, 29 and 41). While this type of network has a superior discrimination power, the number of samples needed to reach desired invariance is very large.

The work reported here proposes a new, general-purpose, multi-layer network, which learns general invariance from experience. The network is biologically inspired (e.g. in-place learning), but is not necessarily biologically fully provable at the current stage of knowledge about the brain. The network has multiple layers; later layers take the response from early layers as their input. Although it is not necessary that a later layer only takes input from its previous layer, we use this restriction in our experiment to facilitate understanding. It is known that each cortical area receives projections not only from early cortical areas, but also from its neighbors and other later processing cortical areas.³⁵ The network enables supervision from two types of projections: (a) supervision from the next layer, which also receives supervision recursively from the output motor layer (the last layer for motor output); (b) supervision from other cortical regions (e.g. motor layer) as attention

^aEach neuron in the network corresponds to a location in the receptor surface.

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. Every layer has the potential to develop invariance, with a generally increased invariance at later layers, with support of lesser invariance from early layers.

The novelty of the work presented here is indicated by the two-way projection model for the general purpose end-to-end multilayer network, together with a scheme for scheduling the degree of supervision based on the maturity of each layer. The importance of this work is indicated by a combination of the following challenging properties that are required by development but none of the existing networks can meet all: (i) high-dimensional input; (ii) incremental learning; (iii) without a significant local minima problem for regression; (iv) without a significant loss of memory problem; (v) in-place learning; (vi) enable supervised and unsupervised learning in any order suited for development; (vii) local-to-global invariance from early to later processing; (viii) 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 (i), (iii), (iv), (v), (vii), and (viii). RBF has major problems with (iii), (iv), (vii), and (viii). SOM does not meet (vi), (vii), and (viii). CCLA is problematic in (i), (iii), (v), (vii), and (viii). SVM is significantly limited in (i), (ii) and does not meet the requirements (v)–(viii). IHDR does well in (i) through (vi) but does not satisfy (vii) and (viii). The biologically inspired MILN introduced here is the only computational network designed for all properties (i) through (viii).

The real-time speed is also an important property of network development. The proposed MILN has the lowest possible order of time complexity (see, e.g. Ref. 47) among all the existing artificial networks that satisfy the above requirement (viii), due to their being free of computation for the matrix of second or higher order statistics and free of computation for partial derivatives. However, the actual processing speed for each input frame depends on the hardware. We report the software updating times which are well within the limit required by the 30 Hz updating speed.

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

2. The In-Place Learning Network

This section presents the architecture of the new multi-layer in-place learning network (MILN).

The network takes a vector as input (a set of receptors). For vision, the input vector corresponds to a retinal image. For example, the input at time t can be an

image frame at time t , or the few last consecutive frames before time t , of images in a video stream. Biologically, each component in the input vector corresponds to the firing rate of the receptor (rod or cones) or the intensity of a pixel.

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 a *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.

The architecture of the multi-layer network is shown in Fig. 1. For biological plausibility, assume that the signals through the lines are *non-negative* signals that indicate the firing rates. 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 feed 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:

- (i) bottom-up (excitatory) weight vector $\mathbf{w}_b$ that links input lines from the previous layer $l - 1$ to this neuron;

Fig. 1. The architecture of the multi-layer in-place learning network. 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).

- (ii) lateral (inhibitory) weight $\mathbf{w}_h$ that links other neurons in the same layer to this neuron;
- (iii) top-down (excitatory or inhibitory) weight $\mathbf{w}_p$. 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 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 at discrete times, $t = 0, 1, 2, \dots$, as a series of open-ended developmental experiences 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 . Because the adaptation of each neuron also uses top-down weights, the imposed motor signal at the output layer can affect the learning of every neuron through many passes of computations.

3. In-Place Learning

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

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

where g is its nonlinear sigmoidal function, taking into account under-saturation (noise suppression), transition, and over-saturation, and $\cdot$ denotes dot product. Next, we discuss various types of algorithms that learn the weight vector w and the sigmoidal function g .

3.1. Learning types

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 $B = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_b\}$, where b is the batch size. That is, $(g, \mathbf{w}) = L_1(B)$, where the argument B is the input to the learner, L_1 , and the right side is its output. This learning algorithm needs to store an entire batch of input vectors B before learning can take place.

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. The Type-3 algorithm below explains how to learn incrementally with block size $b = 1$.

Type-3 incremental: Each input vector must be used immediately for updating the learner's memory (which must not store all the input vectors) and then the input must be discarded before receiving the next input. Type-3 is the extreme case of Type-2 in the sense that block size $b = 1$. At time t , $t = 1, 2, \dots$, taking the previous learner memory $M^{(t-1)}$ and the current input $\mathbf{x}_t$, the learning algorithm L_3 updates $M^{(t-1)}$ into $M^{(t)}$: $M^{(t)} = L_3(M^{(t-1)}, \mathbf{x}_t)$. The learner memory $\mathbf{M}^{(t)}$ is needed to determine the neuron N at time t , such as its sigmoidal $g^{(t)}$ and its synaptic weight $\mathbf{w}^{(t)}$. It is widely accepted that every neuron learns incrementally.

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 second or higher order statistics of the input $\mathbf{x}$. In other words, the learner's memory $M^{(t)}$ cannot contain the second order (e.g. correlation or covariance) or higher order statistics of $\mathbf{x}$. The CCI PCA algorithm⁴⁷ is a covariance-free incremental learning algorithm for computing principal components as the weight vectors of neurons.

Type-5 in-place neuron learning: A Type-5 learning algorithm L_5 is a Type-4 algorithm. However, the learner L_5 must be implemented by the signal processing neuron N itself.

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. In this work, we deal with Type-5 (in-place) learning.

3.2. Recurrent network

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 inputs 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, 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 $\mathbf{w}_p$. Therefore, the response z from this neuron can be written as

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

Since this is a recurrent network, the output from a neuron will be further processed by other neurons whose response will be fed back into the same neuron again.

In real-time neural computation, such a process will be carried out continuously. If the input is stationary, an equilibrium can possibly be reached when the response of every neuron is no longer changed. However, this equilibrium is unlikely even if the input is static since neurons are band-pass filters: a temporally constant input to a neuron will quickly lead to a zero response.

For a digital computer, we simulate this analogue network through discrete times $t = 0, 1, 2, \dots$. 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.

3.3. *Lateral inhibition*

For simplicity, we will first only consider the first two parts of the input: the input from the previous layer $\mathbf{y}$, and the lateral inhibition part from $\mathbf{h}$.

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.

Since each neuron needs the output of other neurons in the same layer and they also need the output from this neuron, a direct computation will require iterations, which is time consuming. We realize that the net effect of lateral inhibition is (a) for the neurons with strong outputs to effectively suppress weakly responding neurons, and (b) for the weak neurons to less effectively suppress strongly responding neurons. Therefore, we can use a computationally more effective scheme to simulate lateral inhibition without resorting to iterations. Sort all the responses. Make top- k responding neurons have non-zero response. All other neurons have zero response (i.e. does not go beyond the under-saturation point).

Larger k gives more information about the position of the input in relation with the top- k winning neurons. However, multiple responses do not help when the density of the prototypes is low. In our experiments, k is set smaller than 20% of the number of neurons and the performance is similar in that region.

It is clear that iteration is slow and the one-pass, top- k computation is fast. However, iterations may result in many passes of wave propagation through back-and-forth inhibitive connections. Therefore, iterative computation and the one-pass, top- k computation are not equivalent, theoretically. Our use of top- k is primarily motivated by computational efficiency. For the same reason, we choose to model neuronal firing rates instead of modeling individual spikes in biological neurons.

3.4. *Hebbian learning*

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

Fig. 2. Neurons adapt using lateral inhibitions and Hebbian learning.

high. Figure 2 illustrates the lateral inhibition and Hebbian learning. In the figure, hollow triangles indicate excitatory connections, and the solid triangles indicate inhibitory connections.

However, this intuitive observation will not lead to optimal performance. We need a more careful discussion about the statistical properties of neural adaptation under two mechanisms: lateral inhibition and Hebbian learning. With our further derivation, the lateral inhibition and Hebbian learning will take a detailed form that leads to a quasi-optimality.

3.5. Lobe components

What is the main purpose of early sensory processing? Atick and coworkers^{3,4} proposed that early sensory processing decorrelates inputs. Weng *et al.*⁴⁷ 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. In the visual pathway, the early cortical processing does not totally whiten the signals, but they have similar variance and are weakly correlated. 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 3D).

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

$$Y = R_1 \cup R_2 \cup \cdots \cup R_c, \quad (3)$$

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),²¹ 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

$$\lambda_{i,1}\mathbf{v}_i \approx \frac{1}{n} \sum_{t=1}^n \mathbf{y}(t)\mathbf{y}(t)^\top \mathbf{v}_i = \frac{1}{n} \sum_{t=1}^n (\mathbf{y}(t) \cdot \mathbf{v}_i)\mathbf{y}(t). \quad (4)$$

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) $\theta = (\theta_1, \dots, \theta_k)$, which are based on the same set of observations $S = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$. 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 $\mathbf{y}(1), \mathbf{y}(2), \dots, \mathbf{y}(t-1)$, the neuron can be updated into $\mathbf{v}_i(t)$ using the current sample defined from $\mathbf{y}(t)$ as:

$$\mathbf{x}_t = \frac{\mathbf{y}(t) \cdot \mathbf{v}_i(t-1)}{\|\mathbf{v}_i(t-1)\|} \mathbf{y}(t). \quad (5)$$

In the above expression on the right-hand side, the factor for $\mathbf{y}(t)$ corresponds to a neuronal response. To better handle values beyond the normal range, a biological neuron applies a nonlinear sigmoidal function g . We will adopt this sigmoidal function g in actual algorithm to emulate a biological neuron.

Then Eq. (4) states that the lobe component vector is estimated by the average:

$$\lambda_{i,1} \mathbf{v}_i \approx \frac{1}{n} \sum_{t=1}^n \mathbf{x}_t. \quad (6)$$

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, pp. 429–430 of Ref. 26). In other words, the estimator in Eq. (6) is nearly optimal given the observations.

3.6. For 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. Therefore, the sensory input process is a nonstationary process too. We use the amnesic mean technique below that 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. (6) is a batch method. For incremental estimation, we use what is called an amnesic mean⁴⁷:

$$\bar{x}^{(t)} = \frac{t-1-\mu(t)}{t} \bar{x}^{(t-1)} + \frac{1+\mu(t)}{t} x_t, \quad (7)$$

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 is new for computing a mean incrementally:

$$\mu(t) = \begin{cases} 0 & \text{if } t \leq t_1, \\ c(t-t_1)/(t_2-t_1) & \text{if } t_1 < t \leq t_2, \\ c + (t-t_2)/r & \text{if } t_2 < t, \end{cases} \quad (8)$$

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. (7) 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. (7) that the increment of the “neuronal” synapse vector $\bar{x}^{(t)}$ cannot be expressed by partial derivative (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.

3.7. 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 Ref. 46. The algorithm incrementally updates c such cells (neurons) represented by the column vectors $\mathbf{v}_1^{(t)}, \mathbf{v}_2^{(t)}, \dots, \mathbf{v}_c^{(t)}$ from input samples $\mathbf{y}(1), \mathbf{y}(2), \dots$ 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 $\mathbf{z} = (z_1, z_2, \dots, z_c)$. The quasi-optimally efficient, in-place learning, single layer CCI LCA algorithm $\mathbf{z} = \text{LCA}(\mathbf{y})$ is as follows:

- (i) Sequentially initialize c cells using first c observations: $\mathbf{v}_i^{(c)} = \mathbf{y}(t)$ and set cell-update age $n(t) = 1$, for $t = 1, 2, \dots, c$.
- (ii) For $t = c + 1, c + 2, \dots$, do
 - (a) If the output is not given, compute output (response) for all neurons: for all i with $1 \leq i \leq c$, compute response:

$$z_i = g_i \left(\frac{\mathbf{y}(t) \cdot \mathbf{v}_i^{(t-1)}}{\|\mathbf{v}_i^{(t-1)}\|} \right), \quad (9)$$

where $g_i(x)$ can take the form of:

$$g(x) = \frac{1}{1 + e^{-(x/\sigma)}}, \quad (10)$$

where σ is the standard deviation of x .

- (b) Simulating lateral inhibition, rank $k + 1$ top winners only (to avoid high computational complexity) so that after ranking, $z_1 \geq z_2 \geq \dots \geq z_k \geq z_{k+1}$, as the belongingness of $y(t)$ to $k + 1$ regions. Use a linear function to scale the response:

$$z'_i = (z_i - z_{k+1}) / (z_1 - z_{k+1}), \quad (11)$$

for $i = 1, 2, \dots, k$.

- (c) Homeostasis: Let the total weight sum be $s = \sum_{i=1}^k z'_i$. The normalized nonzero responses sum to 1 across the layer:

$$z_i = z'_i / s, \quad (12)$$

for $i = 1, 2, \dots, k$. This 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:

$$\mathbf{v}_j^{(t)} = w_1 \mathbf{v}_j^{(t-1)} + w_2 z_j \mathbf{y}(t), \quad (13)$$

where the scheduled plasticity is determined by its two age-dependent weights:

$$w_1 = \frac{n(j) - 1 - \mu(n(j))}{n(j)}, \quad w_2 = \frac{1 + \mu(n(j))}{n(j)}, \quad (14)$$

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 $1 \leq i \leq c$, $i \neq j$, $\mathbf{v}_i^{(t)} = \mathbf{v}_i^{(t-1)}$.

The neuron winning mechanism corresponds to the well-known mechanism called lateral inhibition (see, e.g. Ref. 22, p. 463). The winner updating rule is a computer simulation of the Hebbian rule (see, e.g. Ref. 22, 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

With the material we discussed earlier for single layer, we are ready to discuss multi-layer network MILN. In theory, two layers of MILN are sufficient functionally 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.

Then, why do we need more than two layers in MILN? At least two main reasons: first, receptive fields; second, invariance. Both are meant for better generalization with limited number of samples. We discuss each of them below.

First, receptive fields enable local analysis. Functionally, each layer in MILN corresponds roughly to a cortical area (e.g. V1 or V2 in the visual pathway). All the neurons in the same area have different receptive fields (e.g. different locations in space), but their sizes are similar. In a sensorimotor pathway, neurons in an early processing area have smaller receptive fields while those in a later area have larger ones. Such a hierarchical organization of receptive fields enables the brain to select attention. In other words, under certain attention control, response from neurons whose receptive fields correspond to the desired area is enhanced while the response from other neurons is suppressed.^{20,40} The work reported in this paper does not deal with this subject although it provides a basis for it.

Second, the multilayer structure enables better generalization using invariance. Suppose that a neuron p in an earlier layer is invariant to three types of variation in the input (e.g. three types of faces). Then, a neuron p' in the next layer can use this neuron p as well as other neurons in the previous layer for the detection of a more complex type (e.g. human bodies), without a need to directly deal with these three types of variation by itself. This type of increased invariance from an early layer to a later layer is a major focus of this paper. The major issue for MILN is then how to achieve gradually increased invariance from early layers to later layers.

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,¹⁰ and also Andersen *et al.*² and Boussaoud *et al.*⁶). 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. However, as discussed earlier, during autonomous development the human teacher is not allowed to hardwire the internal representation. In particular, he is not allowed to directly hand-specify, at any time, the value of any neuron of any internal layer — those layers other than the input terminals and the output motor layer. Therefore, any supervision can be gradually propagated from the output layer.

However, 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 and, 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*

Revisit the top-down part in the expression in Eq. (2), which requires feedback from later processing. However, for digital networks, the output from later layers for the current input is not available until after the current layer has computed. 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 :

$$y_i^{(l)}(t) = g_i^{(l)}(\mathbf{w}_b \cdot \mathbf{y}^{(l-1)}(t) - \mathbf{w}_h \cdot \mathbf{y}^{(l)}(t) + \mathbf{w}_p \cdot \mathbf{a}(t - 1)). \quad (15)$$

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.

In other words, the output from the later layers $\mathbf{a}$ can take the previous one $\mathbf{a}(t - 1)$. 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 $\mathbf{a}(t - 1)$. For example, when the firing neurons at a layer fall into the receptive field selected by the attention selection signal represented by $\mathbf{a}(t - 1)$, its belongingness is modified dynamically by the supervised signal $\mathbf{a}$. 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.

The next key issue is whether the single-layer in-place learning algorithm can be directly used when the top-down part $\mathbf{w}_p \cdot \mathbf{a}(t - 1)$ is incorporated into the input of every neuron. Let us discuss two cases for the top-down factor $\mathbf{a}$ and its weight vector w_p , as shown in Fig. 3: (a) The part that corresponds to the output from the next layer. Since the fan-out vector $\mathbf{w}_p$ from neuron i corresponds to the weight vector linking to $\mathbf{a}$, the inner product $\mathbf{w}_p \cdot \mathbf{a}$ directly indicates how much the current top-down part supervises the current neuron i . (b) The part that corresponds to the output from other areas. Again the corresponding part in the top-down factor

Fig. 3. The bottom-up projection and top-down projection of a neuron at layer l . $\mathbf{w}_b$ consists of bottom-up weights from neurons in the previous layer to the neuron under consideration. $\mathbf{w}_p$ consists of top-down weights from this neuron to other neurons in the next layer.

$\mathbf{w}_p$ indicates how the supervision should be weighted in $\mathbf{w}_p \cdot \mathbf{a}$. In summary, by including the supervision part $\mathbf{a}$, the single-layer in-place learning algorithm can be directly used.

With the top-down projection $\mathbf{a}_i^{(t-1)}$, Eq. (9) should be changed to

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

where $\mathbf{w}_{pi}^{(t-1)}$ is the fan-out vector of neuron i 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, we 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. 4 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.

Equation (17) shows how the dynamic scheduling of supervision is used by each neuron, by modifying Eq. (15):

$$y_i^{(l)}(t) = g_i^{(l)}(\beta(t) \cdot \mathbf{w}_b \cdot \mathbf{y}^{(l-1)}(t) - \mathbf{w}_h \cdot \mathbf{y}^{(l)}(t) + (1 - \beta(t)) \cdot \mathbf{w}_p \cdot \mathbf{a}(t-1)). \quad (17)$$

Fig. 4. 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.

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

$$z_i = g_i \left(\beta(t) \cdot \frac{\mathbf{y}(t) \cdot \mathbf{v}_i^{(t-1)}}{\|\mathbf{v}_i^{(t-1)}\|} + (1 - \beta(t)) \cdot \mathbf{w}_{pi}^{(t-1)} \cdot \mathbf{a}_i^{(t-1)} \right). \quad (18)$$

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 do not change the total “energy” of the neuronal dendritic inputs. Based on the profile of $\beta(t)$ in Fig. 4, 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.

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

4.5. Topographic map

In the above discussion, the neurons can reside at any locations and their locations are independent of their weights. The concept of topographic map is that neurons that detect similar features are nearby. This property is important for cortical modularization in the sense that the responses for similar inputs appear nearby in the cortical area, naturally forming what are called “modules.” These are also found in the brain: a certain cortical area is responsible for detecting a class of objects (e.g. faces). A topographic map facilitates generalization because similarity of inputs is translated (mapped) into proximity in cortical locations.

To form a topographic cortical map, we assume that neurons lie in a 2D “cortical” surface $\mathcal{S}$. The topographic map can be realized by updating not only the

Fig. 5. The 3×3 neighbor of the winner $A_{0,0}$ for updating.

winner $A_{0,0}$ at location $(0,0)$ in $\mathcal{S}$ but also its 3×3 neighbors in $\mathcal{S}$, as shown in Fig. 5.

The weight for updating neuron $A(i,j)$ is computed as follows:

$$W_{i,j} = e^{-d_{i,j}}, \quad (19)$$

where $d_{i,j} = i^2 + j^2$.

This weight modifies w_2 in the CCI LCA algorithm through a multiplication, but the modified w_1 and w_2 still sum to one. This is called 3×3 update.

4.6. Sparse coding

Sparse-coding¹¹ is a result of lateral inhibition. It allows relatively few winning neurons to fire in order to disregard less relevant feature detectors. Therefore, the resulting representation is local (small response region) which in turn requires only local connections with the help of a topographic map.

In the algorithm, we only allow k neurons in each layer to have non-zero response. The output of the top i th ranked neuron has a response $z' = \frac{k-i+1}{k}z$, if $i < k$, where z is the original response. All other neurons with $i \geq k$ are suppressed to give zero responses. This is called top- k response.

5. Experimental Examples

From the perspective of developmental robotics, a robot is an agent that senses and acts. As discussed earlier, its 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, such as those presented in Ref. 42 using the earlier IHDR engine, are in order when the new engine MILN is better understood.

MILN presented above is a newer 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, as was done in Ref. 48. 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, for example, character recognition. However, handwritten characters are good for visualizing the invariance properties of the network because 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. MILN treats any input as a general sensory input frame (e.g. scenes, objects, or its own body parts) and any output as a motor output (a hand gesture, an arm movement, or a navigation command). If the input corresponds to a handwritten character, the produced motor action is a learned task-specific action depending on the input character. However, due to the generality of MILN, the input frame can be any natural visual input as shown in the last set of presented experimental data. Likewise, although the output motors in some of our following experiments represent the classes of input, the output signal should be considered as any motor action of a robot.

In the last set of experiments reported here, we also present results of feature development from natural images.

5.1. *A simple three-class example*

MILN is designed to deal with general classification problems, e.g. using natural images as input. We first look at a small sample problem using handwritten characters. The main purpose of this example is to facilitate understanding. For this example, the input images are classified into three classes, X, Y, Z. For clarity, we collected two small sets of training samples, containing 27 samples in each. The shift set and scale set mainly show variations in shifting and in scaling, respectively, as shown in Figs. 6(a) and 6(b). MILN deals with other types of variations too, as indicated in these two data sets, because MILN does not model any particular type of variation. This is a very important property of a task-nonspecific developmental program.⁴²

It is important to note that a 2-layer network with a sufficient number of neurons in both layers is logically sufficient to classify all the training samples, according to the nearest neighbor rule realized by the CCI LCA algorithm using the top-1 winner. However, this is not an interesting case. In practice, the number of neurons is much smaller than the number of sensed sensory inputs.

(a)

(b)

Fig. 6. Two sets of inputs with mainly (a) positional variation and (b) size variation.

Then, what are the major purposes of a hidden layer? With attention selection and different receptive fields in the early layers, a later hidden layer is able to recognize an object from its parts. If the earlier layers have developed invariant features, a later hidden layer is able to recognize an object from its invariant parts detection.

Therefore, in the experiment conducted here, our goal is not exactly to do perfect classification, but rather, to visualize the effects of the introduced mechanisms. Only when these effects are well understood, can we understand the pros and cons of the introduced developmental mechanisms.

In this example, we first decided the network has two layers. The numbers of neurons from layer-1 through layer-2 are $3 \times 3 = 9$ and 3, respectively. The supervision signals for the output layer are $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$, respectively. The number k is set to 3 at layer-1, so that the first layer has 3 non-zero firing neurons. The 27 samples in each set were reused 1,000 times to form a long input stream.

Figure 7 gives the developed weights of the first level neurons on these two different training sets. Specifically, Fig. 7(a) gives the results on positional variation and (b) gives the results on size variation, respectively. To develop neurons, an unsupervised method is used in layer-1 and a supervised method is used in layer-2, respectively. Furthermore, all methods use the top-1 update.

It can be observed from Fig. 7 that each neuron may average a few nearest samples (in the inner-product space), based on top-1 competition. From Fig. 7, we can see that a lobe component might average samples from different classes. In other words, the region a lobe component belongs to may contain samples from different classes. The invariance is achieved from case-based mapping: the responding lobe

Fig. 7. The weight vectors (lobe components) of the first level neurons, with unsupervised layer-1. All use the top-1 update. (a) Positional variation. (b) Size variation.

Fig. 8. The weight vectors (lobe components) of the first level neurons, with supervised layer-1. All use the top-1 update. (a) Positional variation. (b) Size variation.

component from layer-1 is mapped to the node representing the corresponding class in layer-2.

Figure 8 gives the same information as Fig. 7, except that a supervised method is also used in layer-1. We can see that with supervision, the lobe components in layer-1 are more likely to be “pure,” averaging only samples from the same class. Of course, the “purity” is not guaranteed and depends on many factors including the number of neurons. In other words, the manifold of the regions tends to cut along the boundary of output class, a concept of discriminant features. Thus, the lobe components take into account both unsupervised learning (to represent the input space well for information completeness) as well as supervised learning (to give more resource to discriminant features).

Figure 9 shows the effect of 3×3 updating. All the other settings are the same as before. First, compared to top-1 updating, 3×3 updating has a tendency to smooth across samples, as we expected. However, the near features look similar. In

Fig. 9. The weight vectors (lobe components) of the first level neurons, with 3×3 updating, using the training set that mainly shows positional variation. (a) Unsupervised learning in layer-1. (b) Supervised learning in layer-1.

this experiment, the order of the samples in the training set help to self-organize. Furthermore, it can be observed that by incorporating supervision in layer-1, the lobe components in layer-1 tends to be “purer.”

5.2. Experiments with a large data set

In this experiment, we used the MNIST database of handwritten digits available from <http://yann.lecun.com/exdb/mnist/> to show the effect of the topographic map. The data set has ten classes, from “0” to “9”. The MNIST database was collected among the Census Bureau employees and high-school students with 60,000 training samples and 10,000 testing samples. All testing samples are disjoint with all training samples. The size of images in this database is 28×28 pixels. All images have already been translation-normalized, so that each digit resides in the center of the image. Figure 10 shows some examples.

We conducted two sets of experiments for comparison, one with supervision 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 neurons to $10 \times 10 = 100$, $20 \times 20 = 400$, and $40 \times 40 = 1,600$, 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. Table 1 gives the error rates without supervision and with supervision. It can be observed from Table 1 that the corresponding error rate is lower by incorporating supervision.

When the number of clusters along the decision boundaries is sparse, the effect of supervision is relatively larger, in terms of the recognition rate. When the number of clusters in the input space is dense, the nearest neighbor should be largely sufficient

Fig. 10. Some examples of the MNIST training data.

Table 1. Error rates and entropies: Supervision helps to reduce error rates and the average entropies.

Number of Neurons in Layer-1	k	Error Rate (%)		Mean of Entropies	
		Unsupervised	Supervised	Unsupervised	With Supervision
10×10	10	20.2	18.0	0.42	0.31
20×20	20	13.3	11.2	0.30	0.28
40×40	40	7.7	7.4	0.23	0.22

to make a correct classification. Correspondingly, the relative effect of supervision is reduced when the number of neurons is increased.

For a visualization of the developed neurons, Fig. 11 shows the neurons of layer-1 trained by the training set on the $20 \times 20 = 400$ layer-1 nodes case. Some observations can be made based on Fig. 11.

First, the figure demonstrates that the lobe components represent features that run smoothly across the 2D “cortical” surface $\mathcal{S}$, and are somewhat grouped together. That is, within-class variation is naturally organized across a small neighborhood in one or a few regions in the 2D space $\mathcal{S}$ and different classes are imbedded seamlessly into this 2D “cortical” surface.

Second, this experiment shows something very important for development in the natural world: the effect of limited environment. Only sensed space is represented in the representation and the limited resource is dedicated to developmental experience. If the environment only shows a restricted set of patterns (e.g. vertical or horizontal edges, numerals, etc.), neurons developed in the earlier layers are significantly biased toward such limited environments.

(a)

(b)

Fig. 11. The weight vectors (lobe components) of the layer-1 neurons trained from handwritten digits. They represent features that run smoothly across the topographic map. Within-class variation is naturally organized across a small neighborhood. (a) Unsupervised learning in layer-1. (b) Supervised learning in layer-1.

Third, it is also demonstrated that by incorporating supervision, the neurons are statistically “purer.” That is, compared to the unsupervised version, the neuron is updated from more samples from the same class. On the other hand, for the classification task, the samples lie on the boundary with neurons weighted from the samples of different classes are “pulled” to leave the boundary and “pushed” to near to the center of one specific class of the two classes. This is similar to the rule of linear discriminant analysis (LDA), but not the same. LDA does not take into account the completeness of representation (vectors span the space of input well), but the MILN mechanism does by using unsupervised and supervised mechanism in a maturity-scheduled developmental process. To better demonstrate the difference by incorporating the supervision, Fig. 12 shows the difference of neurons by the two methods, without and with supervision in internal representation. For example, white pixels indicate weight regions that were present in the unsupervised but not in the supervised training. Please note that this only shows that the supervision does make a significant difference in developed features. For performance improvement by incorporating supervision, see the examples in Table 1 instead.

To statistically measure the “purity” of the neurons, we use the criterion of entropy. From Eq. (13), we can see that neuron $\mathbf{v}_j$ is a weighted sum of input

Fig. 12. Difference of the weight vectors (lobe components) of the level-1 neurons between unsupervised and supervised learning in the internal representation. White is positive.

samples which have been used to update this neuron j :

$$\mathbf{v}_j = \sum_{k=1}^{n(j)} w_j(k) \mathbf{y}_j(k), \quad (20)$$

where $\mathbf{y}_j(k)$ are samples used to update the neuron $\mathbf{v}_j$ and $w_j(k)$ are the corresponding update weights. To measure the purity of each lobe component, we define the empirical “probability” for samples to arise from class n as

$$p_n = \frac{\sum_m w_j(m)}{\sum_{k=1}^{n(j)} w_j(k)}, \quad m \in \text{class } n. \quad (21)$$

Thus, we have a distribution $p_0, p_1, \dots, p_9$ for each neuron. Using entropy to quantify the “purity” of this distribution for neuron j , we have

$$e_j = - \sum_{n=0}^9 p_n \log_{10}(p_n). \quad (22)$$

A neuron is relatively “pure” if its entropy is small. In particular, if $e_j = 0$, then it is updated using inputs arising from a single class.

Table 1 gives the average of entropy, averaged over all the neurons in layer-1 of six different setting, i.e. the number of neurons in layer-1 ranges from 10×10 , 20×20 , to 40×40 , for unsupervised and supervised cases. It is clear from Table 1 that supervision statistically increase the purity of neurons measured by entropy. To give more detail of the distribution of entropies among different neurons, Figure 13 shows the entropy histograms among layer-1 neurons in each of the six different network settings.

The experiment was conducted on a computer with Pentium IV 2.8GHz CPU, 1G RAM memory. The program was compiled with GCC 4.1 under openSUSE 10.2. The time needed to compute the response and update the network for each input frame is tabulated in Table 2.

The timing data showed that the networks can be computed and updated for each frame within 8 ms, or at a frequency 125 Hz using the software when the number of neurons in layer-1 is $40 \times 40 = 1,600$. The order of the time complexity is linear in the number of total neurons in the network. It is expected that the network can take inputs and generate motor actions at 30 Hz when the total number of neurons is $125/30 \approx 4$ times larger. For larger networks, special hardware is needed to maintain the 30 Hz update rate.

Table 2. Updating speed for each input sample.

Number of Neurons in Layer-1	Time (ms)	
	Unsupervised	Supervised
10×10	0.47	0.54
20×20	1.78	2.44
40×40	7.30	7.55

(a) 10×10 layout unsupervised (b) 10×10 layout with supervision (c) 20×20 layout unsupervised (d) 20×20 layout with supervision (e) 40×40 layout unsupervised (f) 40×40 layout with supervision

Fig. 13. Entropy histograms of layer-1 neurons. Neurons are “purer” with supervision. More neurons have small entropy through supervision. It could be observed that the number of neurons with entropy between 0 to 0.05 in 10×10 layout is increased from 2 to 14. Under the 20×20 and 40×40 layouts, the number is increased from 37 to 81 and 366 to 440, respectively. This shows that some “near pure” neurons become “pure” through supervision. The decrease of the mean entropy shows most neurons become “purer.”

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). Although such intermediate representation is not meant to provide only a unique neuron for each class, the supervision signal from the later layers tends to make such intermediate representation “purer.” Such purer representation is useful for learning new classes for the motor layer.

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 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. This is a general principle. A later layer should have their supervision kicked-in earlier and applied deeper than an earlier layer, because it is the later layer that delivers supervision from the motor layer.

During the process of learning the second task, the first ten 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

Table 3. The parameters for $\beta(t)$ profiles. Layer-1 starts to receive top-down signals at $t = 40,000$, later than layer-2. Layer-2 has smaller β_2 . So it receives more supervised signals than layer-1.

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 4. Error rates of multi-task learning.

Task	Error Rate (%)
Task 1	18.5
Task 2	11.3

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. A total of 2,000 samples, not including all the samples^b learned by task 1, is sufficient for task 2, because layer-1 has already taken into account a large amount of variations learned for task 1.

Table 4 shows the error rates of multi-task learning. Again, the main purpose is not to show the absolute performance, but rather the effects of multi-task learning by such a network.

Figure 14 shows a visualization of neurons in layer-1, -2 and -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:

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

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. One should not consider the response of a neuron as a response to this composite weight vector.

In layer-3, the first ten 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.

Figure 15 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 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

^bIn total, 180,000 samples by repeatedly using the MNIST digits.

Fig. 14. A visualization of neurons developed from multi-task learning. (a) The synaptic weights of all the neurons in layer-1. (b) The synaptic weights of all the neurons in layer-2. Brighter pixels represent larger weights. (c) The composite weight vectors of all the neurons in layer-2. (d) The synaptic weights of all the neurons in layer-3. (e) Broken-down weight vectors: Layer-2 composite weight vector scaled by the synaptic weight of the corresponding layer-3 neuron (i.e. before the spatial summation that gives the composite weight vector of the layer-3 neuron), to give a detailed “case by case” visualization.

13 natural images (available from <http://www.cis.hut.fi/projects/ica/imageica/>). Some of them are shown in Fig. 16. Each image patch simulates an attention fixation on the corresponding image. Each image patch is scaled to the standard size of 16×16 -pixels before being fed into MILN for development.

Fig. 15. Error rates of multi-task learning and single-task learning.

Fig. 16. Some natural images used in the experiment.

The developed feature vectors in the first layer are shown in Fig. 17. They resemble the oriented edge detectors. The smooth self-organization of the features across the 2D 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 13 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. 17(b). In other words, the results in 17(a) takes into account the size variation but those in 17(b) do not. The results in 17(a) and 17(b) are different, but statistically similar. These feature detectors could potentially be

(a) With size variation

(b) Without size variation

Fig. 17. Lobe components from natural images. (a) With size variation: select patches of random size and random position and resize them into 16×16 -pixel ones. (b) Without size variation: randomly select 16×16 -pixel image patches.

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 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

The new multi-layer in-place learning network (MILN) is a simple, biologically inspired, unified computational model with a design goal to develop a recurrent network for learning of general multiple sensorimotor tasks. At the current stage, only hand-written character recognition was tested for performance evaluation and natural images were used for a visualization of internal representations, but the inputs are treated as general visual signals in both studies. Nothing specific about characters or natural images was used in the programing. Further studies are needed to study other sensorimotor tasks. Although the Hebbian rule and lateral inhibition mechanisms are known, design of multilayer network that is truly simple and in-place is not trivial. Unlike the back-propagation learning, which uses partial derivatives as the basic mechanism for synaptic weight search, our new method uses amnesic average. In such a nonlinear adaption process, the former only uses the local behavior of gradient. In contrast, amnesic average fully utilizes all the observations along the path of nonlinear search, which results in reliable and local-minima-free nonlinear adaptation for high-dimensional input and output spaces. The supervised learning (motor imposed) is tightly integrated with unsupervised learning. Theoretically, any type of invariance features can be dealt with by the multi-layer network. A hierarchy of representations in the network is developed with increasing invariance from early to later layers. Such a hierarchy of invariant representations is shown to facilitate later learning of tasks that have shared invariance. An experiment with multitask learning indicated that incremental developmental learning can share the internally developed quasi-invariant representation across multiple tasks so that later task learning is faster than single task learning alone. The near-optimal efficiency, in-place learning, and a hierarchy of invariant representations are the three major advantages of MILN. The experimental results showed that the soft supervision reduces the average neuronal entropy, developing neurons that are “purer” and leads to lower error rates. Potentially, MILN can be used as a core technology for robots with multiple sensors and multiple effectors to incrementally develop perceptual and cognitive skills through real-time sensorimotor experiences.

Acknowledgments

This work was supported in part by the Natural Science Foundation of China under contracts 60533100 and 60402007. Weng's work was supported by a strategic partnership grant from MSU.

References

1. H. Abut (ed.), *Vector Quantization* (IEEE Press, Piscataway, NJ, 1990).
2. R. A. Andersen, C. Asanuma, G. Essick and R. M. Siegel, Cortico-cortical connections of anatomically and physiologically defined subdivisions within the inferior parietal lobule, *J. Compar. Neurol.* **296** (1990) 65–113.
3. J. J. Atick and A. N. Redlich, Towards a theory of early visual processing, *Neural Comput.* **2** (1990) 308–320.
4. J. J. Atick and A. N. Redlich, Convergent algorithm for sensory receptive field development, *Neural Comput.* **5** (1993) 45–60.
5. C. Blakemore and G. F. Cooper, Development of the brain depends on the visual environment, *Nature* **228** (October 1970) 477–478.
6. D. Boussaoud, L. C. Ungerleider and R. Desimone, Pathways for motion analysis: Cortical connections of the medial superior temporal and fundus of the superior temporal visual areas in the macaque, *J. Compar. Neurol.* **296** (1990) 462–495.
7. C. J. C. Burges, A tutorial on support vector machines for pattern recognition, *Data Min. Knowl. Disc.* **2**(2) (1998) 121–167.
8. V. Dragoi and M. Sur, Plasticity of orientation processing in adult visual cortex, in *Visual Neurosciences*, eds. L. M. Chalupa and J. S. Werner (MIT Press, Cambridge, MA, 2004), pp. 1654–1664.
9. 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 (February 1990).
10. D. J. Felleman and D. C. Van Essen, Distributed hierarchical processing in the primate cerebral cortex, *Cerebral Cortex* **1** (1991) 1–47.
11. D. Field, What is the goal of sensory coding?, *Neural Comput.* **6** (1994) 559–601.
12. K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, *Biol. Cybern.* **36** (1980) 193–202.
13. D. H. Hubel, *Eye, Brain and Vision* (Scientific American Library, distributed by Freeman, New York, 1988).
14. D. H. Hubel and T. N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, *J. Physiol.* **160**(1) (1962) 107–155.
15. M. Hubener, D. Shoham, A. Grinvald and T. Bonhoeffer, Spatial relationships among three columnar systems in cat area 17, *J. Neurosci.* **17** (1997) 9270–9284.
16. W. S. Hwang and J. Weng, Hierarchical discriminant regression, *IEEE Trans. Pattern Anal. Mach. Intell.* **22**(11) (2000) 1277–1293.
17. A. Hyvarinen, Survey on independent component analysis, *Neural Comput. Surv.* **2** (1999) 94–128.
18. A. Hyvarinen and P. Hoyer, Emergence of phase- and shift-invariant features by decomposition of natural images into independent feature subspaces, *Neural Comput.* **12** (2000) 1705–1720.
19. A. Hyvarinen, J. Karhunen and E. Oja, *Independent Component Analysis* (Wiley, New York, 2001).

20. M. Ito, G. Westheimer and C. D. Gilbert, Attention and perceptual learning modulate contextual influences on visual perception, *Neuron* **20** (1998) 1191–1197.
21. I. T. Jolliffe, *Principal Component Analysis* (Springer-Verlag, New York, 1986).
22. E. R. Kandel, J. H. Schwartz and T. M. Jessell (eds.), *Principles of Neural Science*, 4th edn. (McGraw-Hill, New York, 2000).
23. T. Kohonen, *Self-Organizing Maps*, 3rd edn. (Springer-Verlag, Berlin, 2001).
24. P. A. Kolars, R. L. Duchnick and G. Sundstroem, Size in visual processing of faces and words, *J. Exp. Psychol. Human Percept. Perform.* **11** (1985) 726–751.
25. D. Lee and S. Seung, Learning the parts of objects by non-negative matrix factorization, *Nature* **401** (1999) 788–791.
26. E. L. Lehmann, *Theory of Point Estimation* (John Wiley and Sons, Inc., New York, 1983).
27. R. Miikkulainen, J. A. Bednar, Y. Choe and J. Sirosh, *Computational Maps in the Visual Cortex* (Springer, Berlin, 2005).
28. T. A. Nazir and J. K. O'Regan, Some results on translation invariance in the human visual system *Spatial Vision* **5**(2) (1990) 81–100.
29. K. Obermayer, H. Ritter and K. J. Schulten, A neural network model for the formation of topographic maps in the CNS: Development of receptive fields in *Proc. Int. Joint Conf. Neural Networks*, San Diego, California (17–21 June 1990), pp. 423–429.
30. M. J. O'Donovan, The origin of spontaneous activity in developing networks of the vertebrate nervous systems, *Curr. Opin. Neurobiol.* **9** (1999) 94–104.
31. W. K. Purves, D. Sadava, G. H. Orians and H. C. Heller, *Life: The Science of Biology*, 7th edn. (Sinauer, Sunderland, MA, 2004).
32. W. L. Sejnowski and T. Sejnowski, Slow feature analysis: Unsupervised learning of invariances, *Neural Comput.* **14**(4) (2002) 715–770.
33. F. Sengpiel, P. Stawinski and T. Bonhoeffer, Influence of experience on orientation maps in cat visual cortex, *Nature Neurosci.* **2**(8) (1999) 727–732.
34. D. C. Somers, S. B. Nelson and M. Sur, An emergent model of orientation selectivity in cat visual cortical simple cells, *J. Neurosci.* **15**(8) (1995) 5448–5465.
35. D. C. Somers, E. V. Todorov, A. G. Siapas, D. S. Kim L. J. Toth and M. Sur, A local circuit approach to understanding integration of long-range inputs in primary visual cortex, *Cerebral Cortex* **8**(3) (1998) 204–217.
36. M. Sur and C. A. Leamey, Development and plasticity of cortical areas and networks, *Nature Rev. Neurosci.* **2** (2001) 251–262.
37. M. Sur and J. L. R. Rubenstein, Patterning and plasticity of the cerebral cortex, *Science* **310** (2005) 805–810.
38. J. B. Tenenbaum, V. de Silva and J. C. Langford, A global geometric framework for nonlinear dimensionality reduction, *Science* **290** (2000) 2319–2323.
39. P. Thompson, Margaret Thatcher: A new illusion, *Perception* **9** (1980) 483–484.
40. A. Van der Heijden, *Selective Attention in Vision* (Routledge, New York, 1992).
41. C. von der Malsburg, Self-organization of orientation-sensitive cells in the striate cortex, *Kybernetik* **15** (1973) 85–100.
42. J. Weng, Developmental robotics: Theory and experiments, *Int. J. Humanoid Robot.* **1**(2) (2004) 199–235.
43. J. Weng, N. Ahuja and T. S. Huang, Learning recognition and segmentation using the Cresceptron, *Int. J. Comput. Vision* **25**(2) (November 1997) 109–143.
44. J. Weng and W. Hwang, Online image classification using IHDR, *Int. J. Doc. Anal. Recogn.* **5**(2–3) (2002) 118–125.

45. J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur and E. Thelen, Autonomous mental development by robots and animals, *Science* **291**(5504) (2001) 599–600.
46. J. Weng and N. Zhang, In-place learning and the lobe component analysis, in *Proc. IEEE World Congress on Computational Intelligence*, Vancouver, BC, Canada (16–21 July 2006).
47. J. Weng, Y. Zhang and W. Hwang, Candid covariance-free incremental principal component analysis, *IEEE Trans. Pattern Anal. Mach. Intell.* **25**(8) (2003) 1034–1040.
48. N. Zhang and J. Weng, A developing sensory mapping for robots, in *Proc. IEEE 2nd Int. Conf. Development and Learning (ICDL 2002)*, MIT, Cambridge, MA (12–15 June 2002), pp. 13–20.

Juyang (John) Weng received his B.S. degree in 1982 from Fudan University, and M.S. and Ph.D. degrees in 1985 and 1989, respectively, from University of Illinois, Urbana-Champaign, all in Computer Science. He is currently a Professor at the Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, USA. He also holds a Visiting Professor position at the Department of Computer Science and Engineering, Fudan University, where he collaborates with

researchers. His research interests include biologically inspired neural systems, learning and development, vision, audition, touch, human-machine multimodal interfaces, and intelligent robots. He is the author or coauthor of over two hundred reviewed articles published in books, journals, conferences and workshops. He is an Editor-in-Chief of International Journal of Humanoid Robotics and the Chairman of the Governing Board of the multidisciplinary International Conferences on Development and Learning (ICDL) (<http://cogsci.ucsd.edu/triesch/icdl>). He was the Chairman of the Autonomous Mental Development Technical Committee of the IEEE Computational Intelligence Society, an Associate Editor of IEEE Transactions on Pattern Recognition and Machine Intelligence, an Associate Editor of IEEE Transactions on Image Processing, a Program Chairman of the NSF/DARPA Workshop on Development and Learning (WDL), which was held in April 2000 at Michigan State University (MSU), East Lansing, MI (<http://www.cse.msu.edu/dl>), and a Program Chairman of the International Conference on Development and Learning 2002 (ICDL'02), which was held at Massachusetts Institute of Technology, Cambridge, MA, June 2002 (<http://www.egr.msu.edu/icdl02>). He initiated and supervised the SAIL (Self-organizing Autonomous Incremental Learner) and Dav projects, in which he and his coworkers have designed and custom built their SAIL and Dav robots for research on robotic computational simulation of autonomous mental development. More details are available online at <http://www.cse.msu.edu/weng>.

Tianyu Luwang received his B.S. degree in Computer Science and Technology from Fudan University, Shanghai, China, in 2004. He is now a Ph.D. student in the Department of Computer Science and Engineering, Fudan University, Shanghai, China. His current research interest includes biologically inspired neural systems.

Hong Lu received his B.Eng. and M.Eng. degrees in Computer Science and Technology from Xidian University, Xi'an, China, in 1993 and 1998, respectively, and his Ph.D. degree from Nanyang Technological University, Singapore, in 2005. From 1993 to 2000, she was a Lecturer and Researcher with the School of Computer Science and Technology, Xidian University. From 2000 to 2003, she was a Research Student with the School of Electrical and Electronic Engineering, Nanyang Technological University. Since 2004, she has been with the Department of Computer Science and Engineering, Fudan University, Shanghai, China, where she is currently an Associate Processor. Her current research interests include image and video processing, pattern recognition, and computer vision.

Xiangyang Xue received his B.S., M.S. and Ph.D. degrees in Communication Engineering from Xidian University, Xi'an, China in 1989, 1992 and 1995, respectively. Since 1995, he has been with the Department of Computer Science and Engineering, Fudan University, Shanghai, China, where he is currently a Professor and Chairman. His research interests include multimedia information process and retrieval, image and video coding, and video streaming.