Top–Down Connections in Self-Organizing Hebbian Networks: Topographic Class Grouping

Matthew Luciw, *Member, IEEE*, and Juyang (John) Weng, *Fellow, IEEE*

***Abstract*—We investigate the effects of top–down input connections from a later layer to an earlier layer in a biologically inspired network. The incremental learning method combines optimal Hebbian learning for stable feature extraction, competitive lateral inhibition for sparse coding, and neighborhood-based self-organization for topographic map generation. The computational studies reported indicate top–down connections encourage features that reduce uncertainty at the lower layer with respect to the features in the higher layer, enable relevant information to be uncovered at the lower layer so that irrelevant information can preferentially be discarded [a necessary property for autonomous mental development (AMD)], and cause topographic class grouping. Class groups have been observed in cortex, e.g., in the fusiform face area and parahippocampal place area. This paper presents the first computational account, as far as we know, explaining these three phenomena by a single biologically inspired network. Visual recognition experiments show that top–down-enabled networks reduce error rates for limited network sizes, show class grouping, and can refine lower layer representation after new conceptual information is learned. These findings may shed light on how the brain self-organizes cortical areas, and may contribute to computational understanding of how autonomous agents might build and maintain an organized internal representation over its lifetime of experiences.**

***Index Terms*—Autonomous feature extraction, deep networks, Hebbian learning, self-organization, top–down connections.**

I. Introduction

ONE particularly challenging issue faced by a designer of a task-nonspecific learning agent is enabling the learning architecture to be general enough to learn any task possible, yet allowing refinement for a task if guidance is available from the environment. Task-specific refinement should be as nondestructive as possible. It should not prevent the agent from learning new tasks later in its lifetime or destroy the internal representation needed for tasks that were already learned.

Consider the requirement that once turned "on," an agent's software and memory cannot be directly accessed by anything besides the software itself. This "skull-closed" requirement is difficult to deal with by current methods. For a skull-closed agent operating in a rich environment, it will not be able learn the distribution of all experiences beyond some level of precision. If it has other goals besides compression, it needs to learn some parts of experience at a much higher precision. Unguided generative learning methods, which try to approximate the distribution of all experience evenly, will not be ideal. On the other hand, unchecked discriminative learning methods are typically designed to use as much resource as possible for a single task; such methods will be too destructive. Yet some form of discriminative learning is necessary.

Since humans deal with the above problem very well, we investigated biological systems. We specifically investigated visual cortex in primates, which deals with many challenging visual recognition tasks. Could cortex integrate both generative and discriminative learning in a unified way through bottom–up and top–down connections? The visual cortex is a deep hierarchical bidirectional network [1]–[3]. Environmental stimuli enters at the bottom (e.g., firing of the photoreceptors of the retina), traveling upwards towards associative cortical areas, sending information to premotor and to motor cortex, which controls movement and behavior. Cortical neurons at any place in the hierarchy generally have input connections from other neurons in three relative locations: earlier layers (bottom–up), from the same layer (lateral), and from later layers (top–down).

Do top–down connections provide abstract and task-specific information to lower layers to sculpt their learning and organization? These top–down connections are as numerous as the bottom–up connections. They are both excitatory and inhibitory in type, but about 85% of all connections are excitatory [4]. In learning, the computational roles of the top–down connections are not yet known. There are areas in cortex that have been found to represent conceptually linked information. Neurons in the inferotemporal (IT) area have been implicated in object, class, and category recognition [5]. An area in IT that represents an abstract category is the parahippocampal place area (PPA), which fires whenever a scene such as a room or landscape is seen [6]. Less abstract, but still encoding an impressive amount of variation, is the fusiform face area (FFA), which fires whenever a face is seen [7]. If they are not somehow innate, for the brain to organize these areas requires a signal representing the underlying concept by which it can "decide" if a stimuli belongs to one of these areas. Presumably, this signal comes from the top–down. Note that, despite this class-grouped organization in IT, the lowest areas in the visual hierarchy such as V1 do not have organization and features that would require task-specific guidance—it is well known that many V1 neurons represent oriented edges [8], features which many computational models have developed in an unsupervised way from images (e.g, [9] and [10]).

Manuscript received December 29, 2009; revised June 09, 2010; accepted August 11, 2010. Date of publication August 30, 2010; date of current version September 10, 2010.

M. Luciw was with Michigan State University, East Lansing, MI 48824 USA. He is now with the Dalle Molle Institute for Artificial Intelligence (IDSIA), 6928 Manno-Lugano, Switzerland (e-mail: matthew@idsia.ch).

J. Weng is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824 USA (e-mail: weng@cse.msu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAMD.2010.2072150

Inspired by the above issues and evidence, in this paper we present a biologically inspired local learning rule for self-organization and for developing the weights of a neural network layer using both bottom–up and top–down activity as bidirectional input. With class-representing top–down inputs, we show that a layer with bidirectional input develops discriminative features; additionally, neurons representing the same class become organized together, an effect we called topographic class grouping (TCG)[1].

This paper is organized as follows. Section II presents some background context. Section III introduces some preliminary ideas needed to understand the learning rule, which is then given in Section IV. Section V interprets and justifies the algorithm. Experimental results are given in Section VI. Section VII contains a discussion on related work and some broader implications. Section VIII concludes the paper.

II. Background

As proposed and refined by Barlow [11], [12], an objective of developing internal representation is to store and exploit redundancies in the observations, i.e., derive features (also called components) or groups of features as statistically independent as possible, and to organize experience into feature groups so that each group does not interfere too much with other groups. To avoid forward propagation of uncertainties in multilayer networks, it is necessary to reduce interference among the features, unless the interference is irrelevant to the "goals" of later layers. Features at lower layers having high interference with respect to some criteria unknown by that layer can make it difficult or impossible to learn for the criteria later in the network. A network that uncovers mutually dependent features makes is easy for top–down refinement to be destructive since it cannot isolate weights to change.

Many researchers developed networks for component extraction based on redundancy reduction; these mostly fall into the framework of independent components analysis (ICA) [13]. This work is built on lobe component analysis (LCA) [10], a biologically inspired competitive learning algorithm that combines biological inspiration (Hebbian learning, lateral inhibition) with candid covariance-free incremental principal component analysis [14] and ICA. But LCA does not exactly follow the ICA formulation. The data is assumed to be compactly generated as a combination of a set of hidden components (e.g., the "independent components"), but LCA does not explicitly assume independence among the components. Instead each internal "lobe component" is designed to find a concentration in the probability density, which is assumed to represent some hidden component and its distortions. By sparse coding through lateral inhibition, the lobe components compete with each other to prevent learning the same components. Each lobe component maintains the energy of the underlying distribution and updates using a quasioptimal learning rate for stability.

This work also utilizes ideas first introduced by the self-organizing maps (SOM) [15]. SOM order a layer of neurons themselves in some 2-D or 3-D space (not the data space), and define a neighborhood updating kernel on this space. In SOM, neurons compete globally with cooperation locally. If the data has some underlying low-dimensional ordering, the self-organizing process may find it and carry forward the input ordering into the internal network layers. Internal maps are often called "topographic maps." Maps reduce dimension: knowing the location of firing on a map of n neurons can give as much information with two values as could otherwise be attained using n values, in the unordered case. Maps also lead to smoothness that reflects the ordered subspace, and network generalization becomes interpretable as interpolation with respect to what the map represents. As we showed in [10], SOM is not formulated to have the stability for component extraction. We combined ideas from SOM with LCA for increased stability, to try to derive components that represent observation regularities at the same time as generating topographic organization on the internal map.

In a multilayer network, finding components with high interference with respect to later layers is typically a problem, which we deal with by using top–down connections. Most network training methods do not use the actual activations of higher layer neurons as part of the input to a lower layer, instead using the output error based on some objective function. We will show that combining self-organization with bidirectional input, and using equal bottom–up and top–down from a classification layer, leads to an emergent specialization and partitioning of class-specific resource . A group of neurons become associated with each class, and each group is connected in the neuronal map space. The TCG method groups a class into specialized features automatically, and lets the features within the groups and the number of features be adaptive based on recent experience.

The method we present combines unsupervised learning using bottom–up and top–down connections for low-interference component extraction with topographic map generation from lateral inhibition and excitation. The top–down is not globally destructive due to sparse coding. The main contributions of this paper are: 1) an efficient biologically inspired local learning rule for feature extraction and topographic map generation, that uses bottom–up and top–down inputs (and does not require output error); and 2) showing how and why class grouping emerges when top–down inputs represent different classes.

III. Preliminaries

The TCG rule is formulated for a network using rate coding and operating in discrete-time. Since we do not deal with sequences here, although we describe a dynamic system, we will mostly omit time t where it is not needed explicitly.

A. Four Types of Connections

In Fig. 1, four connection types are shown for a general network layer l: bottom–up excitation, top–down excitation, lateral excitation, and lateral inhibition, as shown in Fig. 1. The bottom–up and top–down weights are adapted through learning, but the lateral weights are dealt with by approximate methods—global inhibition by winner-take-all (or k-winners take all), and local excitation using the neighborhood kernel.

[1]When the next layer is for classification, we will also refer to the learning rule as TCG in this paper

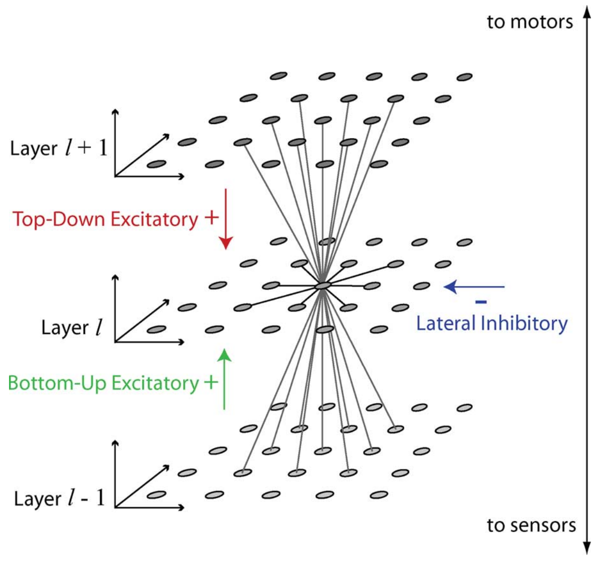

Fig. 1. A general set of three layers in a hierarchy, organized from sensors at the bottom and motors at the top. Each neuron on layer l has bottom–up and top–down excitatory projections to it, interacting with neurons on its same layer, through approximate methods of excitation and inhibition.

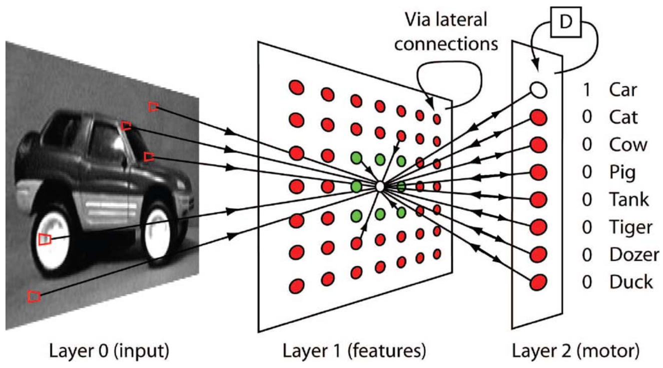

Fig. 2. A three-layer network (best viewed in color). A circle in a layer represents a neuron. The internal layer 1 takes three types of input: bottom–up input from layer 0, top–down input from layer 2, and lateral input from the neurons also in layer 1. To extend to a dynamic system, the top–down input at layer 1 at time t is from layer 2's output at time $t-1$ (represented by the "D" module). For simplicity, this is a fully connected network: every neuron in a layer takes input from every neuron in the later layer and the earlier layer. Local connectivity is possible. For the center neuron (white) in layer 1, neurons in the same layer are inhibitory (which feed inhibitory signals to the white neuron) while its nearby neurons (green) in the same layer are also excitatory. Neurons connected with inhibitory lateral connections compete so that fewer neurons in layer 1 will win, which leads to sparse neuronal updating and firing (sparse coding [16]).

We used a 3×3 pyramid kernel, which decreased interference between class groups as compared to a Gaussian.

B. Layer Input and Output

A specific three-layer network architecture is shown in Fig. 2, which has d pixels, n feature neurons and m motor (output) neurons. In a more general deep network, there are d neurons in layer $l-1$, n neurons in layer l, and m neurons in layer $l+1$. Layer l's firing rate vectors for $l-1$, l, and $l+1$ are $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$, and $\mathbf{z} \in \mathcal{Z}$.

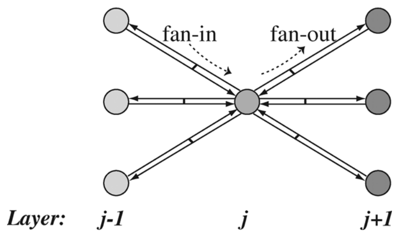

Fig. 3. Bottom–up connections to layer $j+1$ are shared with top–down connections to layer j, in a fully connected network.

Layer l's input space contains *paired vectors* from $\mathcal{X} \times \mathcal{Z}$, containing joint bottom–up and top–down input: $\mathcal{X} \times \mathcal{Z} = \{(\mathbf{x},\mathbf{z}) | \mathbf{x} \in \mathcal{X}, \mathbf{z} \in \mathcal{Z}\}$.

In a three-layer network, any input vector $\mathbf{x}$ is the raw pixel values of a digital image, while $\mathbf{z}$ represents a more abstract concept associated with $\mathbf{x}$. Here, $\mathbf{z}$ has firing equal to one at neuron(s) representing the correct label(s), where each neuron indicates a different label and zero elsewhere. It is of course not realistic to have such "conceptual stimuli," so for AMD $\mathbf{z}$ is more appropriately considered the firing at an associative layer between multiple modalities (e.g., it may represent words learned through audio processing).

C. Shared Weights

The bottom–up weights of layer l are column vectors in $d \times n$ matrix $\mathbf{V}$. The top–down weights to layer l are *shared* or *tied* with the bottom–up weights to the next layer (see Fig. 3). The bottom–up weight matrix of layer $l+1$ are $\mathbf{W}$ ($n \times m$), and then the top–down weight matrix to layer l is $\mathbf{M} = \mathbf{W}^T$. Each neuron i on layer l is represented by vectors $\mathbf{v}_i$ and $\mathbf{m}_i$.

D. Initialization

All neurons must first find the lower-dimensional subspace in which the data lies before extracting components and self-organizing in this space. To avoid a long "initial search" phase, we sequentially initialize n layer l weight vectors using a set of n layer $l-1$ stimuli: $\mathbf{v}_t = \mathbf{x}(t)$ for $t = 1, 2 \ldots n$, once layer $l-1$ neurons are distributed well, and then set the neuron "ages" (described below) to ones. For the three-layer classification networks, we sampled the initialization data from all classes. The weights could be initialized randomly, but learning will take longer, and the parameters of initial learning would have to be adjusted to ensure that all the neurons are drawn onto the data's subspace.

E. Grounding

Grounding involves filling in the internal representation between sensory observation and appropriate action [17]. The imposed action simply involved setting $\mathbf{z}$ in our experiments, but for AMD, one might consider a teacher that can get the agent to imitate an output, without understanding itself what part of the input is related to it. Training the network on both $\mathbf{x}$ and $\mathbf{z}$ allows the internal model parameters to adapt so the pair becomes more likely within the network. Later, if any components of $\mathbf{x}$

or $\mathbf{z}$ are omitted, it fills them in (without using the output error gradient).

F. Testing

For testing, there is no imposed action. The goal of learning a classification task is for the model to generalize well: a trained network provide the correct action for a given test sample drawn from the same distribution as the training data, but not specifically encountered in training. In other words, given a case that was not specifically taught, the agent should act in the correct way.

IV. Algorithm

The following is a layer l's learning algorithm. It follows the general incremental self-organizing framework: given an input, under some criterion, some neurons are considered winners. Next the winning neurons and their neighbors update their weights to do better on that (or related) criterion for the current input.

Layer-specific parameters include the number of neurons n, the sparsity parameter k, the top–down parameter β, and learning rate parameters $\theta = (t_1, t_2, \phi, \varphi)$.

Initialization involves setting weights and neuron ages (to one). After initialization for any time t, a layer does the following:

1. **Sense**. The layer samples the cooccurring bottom–up and top–down firing $\mathbf{x}$ and $\mathbf{z}$.
2. **Precompetition**. Compute competitive potential $\hat{\mathbf{y}}$ using normalized inner product for both bottom–up and top–down
$$\hat{\mathbf{y}} \leftarrow (1-\beta)\frac{\mathbf{x}}{||\mathbf{x}||}\hat{\mathbf{V}} + \beta\frac{\mathbf{z}}{||\mathbf{z}||}\hat{\mathbf{M}} \quad (1)$$
where $0 \leq \beta \leq 1$ controls the relative influence of top–down to bottom–up, and $\hat{\mathbf{V}}$ contains normalized (or zero vectors where appropriate) columns of $\mathbf{V}$ (likewise for $\hat{\mathbf{M}}$).
3. **Lateral Inhibition**. Approximate global lateral inhibition is "winner-take-all," or k-winners take all. Due to competition, most neurons will not update. Use a threshold based on the kth highest competitive potential $s(k)$: set $y_i = 0, \;$ if $\; \hat{y}_i < s(k)$. The other neurons are considered winners. To scale their responses, if i is a winner neuron, let $y_i = (\hat{y}_i - s(k+1))/(s(1) - s(k+1))$. Then normalize by L1 norm so $||\mathbf{y}||_1 = 1$.
4. **Lateral Excitation**. Approximate local lateral excitation involves spreading of response to the winner neurons' adjacent neighbors (in 3×3 neighborhoods). For each nonwinning neighbor neuron j, adjust its firing rate based on the nearest winner i's firing and the neighborhood distance: $y_j \leftarrow (y_i(1 - r_{i,j})/2)$, where $r_{i,j}$ is the distance between j and that nearest winner.
5. **Hebbian Learning**. For each neuron i with nonzero firing rate, update the bottom–up weights as in LCA
$$\mathbf{v}_i \leftarrow (1-\gamma_i)\mathbf{v}_i + \gamma_i\,\mathbf{x} \quad (2)$$
where the learning rate is automatically set from the neuron's age $\rho(a_i)$ and firing rate y_i: $\gamma_i \leftarrow (((1+\rho(a_i))/a_i)y_i)$.

The three-sectioned CCI plasticity function ρ is defined as

$$\rho(a) = \begin{cases} 0 & \text{if } a \leq t_1, \\ \frac{\phi(a-t_1)}{(t_2-t_1)} & \text{if } t_1 < a \leq t_2, \\ \phi + \frac{(a-t_2)}{\varphi} & \text{if } t_2 < a, \end{cases} \quad (3)$$

which combines optimal updating and a level of plasticity for each neuron, eventually converging to a learning rate $1/\varphi$.

Finally, increase the age of each updating neuron i: $a_i \leftarrow a_i + y_i$.

V. Justification of the Algorithm

With input from combined bottom–up and top–down space, LCA, described briefly here, tries to approximate the joint density $Pr(\mathbf{x}, \mathbf{z})$. It does so by competitive local incremental learning.

A. Fast and Stable Incremental Learning

A problem with many incremental updating methods for component extraction is the question of how to tune the learning rate. There is no way to know in general if a hand-tuned learning rate can find the components, or if it to high (in which case the weights will not "stick" to the components), or too low (in which case convergence will be artificially slow). LCA addresses this by using a framework for optimally tuning the learning rate. Optimality depends on the distribution within each partition being best approximated by the expectation of response-weighted input.

For some layer, LCA's functional description is

$$(\mathbf{y}, \mathbf{V}, \mathbf{a}) \leftarrow f_{LCA}(\mathbf{x}, \mathbf{z}|\mathbf{V}, \mathbf{M}, \mathbf{a}, k, \beta, \theta). \quad (4)$$

1) Belongingness: A limited resource of c neurons divides the sample space $\mathcal{P}$ into c mutually nonoverlapping regions, called *lobe regions*

$$\mathcal{P} = R_1 \cup R_2 \cup \cdots \cup R_c. \quad (5)$$

Each lobe region is associated with a single neuron, which represents input that falls within its region. Any input vector $\mathbf{p}$ belongs to a region R_i based on some criteria of best-match, or similarity. Depending on the sparsity, each input $\mathbf{p}$ will only belong to one or a few regions. LCA uses normalized inner product to determine belongingness

$$y_i = \frac{\mathbf{p}^T\mathbf{v}_i}{||\mathbf{v}_i||}. \quad (6)$$

2) Dual Optimality: Under our assumptions, the input is composed of a few hidden components, where each of these is distorted in the observations. If each lobe component sticks to a hidden component, we can expect a low distortion error. Globally, we wish to find set of lobe components $\mathbf{V}^*$ that minimize the expected square approximation error $\mathrm{E}||\hat{\mathbf{p}}(\mathbf{V}) - \mathbf{p}||^2$

$$V^* = (\mathbf{v}_1^*, \mathbf{v}_2^*, \ldots, \mathbf{v}_c^*) = \arg\min_V \mathrm{E}||\hat{\mathbf{p}}(\mathbf{V}) - \mathbf{p}||^2 \quad (7)$$

where $\hat{\mathbf{p}}(\mathbf{V})$ is the reconstruction of $\mathbf{p}$ as response weighted expectation over lobe components that $\mathbf{p}$ belongs to. Unfortunately, if the belongingness (partitioning) must be determined, finding a global solution to this problem is NP-hard [18]–[20].

When the partition does not change, it can be solved optimally. For a single cell, we note that, for many distributions, the sample mean leads to the lowest approximation error for the samples seen so far

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}_t = \arg\min_{\mathbf{v}} \frac{1}{T}\sum_{t=1}^{T}\|\mathbf{v}-\mathbf{p}_t\|^2 \tag{8}$$

for a set of T stimuli that fall into this cell, drawn from distribution $Pr(\mathbf{p})$. As $T \to \infty$, the following tends towards equality

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}_t \approx \arg\min_{\mathbf{v}} \mathrm{E}\|\mathbf{v}-\mathbf{p}\|^2. \tag{9}$$

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 (e.g., [21]). For such distributions, no other estimator could reach as low of an error given the observations.

The above gives the best possible estimator for any time step. Therefore, optimality is spatio–temporal since we use the most efficient estimator for any time T.

3) Optimal Hebbian Updating: We use the spatio–temporal optimality as a guide to set a neuron's learning rate. Most incremental methods do not use the above optimality. To show this, consider the Hebbian learning framework. The basic Hebbian form [22] for updating the weight vector $\mathbf{v}$ of a neuron i is

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \eta\, y(\mathbf{v}_i, \mathbf{p})\, \mathbf{p} \tag{10}$$

where each update follows the direction between weight $\mathbf{v}_i$ and input $\mathbf{p}$. The neuron's amount of updating uses the match between firing rate y_i (postsynaptic activity) and $\mathbf{p}$ (presynaptic activity). The only thing to be determined when using the Hebbian learning form is how to tune η, the learning rate.

But this typical form struggles with stability due to the single learning rate. In comparison, the LCA updating rule, derived from (8), is

$$\mathbf{v}_i(T) = (1-\eta(T))\mathbf{v}_i(T-1) + \eta_t\, y_i(T)\, \mathbf{p} \tag{11}$$

which we rewrite as

$$\mathbf{v}_i(T) = \frac{1-\eta(T)}{\eta(T)}\ [\mathbf{v}_i(T-1) - \mathbf{v}_i(T)] + y_i(T)\, \mathbf{p} \tag{12}$$

(note $1/(1-\eta) = (\eta/(1-\eta)) + 1$, where $0 \leq \eta < 1$). Equation (12) is not usable for updating, but shows a condition that is fulfilled by each update. This form shows that incremental update keeps the energy of $\mathbf{p}$ by automatically scaling $\mathbf{v}$. By the spatio–temporal optimality, there is no better way to tune the learning, and there is no way to tune a single learning rate to match (11) in one step.

Intuitively, the optimal representation of the samples in each partition is the expectation of the response-weighted input: $\mathbf{v}_i$ converges to $\mathrm{E}[y_i\mathbf{p}]$. Instead of just having the correct direction, this formulation gives both direction and distance, giving a guide to select step size.

Of course, there is no way in general that the initialization of the neurons will give a perfect partitioning. There are several ways to deal with shifting partitions. We keep the learning rate artificially high by the CCI adaptive plasticity function, allowing for "forgetting" of older observations, while still adjusting the energy appropriately.

B. Density Estimation

On the surface, the competitive learning tries to minimize distortion via a divide and conquer approach. But it also approximates the density of the input. The density estimate will be similar to that of SOM [23], [24], but using a normalized inner product to determine the partitions instead of Euclidean distance.

It implicitly defines an energy surface on $\mathbf{x}$, where energy E is based on the response of the best matching neuron

$$E(\mathbf{x}) = -\max_i \hat{y}_i. \tag{13}$$

Energy is related to the network's internal estimate of probability as

$$Pr(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{Z_{\mathbf{x}}} \tag{14}$$

where Z handles normalization and is typically intractable to compute. Using normalized inner-product, each neuron defines a Laplacian kernel, which is sharper than a Gaussian and has large tails. Within each partition, each kernel is placed with mean as the weighted centroid of this neuron and its neighbors, weighted by neighborhood distance [24].

By training, we want to decrease the energy of the current $(\mathbf{x}, \mathbf{z})$ pair, but we would like not to do this in a destructive way (i.e., increasing the energy of patterns we have previously trained as a side effect).

For understanding, consider training pair $(\mathbf{x}, \mathbf{z})$ for a simpler three-layer network. In multilayer networks, layer l simply treats the output of the layer $l-1$ as its input. For this net, we are concerned with the network's internal joint probability $Pr(\mathbf{x},\mathbf{y},\mathbf{z}) = Pr(\mathbf{z}|\mathbf{y})\ Pr(\mathbf{y}|\mathbf{x})\ Pr(\mathbf{x})$. Since we do not use probabilistic assignment, $\mathbf{y}$ is deterministic from $\mathbf{x}$, so we can ignore $Pr(\mathbf{y}|\mathbf{x})$.

After evaluating, we get

$$Pr(\mathbf{x},\mathbf{z}) = \frac{e^{-[E(\mathbf{z}|\mathbf{y})+E(\mathbf{x})]}}{Z_{\mathbf{x},\mathbf{z}}} \tag{15}$$

which gives the energy of the pair as

$$E(\mathbf{x},\mathbf{z}) = E(\mathbf{z}|\mathbf{y}) + E(\mathbf{x}). \tag{16}$$

Equation (16) suggests that to decrease the energy given an input pair, neurons on the feature layer should not be considered a winner based only on how well they represent the input, but also on how well they represent the output at the next layer (in other words, how much uncertainty about the true output they provide to the output neurons).

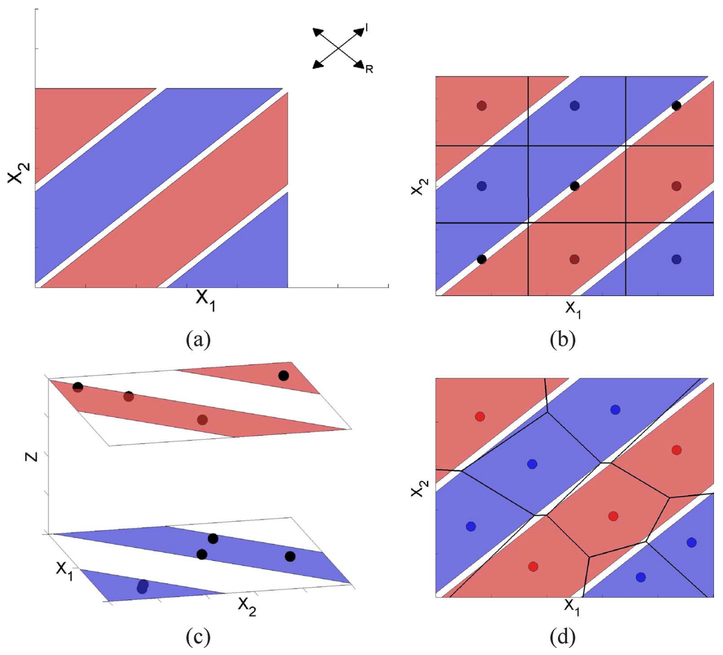

Fig. 4. Self-organization in a weighted semantic-physical space leads to partitions that separate the classes better than when self-organizing in the purely physical space (best viewed in color). (a) There are two bottom–up dimensions x_1 and x_2. Samples falling in the blue area are from one class and those falling in the red area are another class (assume uniform densities). The "relevant" and "irrelevant" dimension are shown by the upper right axes, which are here linear (diagonal). (b) The effect of self-organization using nine neurons in the bottom–up space. Observe from the resulting partitions that the firing of many neurons will transmit ambiguity about the class. (c) Boosting the data with top–down information, which here is shown as a single extra dimension instead of two (for visualization) (d) After self-organizing in the boosted space and embedding into the bottom–up two dimensions. Note how the partition boundaries now line up with the class boundaries and how the data that falls into a given partition is mostly from the same class.

If we do not take this into account, since the energy of $\mathbf{z}$ depends on $\mathbf{y}$, updating the same feature neuron $\mathbf{v}_i$ over a distribution of inputs containing instances of two or more classes leads to output uncertainty. For the network to reduce a single input pair's energy through an internal class-mixed neuron, it will have the side effect of interfering with (increasing energy for) already trained pairs from the other class.

One way around this problem is for the designer to set aside certain amount of resource in the feature layer for each class. But in general we cannot know how much is needed by each class, and this reduces adaptivity (i.e., the current task cannot recruit resource needed). Instead, we use top–down connections to redefine a belongingness that takes into account (16), so that the network can partition the resource itself, with as little updating interference as possible.

C. Top–Down Connections

Much of the information experienced in a real data stream is *irrelevant*. It is crucial for a limited size network to find the relevant information, so the irrelevant information can be discarded if necessary. In supervised learning, we assume that the relevant part of the input matches the expected input over all instances of the same imposed action. For example, if the agent sees the shape of the capital letter "A" in multiple contexts (on a flashcard, on television, on a sign, etc.) and a teacher can cause the agent to focus on the letter and "speak A" in each case, then the relevant information (the shape of the letter and the action performed) will not change too much over the different inputs, but other information (the surrounding visual scene) changes a lot.

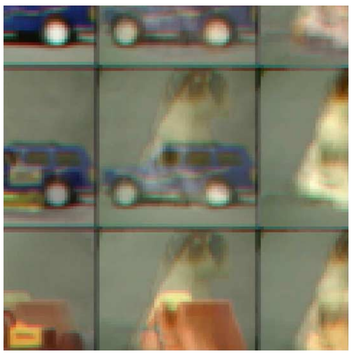

Fig. 5. A layer-one weight vector, around other neighbor weight vectors, viewed as images, of a neuron exhibiting multiclass distortion due to 3×3 updating.

The visual "essence" of the action "speak A" is uncovered as the expectation over the visual stimuli linked to that action.

Let the layer's bottom–up input space $\mathcal{X}$ be made up of relevant subspace $\mathcal{R}$ and irrelevant subspace $\mathcal{I}$. The relevant subspace is correlated with output space $\mathcal{Z}$ and can be uncovered from $\mathcal{X} \times \mathcal{Z}$

$$\mathcal{X} \times \mathcal{Z} = (\mathcal{I} \times \mathcal{R}) \times \mathcal{Z} = \mathcal{I} \times (\mathcal{R} \times \mathcal{Z}).$$

We use a scaled version of $\mathcal{X} \times \mathcal{Z}$, called $\mathcal{P}$. To achieve neuron specificity, it is essential that the scale of top–down influence can match bottom–up. When a layer uses paired input connections, the influence of one of $\mathbf{x}$ or $\mathbf{z}$ could be very low. For example, the dimension of visual dimensions is large (e.g., a 40

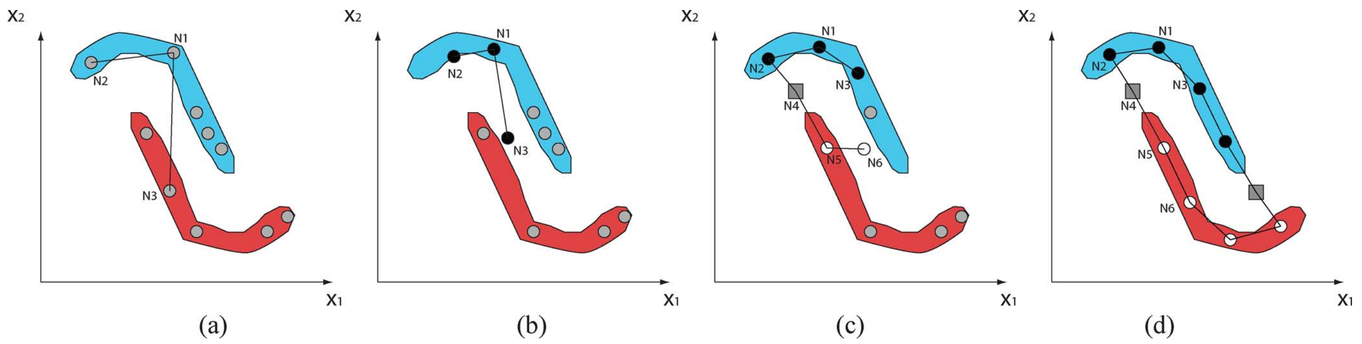

Fig. 6. Topographic class grouping with a 1-D neuron array in 2-D input space (best viewed in color). The red area contains samples from class one, and the blue area contains samples from class two. The 10 neurons' bottom–up vector are circles or squares. Their top–down membership is shown by the color or shape: Gray neurons are unassociated, black neurons are exclusive resource for class two and white neurons exclusive for class one. Square neurons are border neurons. To better understand how TCG emerges, we provide the following four cases. (a) After initialization, all neurons are unassociated. Only three neurons are drawn to show they are neighbors. Other neighbor connections are not shown yet, for clarity. (b) Neuron N1 has won for a nearby sample, becomes linked to class two. Its neighbors are pulled towards it and also link to class two. Note how N3 is actually pulled into the between-class "chasm," and not onto the class distribution. (c) Over wins by N1, N2, and N5, self-organization occurred through neighbor pulling. N4 has become a border neuron, and N6 is starting to be pulled. (d) A final organization, uncovering the relevant dimension. For this data, the relevant dimension is not linear: it is through the center area of each class distribution, lengthwise. The final neuron organization mirrors this, and the neurons have organized along the relevant dimension.

row and 40 column digital image gives 1600 dimensions) compared to the number of label components (e.g., 10 classes). So that the relative influence could be easily controlled, we mapped each $(\mathbf{x}, \mathbf{z})$ to a paired vector $\mathbf{p} \in \mathcal{P}$

$$\mathbf{p} \leftarrow \left(\sqrt{(1-\beta)} \frac{\mathbf{x}}{||\mathbf{x}||}, \sqrt{\beta} \frac{\mathbf{z}}{||\mathbf{z}||} \right). \tag{17}$$

Setting $\beta = 0.5$ gives the bottom–up and top–down equal influence. Raising β will increase the distance between different class distributions in $\mathcal{P}$.

For classification[2], the components of $\mathbf{z}$ represent sets of mutually exclusive events. Each imposed action vector will be codirectional with an axis of $\mathcal{Z}$, and $Pr(\mathbf{z})$ is a mixture of delta functions at different positive axes. In this case, if no samples are members of two classes, any two class distributions are linearly separable in $\mathcal{P}$.

The top–down boosting embeds the samples into a space where samples from different classes are farther apart and always separated. This reduces and can nearly eliminate updating interference between classes. During self-organization, two effects emerge. Partition boundaries will tend to follow the class boundaries to the relevant subspace, and the neuronal entropy becomes lower. An illustration of these effects for a simple data distribution is shown in Fig. 4.

D. Self-Organization

Since $Pr(\mathbf{z})$ is a mixture of delta functions, purely competitive learning using a hard-assignment winner-take-all causes a "hard" separation of per-class resource. We wish to use a softer resource assignment, and include cooperative learning as neighborhood updating, approximating lateral excitatory connections that are more dense closer to the neuron of origin. We use 3×3 updating, meaning the winner neuron will update its weights and so will the neurons adjacent to it.

It is generally not beneficial for neighborhood pulling to be between class distributions. Smoothness is useful for generalization, but it can be harmful if the distribution $Pr(\mathbf{x})$ is multimodal where different peaks are separated by very low-probability areas. Consider a neuron with two neighbors that fire often. This neuron is pulled in $\mathcal{P}$ by both of its neighbors, placing it in between what the neighbor neurons represent. This could be useful if they represent a single class, as it would average into a variation that might generalize well for that class. But if they represent different classes, the averaging effect may lead to a representation of something that would be very unlikely (see Fig. 5).

Using LCA and paired input $\mathbf{p}$, neurons are attracted to high-probability areas in $\mathcal{P}$. Via competition and updating, different neurons seek out different areas with a concentration of probability density. For any input, the best matching neuron represents that input better than the other neurons. The problem is any neighbor neuron that does not match the input well will still update its weight vector if it is adjacent to firing neuron i. The neighbor neuron j's weight is pulled away from its spot in $\mathcal{P}$ and travels linearly between its weight and the input at $(1-\gamma_j)\, \mathbf{v}_j + \gamma_j\, \mathbf{x}$. γ is the percent of the distance it travels towards $\mathbf{x}$. Since this updating has a distance sensitivity, if $\mathbf{x}$ is very far away from $\mathbf{v}_j$, it can change a lot even with a low learning rate. It becomes more likely that $\mathbf{v}_j$ ends up in a very low probability area, which hurts the approximation of $Pr(\mathbf{x})$. Now, since $\mathbf{v}_j$ will have little chance to win in competition, it may not recover.

The top–down connections, and a small neighborhood kernel, help address this issue. The neuronal precompetitive response function

$$\begin{aligned} \hat{y}_i &\leftarrow (1-\beta)\, \hat{y}_{x,i} + \beta\, \hat{y}_{z,i} \\ \hat{y}_{x,i} &\leftarrow \frac{(\mathbf{x}^T \mathbf{v}_i)}{(||\mathbf{x}||\, ||\mathbf{v}_i||)} \\ \hat{y}_{z,i} &\leftarrow \frac{(\mathbf{z}^T \mathbf{m}_i)}{(||\mathbf{z}||\, ||\mathbf{m}_i||)} \end{aligned} \tag{18}$$

shows that by setting β high, it is essential that a neuron find become class-specific in $\mathcal{Z}$ in order to have a chance to win. Normalization encourages neurons to become representative of

[2]Some related results concerning regression are in [25].

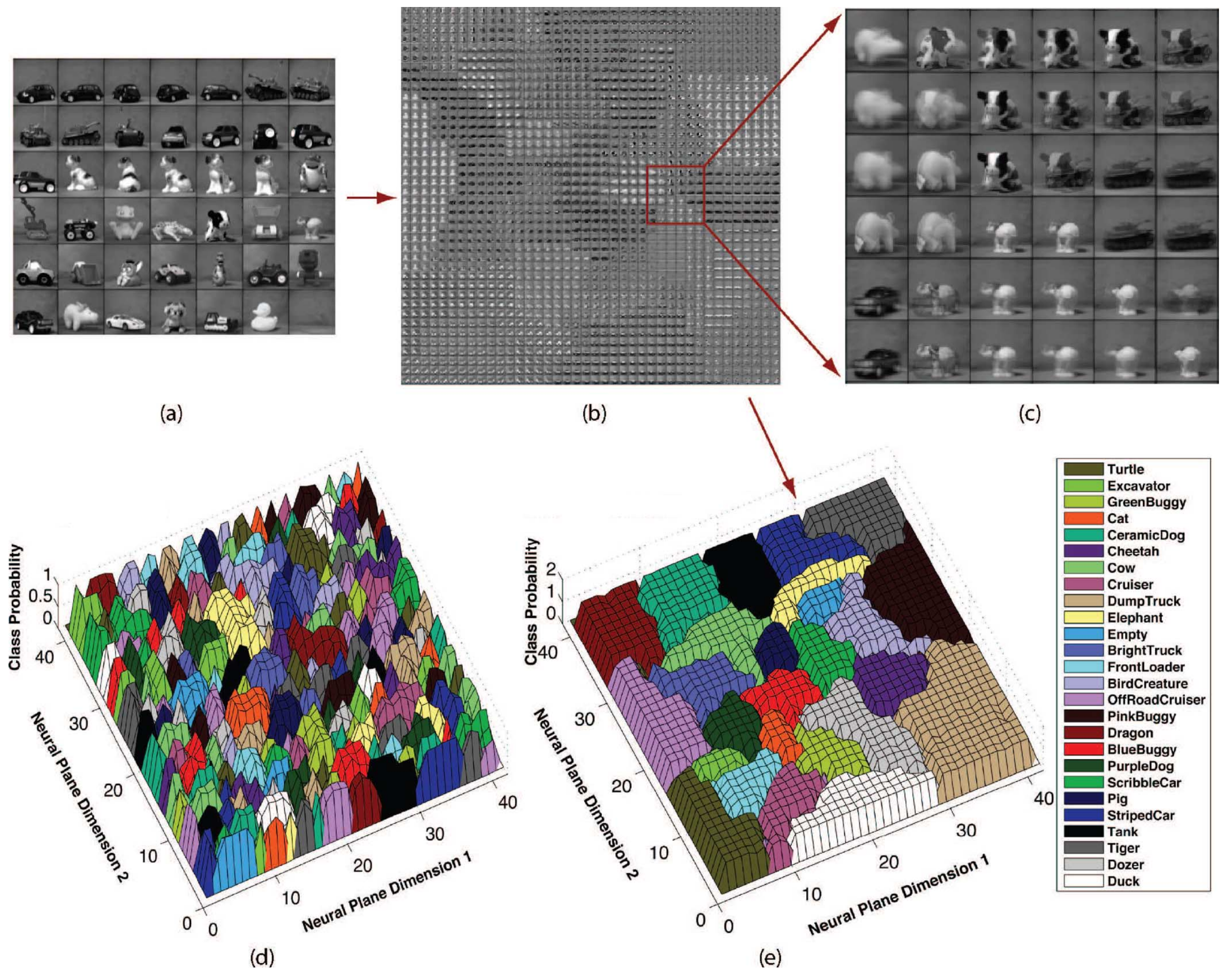

Fig. 7. Summary of experiments for 25 objects viewed from a full range of horizontal 360 degrees (best viewed in color). (a) Some example views of the 25 objects. (b) The bottom–up weights of 40×40 neurons in layer one developed using top–down connections. (c) The expended views of the bottom–up weights of 6×6 neurons. Each small square image corresponds to the weight vector of a neuron. (d) The maximum class representation probabilities of the 40×40 neurons of layer one in a network developed without top–down connections. (e) Developed using top–down connections.

only a single class. Consider such a neuron i as *exclusive* resource for class z_j.

Consider the exclusive neurons as *active* neurons and the class-mixed neurons as *passive* neurons. Additionally, the active neurons are class-specific. The active neurons gradually recruit neighbor passive neurons. Being gradually pulled in $\mathbf{V}$ causes the passive neurons to move towards areas in $\mathcal{X}$ and $\mathcal{Z}$ with higher probabilities. If a passive neuron is pulled by an active neuron enough, the passive neuron finds a high-density area. It can start to compete with the active neurons and becomes an active neuron itself, in turn recruiting its neighbors. In this way, grouping emerges.

Fig. 6 illustrates an incremental grouping and growing with two classes lying in 2-D and using a 1-D circular neuronal array. Such wraparound organizations reduce border effects.

As a final note, even when using top–down, 3×3 could be harmful if the class distributions themselves are disconnected or have holes. In that case, the "harmful pulling" described above could occur even within a class. It may be better and more biologically plausible to do away with the isotropic updating and use instead an adaptive lateral excitation (some preliminary work on this was done in [26]).

VI. Experimental Results

We propose that top–down connections from an abstract layer to a more physical self-organizing feature layer lead to grouped class areas and lower entropies on the physical layer, and higher recognition rates in classification overall. To test this, we compared two types of three-layer networks. In the first type, the feature layer learned by both bottom–up and top–down connections ($\beta > 0$). The second network type only utilized bottom–up connections, and the top–down were disabled ($\beta = 0$). Top–down connections were disabled in the testing phase for all networks tested. To classify a testing sample, the network operates in feed-forward mode, and the maximum firing rate at the motor layer represents the output class label. The networks do not take advantage of temporal dependencies in the inputs.

For CCI Plasticity, we used settings $t_1 = 20$, $t_2 = 200$, $\phi = 2$, $\varphi = 2000$ throughout.

Proof of Concept

We selected 25 toy objects to train different sized networks to recognize each individual object. 200 images of 56×56 were taken in sequence for each object. There is also an "empty" (no

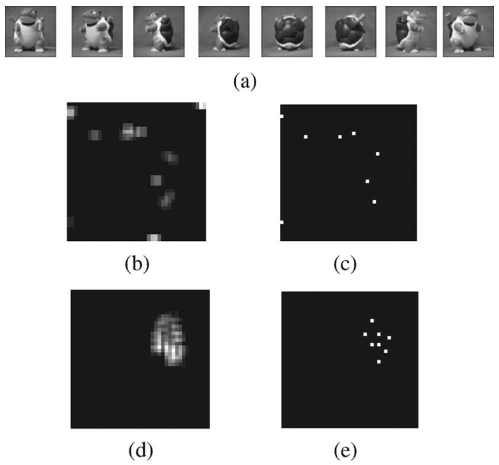

Fig. 8. (a). Images presented to a trained network to measure the class-response scatter. (b) Bottom–up weight to the neuron representing this class ("turtle"). This network was top–down-disabled. (c) Top responding neuron positions for each of these samples for the unsupervised network (d) Layer-two weight for a top–down-enabled network (d) Top responding neuron positions for the top–down network.

TABLE I
ERROR RESULTS FOR THE 25 OBJECTS, AVERAGED OVER FIVE TRIALS

Neural plane size	Avg. error (FF)	Avg. error (FF+TD)	Error reduction
20×20	8.13%	3.03%	**62.7%**
30×30	2.62%	0.83%	**68.3%**
40×40	0.63%	0.33%	**47.6%**

object) class. The 200 images cover about two complete rotations of 360 degrees for each object. An object varies slightly in position and size throughout its sequence. The background color is controlled to not be an issue (more details in [27]).

1) Experiment 1: Top–Down With a Limited Resource: Five top–down-enabled and five top–down-disabled networks were trained for each of the network sizes $n = 20 \times 20$, 30×30, and 40×40[3]. This is a limited resource utilization problem, as the networks cannot just store all the views. Additionally, the networks do not use time so they cannot take advantage of temporal dependencies in the rotation. Given the limited resource, the class and view angle could ideally be represented so that each neuron is responsible for a single class over a set of angles from about 5°(for $n = 1600$) to 25°(for $n = 400$).

Two types of the networks were trained: the first type used excitatory top–down connections ($\beta = 0.3$), while the second type did not ($\beta = 0$). Every fifth image in each sequence was set aside for testing. Training involved random sample selection over 50 000 training samples, using $k = 1$. All the -down enabled networks showed TCG, as seen in Fig. 7. For reporting purposes, TCG was measured by the within-class scatter of neuron responses on the 2-D neuronal plane. The within class scatter of firing for each stimulus class, averaged over all stimulus classes, measures how condensed the neuron responses were. The class-response scatter is the trace of the within class scatter matrix, normalized for map size: $\omega = \sqrt{\mathrm{Tr}(\mathbf{S}_W)}/\sqrt{n}$. Using $\mathrm{Tr}(\mathbf{A}) = \mathrm{Tr}(\mathbf{BAB^{-1}})$, we can

[3]For the larger (40×40) networks, we scaled up the feature layer in resolution from 20×20 networks after 1500 samples.

TABLE II
FEATURE ENTROPY AND GROUPING RESULTS FOR THE EXPERIMENTS WITH 25 OBJECTS

Top-down	Neural grid size	Developmental entropy [0-1]	Class-response scatter [0-1]
Disabled	20×20	0.23	0.35
Enabled		**0.19**	**0.13**
Disabled	30×30	0.22	0.44
Enabled		**0.15**	**0.11**
Disabled	40×40	0.20	0.44
Enabled		**0.08**	**0.10**

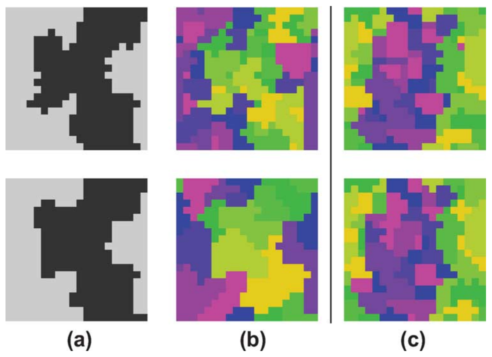

Fig. 9. Category and class sensitivities of neurons in two different 20×20 networks after each stage of two-stage teaching (best viewed in color). The color of each pixel of these 20×20 images indicates either category selectivity (a) or class selectivity (b) and (c). The five classes from one category are given purple shades while the other five classes are given green shades. Selectivity after stage one (top row) and two (bottom row) are shown. (a) and (b) correspond to the network trained on category first, then both category and class. Observe in the lower part of (b) how the class motors have organized the lower-layer representation within each category. (c) shows the control network where only category was taught.

see this measure is invariant to rotation of the 2-D map (see Fig. 8). We also measured developmental entropy, which tells us how class-specific the neurons' updating was.

For testing, we tried different values of k from 1 to 10 for testing and reported the best results. The networks using top–down connections did best with $k = 1$, but the others did best with k close to 4 or 5. The test error rate results are presented in Table I (FF stands for feedforward and TD stands for top–down) and the measured developmental entropy and scatter is presented in Table II. The per-neuron entropy is lower, as is the class-response scatter, in the top–down networks. These results show the top–down enabled networks develop to utilize the same amount of available resource better, shown by better error rate. Especially notable differences are seen when the number of neurons is smaller. In particular, a top–down enabled network with $n = 400$ gets about the same performance as a feedforward network with $n = 900$. A network has $nd + nc$ learnable weights. As an example, for $n = 400$, this equals 1.3×10^6, compared to 2.8×10^6 when $n = 900$. The smaller network using top–down attains 99.6% of the performance of the larger one without top–down, while using just 44% of the resource.

2) Experiment 2: Two Task Learning With a Shared Resource: We guess top–down is necessary to refine existing representation after new concepts are learned. The purpose of this experiment is to illustrate resource management when a network learns two tasks—categorization and classification—at

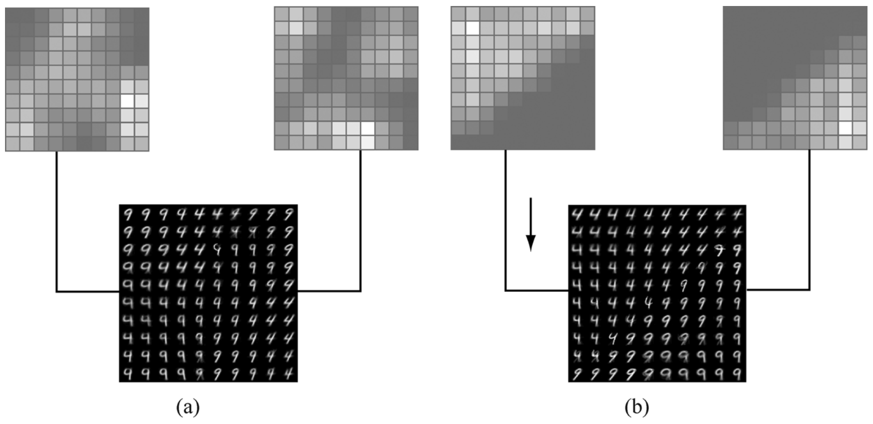

Fig. 10. The handwritten digits "4" and "9" from the MNIST digit database [29] are very similar physically. (a) Result after self-organizing using no motor-boosting. The 100 weight vectors are viewed as images below and the two weight vectors for each motor shown above. White means a stronger weight. The organization is class-mixed. (b) After self-organization using motor-boosted distance (weight of 0.3). Each class is individually grouped in the feature layer, and the averaging of each feature will be within the same class.

TABLE III
RESULTS FOR INCREMENTAL TWO-TASK LEARNING

Tested After	Category Error	Class Error	Category Specific	Class Specific
S1	2.0%	—	92.5%	60.7%
S2	0.06%	0%	92.8%	80.5%
S2-C	0.04%	—	93.0%	65.5%

two different times, but for the same stimuli. We trained the more general categorization task first, followed by the more precise classification task. Many discriminative methods trained on a single task would throw out information that turns out to be useful for learning the second task later.

We selected ten classes from two categories — toy vehicles ("Cruiser," "Off-road cruiser," "Pink buggy," "Blue buggy," and "Green buggy") and toy animals ("Cow," "Tiger," "Duck," "Pig," and "Elephant"). We trained 20×20 networks in two stages. The first stage used 5000 randomly drawn samples (from time $t = 1$ to 5000) and the second stage took 15 000 samples (from $t = 5001$ to 20 000). The networks have 12 motor neurons — 10 for classes and 2 for categories. When a network is trying to learn or produce both category and class, the sparsity parameter k_2 for the motor layer was set to two. Otherwise, k_2 was set to one. The top–down parameter β was set to 0.5. We also used edge-wraparound neighbor relationships.

Test 2.1: Adding Information: In the first stage (S1), the agent is taught to produce the category name when seeing an image. In the network, the appropriate category motor is imposed and the other clamped to zero. In one second stage (S2), the agent hears the teacher say both the category name and the class name, and it repeats both of these. As a control, we tried an alternate stage two (S2-C) where the stage one teaching of category continued.

Results are presented in Table III. Category error was not affected based on whether the classification task was trained or not. To measure how top–down class-based input influenced the first layer after stage 1, we measured the maximum category and class representation probability for the layer 1 neurons. The second two columns show their average, measuring how selective the layer 1 neuron's are for a single category or class. After training with class, layer 1 neurons on average jump up nearly 20% in class-selectivity, whereas for the control condition they on average only increase by about 5% ($p < 0.0001$). We can observe how the top–down class connections influence neuron organization, shown in Fig. 9. After stage 1, there are two large groups for the two categories, but class representations are mixed inside each category group. But after stage 2, subgrouping within each category group emerged.

Test 2.2: Two Stage Learning: In a new second stage the agent is taught class, but *not* category. This test shows the effects of destructive interference [28] (from learning the new task) since both class and category use the same layer 1 representations. Destructive interference has a severe effect, as the categorization error at the final $t = 20\,000$ dropped to 48% on average — nearly chance. The classification error is 0. Such task unlearning is due to the growing and spreading of groups in stage two, while the category weights do not change. Since class and category use the same layer one representations, this interference gets around long term memory by sparse coding.

What can be done to avoid complete task unlearning? We can maintain the shared resource by operating the network in a *semi-supervised* mode, where the network layer l's output causes a firing rate vector at layer $l+1$, and that firing vector is used as top–down input to layer l for learning. If an agent has learned a task well that applies to its current experience, it can practice or rehearse performing that task, and it can maintain a level of performance. For this experiment, it "rehearsed" category while being taught class. After learning over this semi-supervised stage, networks classified with zero error, and did not show much destructive interference: they categorized with average 3.4% error ($p < 0.001$).

3) Experiment 3: Grouping With Physically Similar Classes: As a small test of TCG, we observed that grouping will result even for very similar bottom–up inputs, such as the handwritten digits "4" and "9." This can be seen in Fig. 10.

A. Comparison Datasets

1) MNIST Digits: The well-known MNIST dataset of 70 000 total images (60 000 training, 10 000 testing) contains 10 classes

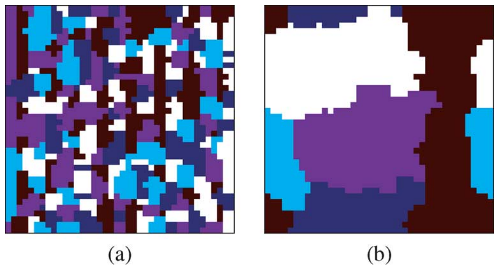

Fig. 11. 2-D class maps for a 40×40 neural grid after training with the NORB data. At each neuron position, a color indicates the largest outgoing weight in terms of class output. There are five classes, so there are five neurons, and five colors. (a) $\beta = 0$ (b) $\beta = 0.3$.

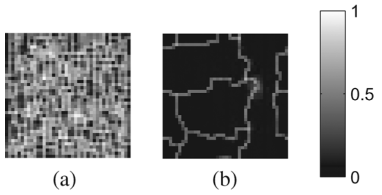

Fig. 12. 2-D entropy maps for the Fig. 11 experiments. High-entropy means neurons represent multiple classes, which can lead to error. Whiter color means a higher entropy. (a) $\beta = 0$ (b) $\beta = 0.3$. Note the high entropy neurons shown here coincide with class group borders shown in Fig. 11.

of handwritten digits (from 0 to 9). Each image is composed of 28×28 pixels. The foreground, digit pixels are nonzero. Each digit resides in the center of its image. We did not use any data prepreprocessing.

The MNIST dataset is useful since we can compare with the many other methods that have tried this data. We trained a three-layer network with 100×100 map with $\beta = 0.3$, and $k = 1$, which reached 2.97% on the test set. Performance of a few other three layer networks without preprocessing[4] are as follows. Contrastive Divergence [30] reached 2.49%. It was fine-tuned using supervised gradient descent, after which it reached 1.25%. It would be interesting to fine-tune our method in such a way. K-nearest neighbors (storing all 60 000 training samples) with L2 distance reached 3.09% and 2.83% for L3 distance. Support vector machines with a Gaussian kernel reached 1.4% with no preprocessing. Deep networks with more than three layers, which were also trained with additional distortions of the data, have reached the best performances on MNIST.

We note here some caveats about the performance of TCG in three-layer networks. The main purposes of this approach are layer-wise feature extraction, developing an internal ordering through self-organization, and resource management over time for AMD. It sacrifices the possibility of learning complex features on the first hidden layer by enforcing a strict sparse coding, so that the features might reduce redundancy well. Since it does not globally follow a gradient to decrease the output error function, or even the reconstruction error function, it is not expected to directly compete with those that learn by a reconstruction or error gradient. And it certainly introduces some distortions in order to generate topography.

[4]Summarized on the MNIST web page.

TABLE IV
PERFORMANCE ON THE NORMALIZED-CENTERED NORB DATA WITH AND WITHOUT TOP–DOWN

Neural plane size	Error (FF)	Error (FF+TD)
40×40	26.5%	17.9%
60×60	26.2%	15.7%
80×80	–	13.8%
100×100	–	12.6%

TABLE V
AVERAGE ENTROPY ON THE NORB FIVE-CATEGORY DATASET

Top down	Neural plane size	Entropy
N		0.5
Y	40×40	**0.07**
N		0.49
Y	60×60	**0.09**

2) NORB Objects: The normalized-centered NORB dataset [31] is a 3-D object categorization dataset[5]. It contains stereo image pairs of five categories of objects (four legged animal, human figure, airplane, truck, car), with 10 different actual objects belonging to each class. The 5 training objects per class and 5 testing objects per class are disjoint. The dataset has 24 300 training images, and 24 300 testing images, with uniform background. The dimension of each training sample is $96 \times 96 \times 2 = 18\,432$. The objects vary in terms of rotation (0 to 340° in 18° increments), elevation (30 to 70° in 5° increments), and lighting (6 different conditions). Recognition must be done based on shape, since all objects have roughly the same texture and color.

The NORB categories have much more variation than the object classes used in the 25 objects' case. The test set is very different from the training set, so it requires powerful generalization. Overfitting on the NORB training data leads to a much worse performance than MNIST.

We compared top–down-disabled and top–down-enabled networks of sizes 40×40 and 60×60 ($k = 1$), and top–down enabled networks only for sizes 80×80 ($k = 3$) and 100×100 ($k = 7$), over five epochs over all the training data. Like in MNIST, we did not use any added deformed patterns. The networks scaled up in size from $\sqrt{n}/4 \times \sqrt{n}/4$ for 8000 samples to $\sqrt{n}/2 \times \sqrt{n}/2$ for 15 000 samples to $\sqrt{n} \times \sqrt{n}$. We used wraparound neighborhoods. Results are presented in Table IV, showing results at the different sizes (average for the two smaller sizes, and best for the larger sizes), and Table V, which shows entropy. Figs. 11 and 12 show that TCG emerged.

Since K-nearest neighbor achieves 18.4% error rate on this data [31], it seems the NORB data is susceptible to overfitting, implying the hidden components that describe the different categories cannot be extracted very well based on the full images. The smallest network we tried, having only 1600 neurons even achieved a slightly better 17.9% performance, achieving this with only 6.7% of the resource (but taking 4 more passes through the data). The generalization provided by top–down connections and resource separation by top–down grouping is beneficial (note the 40×40 network without top–down was much worse than K-NN) and actually gives a better result than

[5]http://cs.nyu.edu/ ylclab/data/norb-v1.0/

nearest neighbor, despite operating in winner-take-all with the full images.

The best performance for a three-layer network was 12.6% for a 100×100 network trained with $k = 7$. As summarized in [32], other results from similar algorithms on this NORB dataset include logistic regression (19.6%), SVM with a Gaussian kernel (11.6%), and deep belief networks with a greedily pretrained sparse layer-one and label units on layer-two, trained entirely through contrastive divergence (11.9%).

The convolutional networks, which use attention (deriving small localized codes), max pooling, and use the error gradient perform the best on the NORB data. We did not experiment with smaller receptive fields on this data yet.

The result of 12.6% for a three-layer network is very far off the results of the best methods. But this result is in the same performance class as the other three-layer models. The NORB data appears to be described well by a more hierarchical and local code, which is also how cortex represents objects. For better performance with a reasonable number of neurons, we will have to extend the network to use limited size receptive fields and more layers. The algorithm in Section IV can be used to train deeper networks in a layer by layer fashion. The main focus of this paper is on the properties of biased self-organization due to top–down connections, so training different types of networks on NORB using the TCG method is a topic of future work.

VII. Discussion

A. Extendability

This paper establishes the fundamental ideas behind the TCG learning rule. To become competitive in learning complex recognition tasks on its own, some extensions are probably needed.

We known that adaptive lateral connections have a positive effect [26]. Actually, even without adaptation, simply including the influence of lateral excitation in the response function leads to an internally consistent learning rule that can be shown to follow the global gradient of minimal distortion [23]. The minimum distortion learning rule might lead to better performance.

Due to difficulties in using backpropagation to train deep networks, the layer-wise unsupervised pretraining method [33], followed by fine-tuning via the error gradient, has become popular. Typically, each layer is trained to minimize reconstruction error, or contrastive divergence error [34]. An issue with layer-wise training is that it is generally unknown what the best features are to extract at a lower layer that will lead to the best performance at the output layer, so it is usually not known how well a deep network has done feature extraction. As our method is a divide and conquer algorithm for "redundancy exploitation," which is interpretable in a divide and conquer way, it may be useful to use this or a similar method in layer-wise training.

Salient future issues involve extending the networks to use local analysis and more layers. In [35], Bengio and LeCun argue that many functions can be represented compactly by deep architecture, and most functions that can be represented compactly by deep architectures cannot be represented compactly by shallow architectures. This includes, e.g., object recognition with complex and widely varying backgrounds, which requires attention. Generalization in shallow networks involves local smoothness and is limited, but generalization in a deeper architecture can be combinatorial and potentially much more useful. As discussed there, SVMs are ill-suited to extend in AI, as they are shallow networks, they struggle when the number of samples increases, do not deal with combinatorial data efficiently, and cannot utilize attention.

Attention is a very important aspect of future work. For this particular work, we handle the components in a permutation invariant way [30]. In cluttered and uncontrolled scenes, this monolithic image analysis is not effective due to the complexity of the background and possible occlusion. Monolithic representations with complex backgrounds require a lot more resource (practically unattainable) to represent all the variations. In general, local analysis will be necessary. We have integrated some TCG networks into some preliminary work on deeper architectures that use local analysis [36], [37].

How appropriate is the learning network we described to extend towards AMD? An algorithm for AMD must be low complexity, fast in training and testing, and be adaptable without being destructive. The core issues are allowing the agent to learn new tasks (possibly unpredicted by the designers) quickly at different times in the agent's life, and integrating what is newly learned with what has already been learned. Of course not all issues have been solved, but the TCG algorithm fits well into a framework for AMD, due to its simplicity, biological plausibility, generality, plasticity, and optimality. The TCG algorithm is very low in computational complexity — both training and testing are linear in the number of neurons, it does not require hardly any extra storage space, and it is purely incremental, discarding the input at each t. If extended to use adaptive lateral connections, all the units could operate asynchronously. It is very general, describing a framework including unsupervised learning, supervised learning, semi-supervised learning, and communicative learning.

B. Related Work

Artificial neural networks have traditionally operated using bottom–up (feedforward) connections as primary connections, with the top–down (feedback) connections approximately used in a separate weight-tuning mode. For example, backpropagation-based networks [29], [38] use the top–down *error signal* to train the bottom–up weights, but it does not use explicit top–down connections. Learning vector quantization (LVQ) [39] integrates a SOM-style learning rule with class label information by negative "unlearning" if the label of a new sample's closest match would be incorrect, but it does not actually use the label as an input. In comparison, the TCG algorithm uses the label only indirectly represented by activity from upstream neurons, and it does not do any unlearning.

A few networks used top–down information explicitly as part the input. The use of an expanded input including both input and output vectors for the SOM was briefly mentioned as a possibility by Kohonen 1997 [40]. In the SOM extension laterally interconnected synergetically self-organizing map (LISSOM), which used adaptive lateral connections [41], neurons take input from bottom–up, lateral and top–down connections. Sit and Miikkulainen 2006 [42] explained how a neuron responding to an edge, for example, develops to receive top–down feedback from neurons that detect corners including that edge in the next layer. LISSOM was not designed as a classifier or regressor so it did not address the issue of top–down biasing of a limited internal resource with respect to task performance.

C. Topography and Modularity

The topographic organization of many cortical areas such as somatosensory, motor, and visual, is well-known. SOM and LISSOM showed that in some cases lateral excitation can be the impetus for the development of topography [15], [41]. In SOM, and in this paper, lateral excitation occurs from isotropic updating, where a firing neuron will excite all the neurons around it, decreasing as a function of radial distance. But an isotropic lateral excitation function is not biologically supported. In LISSOM, an orientation map with such patchy connectivity developed by incorporating adaptive, limited-range, lateral connections. LISSOM's initial short-range isotropic lateral excitation caused initial smoothness — the features represented in the local area tended to be similar. After some learning, excitatory connections between neurons that represented independent stimuli were not present.

The brain is also characterized by a modular organization [43]. In the modularity as described there, nodes within each module are highly interconnected but not well-connected to neurons in other modules. There are a few "hub" nodes, some with many connections within a module, and some with connections between modules. We note that the TCG method develops networks with modular connectivity. In the experimental three layer networks, a module is a motor neuron and its associated feature neurons. A single motor neuron and its supporting neurons project in an excitatory way exclusively within the module, while only a few border neurons have connections to multiple motor neurons.

VIII. Conclusion

The work reported here showed how top–down connections in a self-organizing Hebbian learning network with sparse coding and quasi-optimal updating lead to topographic class grouping. Further, the work explains why TCG leads to significantly lower error rates. Class distributions are separated and farther apart in the top–down boosted input space. Such a weighting encourages class specificity in neuron representations as less class-specific neurons will have a greatly decreased chance to win in competition. Using 3×3 neighborhood smoothing in the top–down space, class grouping on the neuronal plane emerges, since the neurons that represent each class well are the only possible winners. The emergent class-specific neurons lie near high-density areas in the joint probability, and actively "recruit" nearby class-mixed neurons. The lateral excitatory effects enable local smoothing of resource within class distributions, but discourage smoothing of resource between class distributions. The algorithm is biologically plausible and of low complexity, making it suitable to build on for AMD. It handles stability and plasticity via LCA's tuning of learning rate. Experimental results in some visual recognition problems (without needing attention) showed that networks self-organized to group the classes, some of which have high variability, using a limited resource. Top–down connections led to networks that classified with lower errors compared to networks of the same size without top–down. TCG networks deliver similar performance to other comparable methods, and are scalable to deeper networks with local analysis, for use in more complex problems and architectures. This work has implications towards understanding how internal representation can be developed and maintained, and for understanding how cortex develops category-specific grouped areas.

Appendix
Developmental Entropy

Typically, entropy is used to measure the uncertainty at a certain point in time. For example, it can measure how class-mixed a neuron is after learning has completed by its sampled probability of firing for the different classes in the testing phase. Developmental entropy measures class-specificity throughout learning, and we used it to evaluate the learning algorithm. Neuron i developed from few classes if its entropy of weighted stimulus history is small. From (11), it can be seen that bottom–up weight $\mathbf{v}_i$ of neuron i is a weighted sum of input samples

$$\mathbf{v}_i = \sum_{t=1}^{T} \alpha_i(t)\mathbf{x}(t) \tag{19}$$

where $\mathbf{x}(t)$ are stimuli (samples) that were used to update and $\alpha_i(t)$ are the weights retroactively defining each sample's contribution. The current $\alpha_i(T) = \gamma_i(T)$, and for all previous weights, $\sum_d^{T-1} \alpha_i(d) = 1 - \gamma_i(T)$.

The empirical probability that samples arose from class j is simply the sum of weights for class j's samples

$$p_{i,j} = \sum_{\mathbf{x}(d) \in c_j} \alpha_i(d). \tag{20}$$

To quantify entropy of the probability distribution for the ith neuron, we have

$$-\sum_{d=1}^{m} p_{i,d} \log_m(p_{i,d}). \tag{21}$$

References

[1] J. H. Maunsell and D. C. Van Essen, "The connections of the middle temporal visual area (mt) and their relationship to a cortical hierachy in the macaque monkey," *J. Neurosci.*, vol. 3, no. 12, pp. 2563–2586, 1983.

[2] D. J. Felleman and D. C. Van Essen, "Distributed hierarchical processing in the primate cerebral cortex," *Cereb. Cortex*, vol. 1, no. 11, pp. 1–47, 1991.

[3] J. Bullier, "Hierarchies of cortical areas," in *The Primate Visual System*, J. H. Kaas and C. E. Collins, Eds. New York: CRC Press, 2004, pp. 181–204.

[4] R. J. Douglas, C. Koch, M. Mahowald, K. A. Martin, and H. H. Suarez, "Recurrent excitation in neocortical circuits," *Science*, vol. 269, no. 5226, pp. 981–981, 1995.

[5] K. Tanaka, "Inferotemporal cortex and object vision," *Annu. Rev. Neurosci.*, vol. 19, pp. 109–139, 1996.
[6] R. Epstein and N. Kanwisher, "A cortical representation of the local visual environment," *Nature*, vol. 392, no. 6676, pp. 598–601, 1998.
[7] R. Desimone, T. D. Albright, C. G. Gross, and C. Bruce, "Stimulus-selective properties of inferior temporal neurons in the macaque," *J. Neurosci.*, vol. 4, no. 8, pp. 2051–2062, 1984.
[8] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.*, vol. 160, no. 1, pp. 107–155, 1962.
[9] A. J. Bell and T. J. Sejnowski, "The 'independent components' of natural scenes are edge filters," *Vis. Res.*, vol. 37, no. 23, pp. 3327–3338, 1997.
[10] J. Weng and M. Luciw, "Dually-optimal neuronal layers: Lobe component analysis," *IEEE Trans. Autonom. Mental Develop.*, vol. 1, no. 1, pp. 68–85, May 2009.
[11] H. B. Barlow, "The coding of sensory messages," *Current Prob. Animal Behav.*, pp. 331–360, 1961.
[12] H. Barlow, "Redundancy reduction revisited," *Netw.: Comput. Neural Syst.*, vol. 12, pp. 241–253, 2001.
[13] A. Hyvarinen, J. Karhunen, and E. Oja, *Independent Component Analysis*. New York: Wiley, 2001.
[14] J. Weng, Y. Zhang, and W. Hwang, "Candid covariance-free incremental principal component analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 8, pp. 1034–1040, Aug. 2003.
[15] T. Kohonen, *Self-Organizing Maps*, 3rd ed. Berlin, Germany: Springer-Verlag, 2001.
[16] B. A. Olshaushen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," in *Nature*, Jun. 13, 1996, vol. 381, pp. 607–609.
[17] S. Harnad, "The symbol grounding problem," *Physica D*, vol. 42, pp. 335–346, 1990.
[18] J.-P. Braquelaire and L. Brun, "Comparison and optimization of methods of color image quantization," *IEEE Trans. Image Process.*, vol. 6, pp. 1048–1051, Jun. 1997.
[19] X. Wu and K. Zhang, "A better tree-structured vector quantizer," in *Proc. Data Compress. Conf., DCC'91*, Snowbird, Utah, 1991, pp. 392–401.
[20] S. M. Pan and K. S. Cheng, "An evolution-based tabu search approach to codebook design," *Pattern Recogn.*, vol. 40, no. 2, pp. 476–491, 2007.
[21] E. L. Lehmann, *Theory of Point Estimation*. New York: Wiley, 1983.
[22] P. Dayan and L. F. Abbott, *Theoretical Neuroscience*. State College, PA: Citeseer, 2001.
[23] T. Heskes, "Energy functions for self-organizing maps," *Kohonen Maps*, pp. 303–316, 1999.
[24] J. Lampinen and T. Kostiainen, "Generative probability density model in the self-organizing map," *Stud. Fuzz. Soft Comput.*, vol. 78, pp. 75–94, 2002.
[25] M. Solgi and J. Weng, "Developmental stereo: Emergence of disparity preference in models of the visual cortex," *IEEE Trans. Autonom. Mental Develop.*, vol. 1, no. 4, pp. 238–252, Dec. 2009.
[26] M. Luciw and J. Weng, "Laterally connected lobe component analysis: Precision and topography," in *Proc. 9th Int. Conf. Develop. Learn. (ICDL '09)*, Shanghai, China, Jun. 5–7, 2009.
[27] M. D. Luciw and J. Weng, "Topographic class grouping with applications to 3D object recognition," in *Proc. Int. Joint Conf. Neural Netw.*, Hong Kong, Jun. 1–6, 2008.
[28] S. Schaal and C. G. Atkeson, "Constructive incremental learning from only local information," *Neural Comput.*, vol. 10, no. 8, pp. 2047–2084, 1998.
[29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[30] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.
[31] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proc. IEEE Conf. Comput. Vis. Pattern Recogn.*, Washington, DC, 2004.
[32] V. Nair and G. Hinton, "3-d object recognition with deep belief nets," *Adv. Neural Inform. Process. Syst.*, 2010.
[33] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Adv. Neural Inform. Proces. Syst. 19: Proc. 2006 Conf.*, Vancouver, BC, Canada, 2007, p. 153.
[34] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, 2002.
[35] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards AI," *Large-Scale Kernel Mach.*, pp. 321–360, 2007.
[36] Z. Ji and J. Weng, "Where what network-2: A biologically inspired neural network for concurrent visual attention and recognition," in *Proc. IEEE World Congress Comput. Intell.*, Barcelona, Spain, 2010.
[37] M. Luciw and J. Weng, "Where what network-3: Developmental top-down attention with multiple foregrounds and complex backgrounds," in *Proc. IEEE World Congress Comput. Intell.*, Barcelona, Spain, 2010.
[38] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. New York: Wiley-Interscience, 1994.
[39] T. Kohonen, "An introduction to neural computing," *Neural Netw.*, vol. 1, no. 1, pp. 3–16, 1998.
[40] T. Kohonen, *Self-Organizing Maps*, 2nd ed. Berlin, Germany: Springer-Verlag, 1997.
[41] R. Miikkulainen, J. A. Bednar, Y. Choe, and J. Sirosh, *Computational Maps in the Visual Cortex*. Berlin, Germany: Springer-Verlag, 2005.
[42] Y. F. Sit and R. Miikkulainen, "Self-organization of hierarchical visual maps with feedback connections," *Neurocomputing*, vol. 69, pp. 1309–1312, 2006.
[43] E. Bullmore and O. Sporns, "Complex brain networks: Graph theoretical analysis of structural and functional systems," *Nature Rev. Neurosci.*, vol. 10, no. 3, pp. 186–198, 2009.

Matthew Luciw (S'06–M'09) received the M.Sc. degree in 2006, and Ph.D. degree in 2010, both in computer science from Michigan State University (MSU), East Lansing.

He is currently a Researcher at the Dalle Molle Institute for Artificial Intelligence (IDSIA), Manno-Lugano, Switzerland. He was previously a member of the Embodied Intelligence Laboratory at MSU. His research involves the study of biologically inspired algorithms for autonomous development of mental capabilities, especially for visual attention and recognition. He is a member of the IEEE Computational Intelligence Society.

Juyang (John) Weng (S'88–M'89–SM'05–F'09) received the B.Sc. degree from Fudan University, Shanghai, China, and the M.Sc. and Ph.D. degrees 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. He is also a Faculty Member of the Cognitive Science Program and the Neuroscience Program at Michigan State University. Since Cresceptron (the first published work that reports recognizing and segmenting general objects from their visual appearances in natural complex backgrounds) he has further expanded his research interests in brain-mind inspired systems, especially a computationally efficient and unified framework for the autonomous development of a variety of mental capabilities by active robots and animals, including perception, cognition, behaviors, motivation, and thinking. He has published research articles on related subjects, including task muddiness, intelligence metrics, mental architectures, vision, audition, touch, attention, recognition, autonomous navigation, reaching, manipulation, and language acquisition. He and his co-workers developed SAIL and Dav robots as research platforms for research on autonomous mental development.

Dr. Weng is an Editor-in-Chief of *International Journal of Humanoid Robotics* and an Associate Editor of the new IEEE TRANSACTIONS ON AUTONOMOUS MENTAL DEVELOPMENT, as well as a member of the Executive Board of the International Neural Network Society. He was a PI (with Ida Stockman) and a program chairman for the NSF/DARPA funded Workshop on Development and Learning 2000 (1st ICDL), the Chairman of the Governing Board of the International Conferences on Development and Learning (ICDL) (2005–2007, http:/ /cogsci.ucsd.edu/ triesch/icdl/), the chairman of the Autonomous Mental Development Technical Committee of the IEEE Computational Intelligence Society (2004–2005), a program chairman of 2nd ICDL, and a general chairman of 7th ICDL (2008) and 8th ICDL (2009). He was also an Associate Editor of the IEEE TRANSACTIONS ON PATTERN RECOGNITION AND MACHINE INTELLIGENCE and an Associate Editor of the IEEE TRANSACTIONS ON IMAGE PROCESSING.