Unsupervised Hebbian learning by recurrent multilayer neural networks for temporal hierarchical pattern recognition

James Ting-Ho Lo*

Abstract – Recurrent multilayer network structures and Hebbian learning are two essential features of biological neural networks. An artificial recurrent multilayer neural network that performs supervised Hebbian learning, called probabilistic associative memory (PAM), was recently proposed. PAM is a recurrent multilayer network of processing units (PUs), each processing unit comprising a group of novel artificial neurons, which generate spike trains. PUs are detectors and recognizers of the feature subvectors appearing in their receptive fields. In supervised learning by a PU, the label of the feature subvector is provided from outside PAM. Since the feature subvector may be shared by many causes and may contain parts from many causes, the label of the feature subvector is sometimes difficult to obtain, not to mention the cost, especially if there are many hidden layers and feedbacks.

This paper presents an unsupervised learning scheme, which is Hebbian in the following sense: The strength of a synapse increases if the outputs of the presynaptic and postsynaptic neurons are identical and decreases otherwise. This unsupervised Hebbian learning capability makes PAM a good functional model of neuronal networks as well as a good learning machine for temporal hierarchical pattern recognition.

Key Words – unsupervised learning; learning machine; recurrent neural network; Hebbian learning; multilayer neural network; spike trains; probability distribution; maximal generalization; orthogonal expansion.

1 Introduction

Commonly used pattern classifiers include SVMs (support vector machines), MLPs (multilayer perceptrons), RBF (radial basis function) networks [11, 21, 3, 10], template matching, nearest mean classifiers, subspace methods, 1-nearest neighbor rule, k-nearest neighbor rule, Bayes plug-in, logistic classifiers, Parzen classifiers, Fisher linear discriminants, and binary decision trees [6, 24]. They each are suitable for some classification problems. However, in general, they all suffer from some of such shortcomings as difficult training/design, much computation/memory requirement, ad hoc characteristics of the penalty function, poor generalization/performance, etc.

Two recent pattern classification schemes, that are seldom mentioned in the pattern recognition literature, are the holographic neural nets (HNets) [23] and the sparsely encoded correlation matrix memories (CMMs) [26, 27, 17, 20, 7, 1, 25, 12]. HNets are also CMMs. These two types of CMM are relatively new members of the associative memories family which have long been studied in the neural networks community [14, 11, 2, 10, 21]. The successes of HNets and sparsely encoded CMMs in learning and recognizing large patterns with good generalization abilities indicate that associative memories based on Hebbian learning have significant advantages. Nevertheless, HNets suffer from ambiguity (or high error rate), and both HNets and sparsely encoded CMMs suffer from low memory density and design difficulties.

We further remark that most, if not all, of the commonly used pattern classifiers and CMMs cannot recognize multiple causes with different class labels in a pattern. However, many patterns can be fully recognized only if the hierarchical causes in the patterns can be detected and recognized. Furthermore, the commonly used pattern classifiers and CMMs except recurrent MLPs do not have recurrent structures to recognize temporal or scanned patterns.

To alleviate these and the foregoing shortcomings of the commonly used pattern classifiers and CMMs, a new neural network paradigm, called probabilistic associative memory (PAM), was proposed in [15]. PAM can be viewed as an organization of recurrent multilayer networks of computing nodes. It is a recurrent multilayer of processing units (PUs), each PU being a CMM that generates point estimates as well as subjective probabilities of the label of each feature subvector appearing in the PU's receptive field. The weights in each PU learn by a supervised Hebb rule. The PU can tolerate, with gracefully degraded performance, up to a preset percentage of errors in the components of its input feature subvector. This amounts to using feature subvector similarity for generalization at the feature subvector level.

However, PAM in [15] suffers from lacking unsupervised learning ability. Assigning labels to a large number of feature subvectors appearing in the receptive fields of PUs is

*Department of Mathematics and Statistics, University of Maryland Baltimore County, Baltimore, MD 21250, USA, Email: jameslo@umbc.edu

very expensive and difficult, because the feature subvector may be shared by many causes and may contain parts from many causes. In online learning, it is simply impossible if the PAM has many layers and many feedbacks.

A well-known unsupervised learning method based on a kind of the Hebb rule is the Oja learning algorithm [18, 22, 19, 5, 10]. Learning with this algorithm, the synaptic weight vectors converge to the principal directions of the covariance matrix of the vectors input to the feedforward linear neural network. In response to an input vector, the trained feedforward network outputs its principal components. The Oja learning algorithm is ingenious, but suffers from the following shortcomings: (1) The algorithm converges asymptotically. (2) If the dimensionality of the output vector of the feedforward network is smaller than that of the input vector, much information contained in the input vector is lost unless the eigenvalues of the covariance matrix taper down fast, which is true only if the input vectors do not have too many major features. (3) It is not clear how the algorithm can be extended to training a multilayer or recurrent network.

The purpose of this paper is to show how PUs, also called processing elements (PEs), in [15] can perform unsupervised Hebbian learning. Given the structures and functions of PAM and its PUs, unsupervised Hebbian learning is performed in essentially the same way as supervised learning, making it easy to switch between the two. When a desired output (i.e., label) is unavailable for a PU, the output of the PU is used as the desired output. This is in fact what the Hebb rule is all about. Compared with other unsupervised (or supervised) learning schemes, a main advantage of PAM is the advantage of the Hebb rule – simplicity. It requires no differentiation, optimization, cycling through all learning data iteratively, or waiting for an asymptotic convergence to emerge.

In the rest of this Section, we will establish terminologies and notations. In subsequent sections, PAM will be reviewed briefly before the unsupervised Hebbian learning scheme is described.

A ternary vector is a vector with components from the ternary set $\{-1, 0, 1\}$. A ternary vector input to PAM is called an exogenous feature vector, and a ternary vector input to a layer of PUs is called a feature vector. A feature vector input to a layer usually contains not only feedforwarded outputs from a preceding layer but also feedbacked outputs from the same or other layers with a time delay. A feature vector may contain components from an exogenous feature vector. For simplicity, we assume that the exogenous feature vector is only input to layer 1.

A subvector of a feature vector that is input to a PU is called a feature subvector. Trace the feedforward connections backward from a PU to a subvector of the exogenous feature vector. This subvector is called the receptive field of the PU. The feature subvector directly input to the PU, also called the receptive field of the PU, is assigned the same label as the subvector of the exogenous feature vector that appears in the receptive field of the PU.

The feature vector input to layer l with numbering t is denoted by x_t^{l-1}, and the output from the layer at t is denoted by $x\left\{y_t^l\right\}$, which is a point estimate of the label of the feature vector x_t^{l-1}. An exogenous feature vector is denoted by x_t^{ex}. It is a subvector of x_t^0, which may contain feedbacked components. For notational simplicity, the superscript $l-1$ in x_t^{l-1} and dependencies on $l-1$ or l in other symbols are sometimes suppressed in the following when no confusion is expected.

Let x_t, $t = 1, 2, \ldots$, denote a sequence of M-dimensional feature vectors $x_t = \left[\begin{array}{ccc} x_{t1} & \cdots & x_{tM} \end{array}\right]'$, whose components are ternary numbers. Let $n = \left[\begin{array}{ccc} \mathbf{n}_1 & \cdots & \mathbf{n}_k \end{array}\right]'$ be a subvector $\left[\begin{array}{ccc} 1 & \cdots & M \end{array}\right]'$ such that $\mathbf{n}_1 < \cdots < \mathbf{n}_k$. The subvector $x_t(\mathbf{n}) := \left[\begin{array}{ccc} x_{t\mathbf{n}_1} & \cdots & x_{t\mathbf{n}_k} \end{array}\right]'$ of x_t is a feature subvector of the feature vector x_t. $\mathbf{n}$ is called a feature subvector index (FSI), and $x_t(\mathbf{n})$ is said to be a feature subvector on the FSI $\mathbf{n}$ or have the FSI $\mathbf{n}$. Each PU is associated with a fixed FSI $\mathbf{n}$ and denoted by PU($\mathbf{n}$). Using these notations, the sequence of subvectors of x_t, $t = 1, 2, \ldots$, that is input to PU($\mathbf{n}$) is $x_t(\mathbf{n})$, $t = 1, 2, \ldots$. The FSI $\mathbf{n}$ of a PU usually has subvectors, $\mathbf{n}(u)$, $u = 1, ..., U$, on which subvectors $x_t(\mathbf{n}(u))$ of $x_t(\mathbf{n})$ are separately processed by PU($\mathbf{n}$) at first. The subvectors, $\mathbf{n}(u)$, $u = 1, ..., U$, are not necessarily disjoint, but are all inclusive in the sense that every component of $\mathbf{n}$ is included in at least one of the subvectors $\mathbf{n}(u)$. Moreover, the components of $\mathbf{n}(u)$ are usually randomly selected from those of $\mathbf{n}$.

The PUs in layer l have FSIs (feature subvector indices) denoted by $\mathbf{1}^l$, $\mathbf{2}^l$, ..., $\mathbf{N}^l$. Upon receiving a feature vector x_τ^{l-1} by layer l, the feature subvectors, $x_\tau^{l-1}\left(\mathbf{1}^l\right)$, $x_\tau^{l-1}\left(\mathbf{2}^l\right)$, ..., $x_\tau^{l-1}\left(\mathbf{N}^l\right)$, are formed and processed by the PUs, PU$\left(\mathbf{1}^l\right)$, PU$\left(\mathbf{2}^l\right)$, ..., PU$\left(\mathbf{N}^l\right)$, to generate $x\left\{y_\tau^l\left(\mathbf{1}^l\right)\right\}$, $x\left\{y_\tau^l\left(\mathbf{2}^l\right)\right\}$, ..., $x\left\{y_\tau^l\left(\mathbf{N}^l\right)\right\}$, respectively. These ternary vectors are then assembled into the output vector $x\left\{y_\tau^l\right\}$ of layer l. The symbols, y_t^l and $x\left\{y_t^l\right\}$, are defined and explained in more detail in Section **??**.

The components of a feature vector x_τ^{l-1} input to layer l at time (or with numbering) τ comprise components of ternary vectors generated by PUs in layer $l-1$ and those generated at a previous time by PUs in the same layer l or PUs in higher layers with layer numbering $l+k$ for some positive integers k. The time delays may be of different durations.

Once an exogenous feature vector is received by PAM, the PUs perform functions of retrieving and/or learning from layer to layer starting with layer 1, the lowest layer. After the PUs in the highest layer, layer L, complete performing their functions, PAM is said to have completed one round of retrievings and/or learnings (or memory adjustments). For each exogenous feature vector, PAM will

continue to complete a certain number of rounds of retrievings and/or learnings.

2 Orthogonal Expansion

The orthogonal expansion $\breve{v}$ of an m-dimensional ternary vector $v = \left[\begin{array}{ccc} v_1 & \cdots & v_m \end{array}\right]'$ is defined by $\breve{v}(1) = \left[\begin{array}{cc} 1 & v_1 \end{array}\right]'$, and for $j = 1, ..., m-1$,

$$\breve{v}(1, ..., j+1) = \left[\begin{array}{cc} \breve{v}'(1, ..., j) & v_{j+1}\breve{v}'(1, ..., j) \end{array}\right]' \quad (1)$$

Note that the basic operation in the orthogonal expansion is the multiplication of ternary numbers. Let $a = \left[\begin{array}{ccc} a_1 & \cdots & a_m \end{array}\right]'$ and $b = \left[\begin{array}{ccc} b_1 & \cdots & b_m \end{array}\right]'$ be two m-dimensional ternary vectors. Then the inner product $\breve{a}'\breve{b}$ of their orthogonal expansions, $\breve{a}$ and $\breve{b}$, can be expressed as follows:

$$\breve{a}'\breve{b} = \prod_{j=1}^{m}(1 + a_j b_j) \quad (2)$$

If $a_k b_k = -1$ for some $k \in \{1, \ldots, m\}$, then $\breve{a}'\breve{b} = 0$. If $a_k b_k = 0$ for some $k \in \{1, \ldots, m\}$, then $\breve{a}'\breve{b} = \prod_{j=1, j\neq k}^{m}(1 + a_j b_j)$. If $\breve{a}'\breve{b} \neq 0$, then $\breve{a}'\breve{b} = 2^{a'b}$. If a and b are bipolar binary vectors, then $\breve{a}'\breve{b} = 0$ if $a \neq b$; and $\breve{a}'\breve{b} = 2^m$ if $a = b$. The proof of these can be found in [15].

Once the feature subvector $x_\tau(\mathbf{n})$ is received by PU($\mathbf{n}$), its subvectors $x_\tau(\mathbf{n}(u))$, $u = 1, ..., U$, on preset feature subvector indices $\mathbf{n}(u)$, are separately orthogonalized into $\breve{x}_\tau(\mathbf{n}(u))$. Combining these orthogonal expansions, we obtain the general orthogonal expansion,

$$\breve{x}_t(\mathbf{n}) = \left[\begin{array}{cccc} \breve{x}'_t(\mathbf{n}(1)) & \breve{x}'_t(\mathbf{n}(2)) & \cdots & \breve{x}'_t(\mathbf{n}(U)) \end{array}\right]' \quad (3)$$

Notice that (1) involves only ternary number multiplication, and that the first component 1 of $\breve{x}_\tau(\mathbf{n}(u))$ is the square of any component of $x_\tau(\mathbf{n}(u))$. Hence, each orthogonal expansion $\breve{x}_\tau(\mathbf{n}(u))$ is a tree whose nodes perform multiplications. These U trees are the compartmentalized dendritic trees in PU($\mathbf{n}$). They encode the input feature subvector $x_\tau(\mathbf{n})$ into its code $\breve{x}_t(\mathbf{n})$ for learning and retrieving.

3 Synaptic Weights

Let the label of $x_t(\mathbf{n})$, denoted by $r_t(\mathbf{n})$, be an R-dimensional ternary vector. All subvectors, $x_t(\mathbf{n}(u))$, $u = 1, ..., U$, of $x_t(\mathbf{n})$ share the same label $r_t(\mathbf{n})$. In supervised learning, $r_t(\mathbf{n})$ is provided from outside PAM. If $r_t(\mathbf{n})$ is not provided, $r_t(\mathbf{n})$ must be generated internally and unsupervised learning is performed in PU($\mathbf{n}$). This will be discussed later on.

Here we assume that $r_t(\mathbf{n})$, $t = 1, 2, \ldots$, are available – provided from outside PAM or generated within PU($\mathbf{n}$). The pairs $(x_t(\mathbf{n}(u)), r_t(\mathbf{n}))$, $t = 1, 2, \ldots$, are learned by the PU to form expansion correlation matrices (ECMs), $D(\mathbf{n}(u))$ and $C(\mathbf{n}(u))$ on $\mathbf{n}(u)$. After the first T pairs are learned, these matrices are

$$D(\mathbf{n}(u)) = \Lambda \sum_{t=1}^{T} \lambda^{T-t} r_t(\mathbf{n})\, \breve{x}'_t(\mathbf{n}(u)) \quad (4)$$

$$C(\mathbf{n}(u)) = \Lambda \sum_{t=1}^{T} \lambda^{T-t} \breve{x}'_t(\mathbf{n}(u)) \quad (5)$$

where λ is a forgetting factor and Λ is a scaling constant that is selected to keep all numbers involved in PAM manageable. Weight functions other than $\lambda^{T-t}\Lambda$ can be used. Note that the ECMs, $D(\mathbf{n}(u))$ and $C(\mathbf{n}(u))$, are $R \times 2^{\dim \mathbf{n}(u)}$ and $1 \times 2^{\dim \mathbf{n}(u)}$ matrices respectively.

The synaptic weights in PU($\mathbf{n}$) are the general expansion correlation matrices (GECMs),

$$D(\mathbf{n}) = \left[\begin{array}{cccc} D(\mathbf{n}(1)) & D(\mathbf{n}(2)) & \cdots & D(\mathbf{n}(U)) \end{array}\right] \quad (6)$$

$$C(\mathbf{n}) = \left[\begin{array}{cccc} C(\mathbf{n}(1)) & C(\mathbf{n}(2)) & \cdots & C(\mathbf{n}(U)) \end{array}\right] \quad (7)$$

Note that the GECMs, $D(\mathbf{n})$ and $C(\mathbf{n})$, are $R \times \sum_{u=1}^{U} 2^{\dim \mathbf{n}(u)}$ and $1 \times \sum_{u=1}^{U} 2^{\dim \mathbf{n}(u)}$ matrices respectively.

4 Computation by Neurons

There are R neurons in PU($\mathbf{n}$). Neuron j's $2\sum_{u=1}^{U} 2^{\dim \mathbf{n}(u)}$ synaptic weights are the entries of the jth row $D_j(\mathbf{n})$ of $D(\mathbf{n})$ and $C(\mathbf{n})$. Note that to reduce the memory requirement, $C(\mathbf{n})$ can be shared by the R neurons. In response to $\breve{x}_\tau(\mathbf{n})$, the jth neuron performs the following computation:

$$d_{\tau j}(\mathbf{n}) = D_j(\mathbf{n})\, \breve{x}_\tau(\mathbf{n}) = \sum_{u=1}^{U} D_j(\mathbf{n}(u))\, \breve{x}_\tau(\mathbf{n}(u)) \quad (8)$$

$$c_\tau(\mathbf{n}) = C(\mathbf{n})\, \breve{x}_\tau(\mathbf{n}) = \sum_{u=1}^{U} C(\mathbf{n}(u))\, \breve{x}_\tau(\mathbf{n}(u)) \quad (9)$$

Note that $c_\tau(\mathbf{n})$ is approximately the total number of times $x_\tau(\mathbf{n})$ has been learned. Note also that $d_{\tau j}(\mathbf{n})$ is approximately the total number of times $x_\tau(\mathbf{n})$ has been learned with the jth component $r_{\tau j}(\mathbf{n})$ of $r_\tau(\mathbf{n})$ being $+1$ minus the total number of times $x_\tau(\mathbf{n})$ has been learned with the jth component $r_{\tau j}(\mathbf{n})$ of $r_\tau(\mathbf{n})$ being -1. They are approximates because of the effects of forgetting factors and normalizing constant involved in computing them.

Therefore, $(c_\tau(\mathbf{n}) + d_{\tau j}(\mathbf{n}))/2$ is the total number of times $x_\tau(\mathbf{n})$ has been learned with the jth component

$r_{\tau j}(\mathbf{n})$ of $r_\tau(\mathbf{n})$ being +1. Consequently, $(y_{\tau j}(\mathbf{n})+1)/2$, with $y_{\tau j}(\mathbf{n})$ denoting $d_{\tau j}(\mathbf{n})/c_\tau(\mathbf{n})$, is the subjective probability $p_{\tau j}(\mathbf{n})$ that $r_{\tau j}(\mathbf{n})$ is equal to +1. The jth neuron then uses a pseudo-random generator to generate $x\{y_{\tau j}(\mathbf{n})\} = +1$ with probability $p_{\tau j}(\mathbf{n})$ and $x\{y_{\tau j}(\mathbf{n})\} = -1$ with probability $1- p_{\tau j}(\mathbf{n})$. This +1 or −1 is the output of the jth neuron.

The vector

$$p_\tau(\mathbf{n}) = \left[\begin{array}{cccc} p_{\tau 1}(\mathbf{n}) & p_{\tau 2}(\mathbf{n}) & \cdots & p_{\tau R}(\mathbf{n}) \end{array}\right]'$$

is a representation of a subjective probability distribution of the label $r_\tau(\mathbf{n})$ of the feature subvector $x_\tau(\mathbf{n})$ input to PU($\mathbf{n}$). The outputs of the R neurons in response to $x_\tau(\mathbf{n})$ form a bipolar vector, which is a point estimate of the label $r_\tau(\mathbf{n})$ of $x_\tau(\mathbf{n})$.

5 Maximal Generalization

Let a feature subvector that deviates from each of a group of feature subvectors that have been learned by PU($\mathbf{n}$) due to corruption, distortion or occlusion be presented to the PU. If the PU is able to automatically find the largest subvector of the presented subvector that matches at least one subvector among the group and generate the SPD of the label of the largest subvector, the PU is said to have a maximal generalization capability. This maximal capability is achieved by the use of masking matrices described in this section.

Let us denote the vector $v = \left[\begin{array}{cccc} v_1 & v_2 & \cdots & v_n \end{array}\right]'$ with its i_1-th, i_2-th, ..., and i_j-th components set equal to 0 by $v\left(i_1^-, i_2^-, \ldots, i_j^-\right)$, where $1 \leq i_1 < i_2 < ... < i_j \leq n$. Denoting the n-dimensional vector $\left[\begin{array}{cccc} 1 & 1 & \cdots & 1 \end{array}\right]'$ by $\mathbf{I}$ and denoting the orthogonal expansion of $v\left(i_1^-, i_2^-, \ldots, i_j^-\right)$ by $\breve{v}\left(i_1^-, i_2^-, \ldots, i_j^-\right)$, we note that $v\left(i_1^-, i_2^-, \ldots, i_j^-\right) = \mathrm{diag}\left(\mathbf{I}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right) v$ and $\breve{v}\left(i_1^-, i_2^-, \ldots, i_j^-\right) = \mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right) \breve{v}$.

Using these notations, a feature subvector $x(\mathbf{n}(u))$ with its i_1-th, i_2-th, ..., and i_j-th components set equal to 0 is $x_t(\mathbf{n}(u))\left(i_1^-, i_2^-, \ldots, i_j^-\right)$, and the orthogonal expansion of $x_t(\mathbf{n}(u))\left(i_1^-, i_2^-, \ldots, i_j^-\right)$ is $\mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right) \breve{x}_t(\mathbf{n}(u))$. Hence, the matrix $\mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right)$, as a matrix transformation, sets the i_1-th, i_2-th, ..., and i_j-th components of $x_t(\mathbf{n}(u))$ equal to zero in transforming $\breve{x}_t(\mathbf{n}(u))$ (i.e., in $\mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right) \breve{x}_t(\mathbf{n}(u))$). Therefore, $\mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right)$ is called a masking matrix.

An important property of the masking matrix $\mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right)$ is the following: If

$$\begin{aligned} & \mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right) \breve{x}_t(\mathbf{n}(u)) \\ = \; & \mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right) \breve{x}_\tau(\mathbf{n}(u)) \end{aligned}$$

then

$$\breve{x}'_t(\mathbf{n}(u)) \,\mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right) \breve{x}_\tau(\mathbf{n}(u)) = 2^{\dim \mathbf{n}(u) - j}.$$

If

$$\begin{aligned} & \mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right) \breve{x}_t(\mathbf{n}(u)) \\ \neq \; & \mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right) \breve{x}_\tau(\mathbf{n}(u)) \end{aligned}$$

then $\breve{x}'_t(\mathbf{n}(u))\mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right) \breve{x}_\tau(\mathbf{n}(u)) = 0$.

Using this property, we combine all such masking matrices that set less than or equal to a selected positive integer $J(\mathbf{n}(u))$ of components of $x_t(\mathbf{n}(u))$ equal to zero into the following masking matrix

$$\begin{aligned} M(\mathbf{n}(u)) \;=\; & I + \sum_{j=1}^{J(\mathbf{n}(u))\dim \mathbf{n}(u)} \sum_{i_j=j} \cdots \sum_{i_2=2}^{i_3-1} \sum_{i_1=1}^{i_2-1} \\ & 2^{-6j} 2^j \mathrm{diag}\left(\breve{\mathbf{I}}\left(i_1^-, i_2^-, \ldots, i_j^-\right)\right) \end{aligned} \tag{10}$$

where 2^j is used to compensate for the factor 2^{-j} in $2^{\dim \mathbf{n}(u) - j}$ in the important property stated above, and 2^{-6j} is an example weight selected to differentiate between different levels j of maskings.

Corresponding to $\breve{x}_t(\mathbf{n})$, $D(\mathbf{n})$ and $C(\mathbf{n})$ defined in (3), (6) and (7), a general masking matrix is defined as follows:

$$M(\mathbf{n}) = \mathrm{diag}\left[\begin{array}{cccc} M(\mathbf{n}(1)) & M(\mathbf{n}(2)) & \cdots & M(\mathbf{n}(U)) \end{array}\right] \tag{11}$$

where the right side is a matrix with $M(\mathbf{n}(u))$, $u = 1, 2, ..., U$, as diagonal blocks and zero elsewhere.

If the masking matrix $M(\mathbf{n}(u))$ is used, the symbols $c_\tau(\mathbf{n}(u))$, $d_\tau(\mathbf{n}(u))$ are defined as follows:

$$d_\tau(\mathbf{n}(u)) := D(\mathbf{n}(u))\, M(\mathbf{n}(u))\, \breve{x}_\tau(\mathbf{n}(u)) \tag{12}$$

$$c_\tau(\mathbf{n}(u)) := C(\mathbf{n}(u))\, M(\mathbf{n}(u))\, \breve{x}_\tau(\mathbf{n}(u)) \tag{13}$$

With the masking matrix $M(\mathbf{n})$, the symbols $c_\tau(\mathbf{n})$, $d_\tau(\mathbf{n})$ are in turn defined as follows:

$$d_\tau(\mathbf{n}) := D(\mathbf{n})\, M(\mathbf{n})\, \breve{x}_\tau(\mathbf{n}) \tag{14}$$

$$c_\tau(\mathbf{n}) := C(\mathbf{n})\, M(\mathbf{n})\, \breve{x}_\tau(\mathbf{n}) \tag{15}$$

where $\breve{x}_t(\mathbf{n})$ is a general orthogonal expansion (GOE) and $D(\mathbf{n})$ and $C(\mathbf{n})$ are general expansion correlation matrices

(GECMs) for PU(**n**). It follows that

$$d_\tau(\mathbf{n}) = \sum_{u=1}^{U} d_\tau(\mathbf{n}(u)) \tag{16}$$

$$c_\tau(\mathbf{n}) = \sum_{u=1}^{U} c_\tau(\mathbf{n}(u)) \tag{17}$$

which are used instead of those in (8) and (8) in the computation by neurons.

If terms in (10) are missing, PU(**n**) suffers only graceful degradation of its generalization capability. From the neurobiological point of view, a masking matrix is a mathematical idealization and organization of a large number of nested and overlapped subvectors of feature subvectors and their dendritic trees.

6 Supervised and Unsupervised Learning

ECMs, $D(\mathbf{n}(u))$ and $C(\mathbf{n}(u))$, in (4) and (5) are adjusted to learn a pair $(x_t(\mathbf{n}(u)), r_t(\mathbf{n}))$ as follows: If $r_t(\mathbf{n}) \neq 0$,

$$D(\mathbf{n}(u)) \leftarrow \lambda D(\mathbf{n}(u)) + \Lambda r_t(\mathbf{n}) \breve{x}'_t(\mathbf{n}(u)) \tag{18}$$

$$C(\mathbf{n}(u)) \leftarrow \lambda C(\mathbf{n}(u)) + \Lambda \breve{x}'_t(\mathbf{n}(u)) \tag{19}$$

If $r_t(\mathbf{n}) = 0$, then $D(\mathbf{n}(u))$ and $C(\mathbf{n}(u))$ are unchanged. For supervised learning, $r_t(\mathbf{n})$ is provided from outside PAM and dominates the neurons outputs, thereby allowing the Hebb rule to be applied by PU(**n**) to increment $D(\mathbf{n}(u))$ and $C(\mathbf{n}(u))$ by $\Lambda r_t(\mathbf{n}) \breve{x}'_t(\mathbf{n}(u))$ and $\Lambda \breve{x}'_t(\mathbf{n}(u))$, respectively.

For unsupervised learning, $r_t(\mathbf{n})$ must be generated internally by the PU(**n**). By the Hebb rule, $r_t(\mathbf{n})$ in (18) should be the output vector $x\{y_\tau(\mathbf{n})\}$ of the R neurons for adjusting $D(\mathbf{n})$. If $(x_t(\mathbf{n}), r_t(\mathbf{n}))$ has been learned and stored in $D(\mathbf{n})$, $x\{y_\tau(\mathbf{n})\}$ is equal to r_t. If $p_\tau(\mathbf{n})$ is not a binary vector (with components being 1 or 0), $x\{y_\tau(\mathbf{n})\}$ is a pseudo-random vector generated by the SPD $p_\tau(\mathbf{n})$. In case, all the components of $p_\tau(\mathbf{n})$ are 1/2, $x\{y_\tau(\mathbf{n})\}$ is a purely (or uniformly distributed) random vector. Otherwise, $x\{y_\tau(\mathbf{n})\}$ resembles in part fragments of those stored feature subvectors that have contributed to the SPD $p_\tau(\mathbf{n})$. Unsupervised learning by PU(**n**) thus creates a vocabulary for itself.

7 Spike Trains for Each Exogenous Feature Vector

The subscript t or τ in $x_\tau(\mathbf{n})$, $y_\tau(\mathbf{n})$, and $x\{y_\tau(\mathbf{n})\}$ denote the time or numbering of the quantities going through PU(**n**) in the preceding Sections. In this section, assume that each exogenous feature vector is presented to PAM for one unit of time, and that during this one unit of time, there are ζ spikes in each spike train. Here, the subscript t or τ denotes only the time the exogenous feature vector x_t^{ex} or x_τ^{ex} arrives at the input terminals of PAM. For each exogenous feature vector x_τ^{ex}, ζ rounds of retrieving and learning are performed by PAM at times, $\tau + i/\zeta$, , $i = 0, 1, ..., \zeta - 1$. Consequently, PU(**n**) generates a sequence of ternary vectors denoted by $x\{y_{\tau+i/\zeta}(\mathbf{n})\}$, $i = 0, 1, ..., \zeta-1$, for each exogenous feature vector x_τ^{ex}. This sequence consists of R spike trains, each having ζ spikes each of $1/\zeta$ unit of time.

A feedback connection from layer $l + k$ to layer l for $k \geq 0$ must have at least one delay device to ensure stability. Each delay device holds a spike for $1/\zeta$ unit of time before it is allowed to pass. Causes in patterns, temporal or spatial, usually form a hierarchy. The higher a layer in PAM is, the higher in the hierarchy the causes the PUs in the layer treat, and the more time it takes for the causes to form and be recognized by the PUs. Therefore, the number of delay devices on a feedback connection is a monotone increasing function of k.

8 Conclusion

An unsupervised Hebbian learning scheme is proposed in this paper. The scheme is made possible with the novel neurons in the PUs (processing units) that generate spike trains. The generation of spike trains is made possible with the orthogonal expansion of the feature subvectors input to the PUs. The orthogonal expansion is believed to be the function of the biological dendritic trees, and this function is what neuroscientists refer to as information encoding.

The unsupervised Hebbian learning is simple, natural and suitable for online learning. It does not involve differentiation, optimization, cycling through all training data iteratively, or waiting for an asymptotic convergence to emerge. A PU in PAM can perform supervised learning whenever a label is provided from outside PAM and perform unsupervised learning otherwise. Spurious exogenous feature vectors may be learned by the unsupervised learning scheme, but they self-destruct like random walks and the residues are eliminated by a forgetting factor.

With the unsupervised Hebbian learning scheme, PAM has the following unique features as a learning machine [4, 16, 13, 3, 9, 12, 10, 8]:

1. a recurrent multilayer network learning by the Hebb rule;

2. fully automated unsupervised and supervised Hebbian learning mechanisms (involving no optimization, iteration or asymptotic behavior);

3. masking matrices for recognizing corrupted, distorted, and occluded patterns;

4. neurons communicating with spike trains; and

5. each PU being a pattern detector and recognizer that computes SPDs (subject probability distributions).

An additional feature of PAM is storing a model of the hierarchical world with higher-layer PUs outputting SPDs about higher-level causes in space and time and lower-layer PUs using such SPDs feedbacked from higher layers to predict, extend or speculate lower-level causes.

References

[1] S. Amari. Characteristics of sparsely encoded associative memory. *Neural Networks*, 2(6):451–457, 1989.

[2] J. A. Anderson. *An Introduction to Neural Networks*. The MIT Press, Cambridge, Massachussetts, 1995.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science, New York, 2006.

[4] P. Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, Cambridge, MA, 2001.

[5] K. I. Diamantaras and S. Y. Kung. *Principal Component Neural Networks: Theory and Applications*. Wiley-Interscience, New York, 1996.

[6] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification, Second Edition*. John Wiley and Sons, New York, 2001.

[7] E. Gardner. The space of interactions in neural network models. *Journal of Physics*, A21:257–271, 1946.

[8] D. George and J. Hawkins. Towards a mathematical theory of cortical micro-circuits. *PLoS Computational Biology*, 5-10:1–26, 2009.

[9] S. Grossberg. Towards a unified theory of neocortex: Laminar cortical circuits for vision and cognition. *Progress in Brain Research*, 165:79–104, 2007.

[10] S. Haykin. *Neural Networks and Learning Machine, Third Edition*. Pretice Hall, Upper Saddle River, New Jersey, 2009.

[11] R. Hecht-Nielsen. *Neurocomputing*. Addison-Wesley Publishing Company, New York, NY, 1990.

[12] R. Hecht-Nielsen. *Confabulation Theory*. Springer-Verlag, New York, NY, 2007.

[13] R. Hecht-Nielsen and T. McKenna. *Computational Models for Neuroscience*. Springer-Verlag, New York, NY, 2003.

[14] T. Kohonen. *Self-Organization and Associative Memory, second edition*. Springer-Verlag, New York, 1988.

[15] J. T.-H. Lo. Probabilistic associative memories. In *Proceedings of the 2008 International Joint Conference on Neural Networks*, pages 3895–3903. IEEE Xplore, The IEEE Press, 2008.

[16] K. A. C. Martin. Microcircuits in visual cortex. *Current Opinion in Neurobiology*, 12-4:418–425, 2002.

[17] K. Nagano. Association - a model of associative memory. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-2:68–70, 1972.

[18] E. Oja. A simplified neuron nodel as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273, 1982.

[19] E. Oja. Principal components, minor components, and linear neural networks. *Neural Networks*, 5:927–936, 1992.

[20] G. Palm. On associative memory. *Biological Cyberbentics*, 36:19–31, 1980.

[21] J. C. Principe, N. R. Euliano, and W. C. Lefebvre. *Neural and Adaptive Systems: Fundamentals through Simulations*. John Wiley and Sons, Inc., New York, 2000.

[22] T. D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 12:459–473, 1989.

[23] J. G. Sutherland. The holographic neural method. In S. Branko, editor, *Fuzzy, Holographic, and Parallel Intelligence*. John Wiley and Sons, 1992.

[24] S. Theodoridis. *Pattern Recognition, Second Edition*. Academic Press, New York, 2003.

[25] M. Turner and J. Austin. Matching performance of binary correlation matrix memories. *Neural Networks*, 1997.

[26] D. J. Willshaw, O. P. Buneman, and H. C. Longet-Higgins. Non-holographic associative memory. *Nature*, 222:960–962, 1969.

[27] D. J. Willshaw and H. C. Longet-Higgins. Associative memory models. In B. Meltzer and O. Michie, editors, *Machine Intelligence*. Edingburgh University Press, 1970.