OPEN ACCESS Freely available online

IJCMB
International Journal of Clinical Medicine and Bioengineering

ISSN 2737-534X
Volume 1, Issue 1
https://www.iikii.com.sg/journal/IJCMB
International Journal of Clinical Medicine and Bioengineering

Article

A Cortical Learning Machine for Learning Real-Valued and Ranked Data

James Ting-Ho Lo [1,*] and Bryce Mackey-Williams Carey [2]

[1] Department of Mathematics and Statistics, University of Maryland Baltimore County, USA; jameslo@umbc.edu
[2] Amazon Web Services, Amazon.com, USA; bcarey2@umbc.edu
* Correspondence: jameslo@umbc.edu; Tel.: 1-410-455-2432

Received: Aug 26, 2021; **Accepted:** Nov 30, 2021; **Published:** Dec 30, 2021

Abstract: The cortical learning machine (CLM) introduced in [1-3] is a low-order computational model of the neocortex. It has the real-time, photogragraphic, unsupervised, and hierarchical learning capabilities, which existing learning machines such as the multilayer perceptron and convolutional neural network do not have. The CLM is a network of processing units (PUs) each comprising novel computational models of dendrites (for encoding), synapses (for storing code covariance matrices), spiking/nonspiking somas (for evaluating empirical probabilities and generating spikes), and unsupervised/supervised Hebbian learning schemes. In this paper, the masking matrix in the CLM in [1-3] is generalized to enable the CLM to learn ranked and real-valued data in the form of the binary numbers and unary (thermometer) codes. The general masking matrix assigns weights to the bits in the binary and unary code to reflect their relative significances. Numerical examples are provided to illustrate that a single PU with the general masking matrix is a pattern recognizer with an efficacy comparable to those of leading statistical and machine learning methods, showing the potential of CLMs with multiple PUs especially in consideration of the aforementioned capabilities of the CLM.

Keywords: real-time learning; photographic learning; real-valued; ranked; masking matrix; neocortex; pattern recognition; associative memory; unsupervised learning, hierarchical learning

1. Introduction

The deep learning machines (DLMs) have been the most widely used type of learning method. As many applications as the they have, the existing DLMs such as the multilayer perceptron (MLPs) and convolutional neural networks (CNNs) lack the following learning capabilities, which if acquired in learning machines, are expected to greatly expand the range of applications of learning machines:

1. Recursive learning: By recursive learning, we mean that upon the arrival of a piece of new data, the old knowledge already learned is updated into new knowledge by learning only the piece of new data (without using the learned old data). Note that this is not the case with a deep learning machines (DLM) such as the multilayer perceptron and convolutional neural network, where the learned knowledge is stored as weights on the connections of the DLM. Whenever a piece of new data needs to be learned, the DLM trained on old data must learn the new data and all the old data jointly from scratch. If a sequence of new data keep coming in, the memory space of the DLM for storing the new knowledge as well as the old data, computational complexity of learning, and sometime the size of the DLM (i.e., the number of layers and number of connections) must be increased.
2. Unsupervised learning: Learning data without labels is called unsupervised learning. Labels are usually obtained by handcrafting, which is often costly or impossible for a large number of such images rapidly in real time. The unsupervised learning of the cortical learning machine (CLM) generates its own labels. When used consistently, these labels form a vocabulary or a complement of a common vocabulary (e.g., English, Chinese, etc.) for the CLM. Whenever the handcrafted labels are available, this self-generated vocabulary is translated into a common vocabulary by parts of the CLM called interpreters.
3. Photographic learning: By photographic learning, we mean learning (a) without waiting for the labels to be handcrafted, (2) without iteratively minimizing a training error criterion, (3) without cross validation on extra data (to avoid overfitting), and (4) without repeated learning sessions to pick the best training results (to avoid nonglobal local-minima and saddle points). We also mean that each piece of data can be learned and the knowledge in it can be used almost as soon as it arrives. Photographic learning is enabled mainly by the above recursive learning and unsupervised learning.

4. Hierarchical learning: Multiple whole or partial objects usually form a hierarchy in an image. Recognizing all the whole or partial objects in the hierarchy is required for humans to see the relative locations and relative relations of the objects. Because a DLM can learn an image only with a single label, to learn all the objects and partial objects in an image, each must be isolated from others in the image and learned separately. The CLM learns all the objects or parial objects together and each is assigned a label, handcrafted or self-generated, in their hierarchy in the image.
5. Real-time learning: Humans learn in real-time. The capabilities to perform the above recursive, unsupervised, photographic and hierarchical learning are prerequisites for real-time learning. In addition, the computation involved must be performed in real time. This requires the learning algorithm and the hardware implementing the algorithm to be effective to effect real-time learning.

Strictly speaking, biological plausibility of a learning machine does not guarantee it to be better than those without a resemblance to the biological learning system. However, recall that the McCulloch-Pitts model of the neuron and layered network structure in the neocortex led to the multilayer perceptron (MLP), and the hierarchical architecture and max pooling in the same are essential to the generalization capability of the convolutional neural network (CNN). The MLP and CNN outperform all the other learning machines such as Bayesian, decision tree, support vector machine, adaboosting, etc. that do not mimick the bio neural networks. This superiority of learning machines with biological features to those without suggests that more biological features of the brain might help develop future learning machines. After all, our brain (even the animal brain) is the only proven "technology" that generates true "intelligence". Many good learning machines with more features of the neocortex than the MLP and CNN can be found in literature, for example, [1-8].

A low-order model (LOM) of the biological neural networks in the neocortex was derived [1-3] from four neurobiological postulates: 1) most neurons communicate by spike trains; 2) neural networks learn by the Hebbian rule; 3) dendrites and axons perform computation; and 4) neural networks are hierarchical networks with feedback structures; and a creationist/evolutionary postulate: 5) functions and features of neural networks and their components are mathematically ideal. The LOM explains how biological neural networks (BNNs) encode, learn, memorize, recall and generalize, and thereby explains how the brain has the aforementioned desirable capabilities as a learning machine at the neuronal level [1-3].

The LOM is a hierarchical multilayer network of processing units (PUs), each comprising novel models of dendrites (for encoding), synapses (for storing code covariance matrices), spiking/nonspiking somas (for evaluating empirical probabilities and generating spikes), unsupervised/supervised Hebbian learning schemes, and a masking matrix to represent interneuron connections (maximal generalization). A PU is either an unsupervised PU (UPU) or a supervised PU (SPU) depending on whether the PU learns without or with supervision. Almost all of these component models (except that of the dendritic trees) are supported in the neurobiological literature. The LOM integrates these novel model components and expains how they encode, learn, memorize, recall and generalize for the first time in neuroscience literature.

When used as a learning machine, the LOM is called a cortical learning machine (CLM). As a model of the biological neural network in the neocortex which processes spikes, the CLM processes only binary data. Real-valued and ranked data must first be converted into a binary form such as the standard binary numbers and the unary codes to be processed by the CLM. While the standard binary number uses the smallest number of bits, the unary code keeps the Hamming distance between codes. Different positions of the bits in a binary number and a unary code carry different significance. In this paper, the masking matrix in the CLM [1-3] is generalized by assigning different weights to different positions of the bits for selecting different neuronal networkings of the CLM. This generalization greatly improves the accuracy rates of the CLM in recognizing real-valued and ranked data.

Three numerical examples are provided to illustrate how a CLM with a single PU with the generalized masking matrix works as a pattern recognizer for real-valued and ranked data. In each example, 10,000 CLMs are implemented by a 10-fold cross-validation procedure on the given dataset. The average, maximum and minimum accuracy rate are reported for the 10,000 CLMs with a single PU. Historical results by other pattern recognition methods following the 10-fold cross-validation procedure are used for comparison. Nevertheless, the number of times for each of the other methods is performed in the historical results is much less than 10,000. Besides, the accuracy rates of the learning methods are highly dependent on the hyperparameters selected for the learning methods. Therefore, these historical and our experimental results can only provide a rough comparison.

With this understanding, we can only say that the accuracy rates of the CLMs are comparable to those of leading learning methods on each of the iris classification, car evaluation, and congressional voting prediction datasets from the UCI Machine Learning Repository [12], showing the potential of the CLM with multiple PUs. It is stressed here that the CLM has the unique capabilities of real-time, photographic, unsupervises and hierarchical learning mentioned above.

The rest of the paper is organized as follows: In Section 2, the dendritic trees of the neurons in a PU is briefly explained. The dendritic trees encode the inputs to the PU into orthogonal binary codes for the neurons. In Section 3, the supervised and

unsupervised Hebbian learning rules with the dendritic codes are described. In Section 4, the general masking matrix is defined and some identities that help reducing the computation of the masking matrix are mentioned. In the same section, the functions of the synapses on the dentrites not masked are also stated. In Section 5, how the spiking and nonspiking somas transform the outputs from the synapses into estimates of the labels of the inputs to the PU is explained in more detail. In Section 6, the dendritic trees, learning rules, masking matrices, spiking/nonspiking somas are integrated into the unsupervised/supervised PU. In Section 7, a network of unsupervised PUs serving as a clusterer and a network of supervised PUs serving as an interpreter are organized to enable the CLM to learning without supervision all the time and to learn and generalize with supervision. In Section 8, the results of three numerical experiments are provided before a brief conclustion in Section 9.

2. Encoding inputs to neurons

Dendritic trees use more than 60% of the energy consumed by the brain, occupy more than 99% of the surface of some neurons, and are the largest component of neural tissue in volume. Yet, dendritic trees are missing in existing deep learning machines (DLMs), including the CNN, and associative memories, overlooking a large proportion of the neuronal circuit.

A key feature of CLM is a novel model of the dendritic encoder: A dendritic encoder that inputs $v_\tau = [v_{\tau 1} \quad v_{\tau 2} \quad v_{\tau 3}]'$ encodes it into the dendritic code $\breve{v}_\tau$ by the standard parity function ϕ as follows:

$$\breve{v}_\tau = [0 \quad v_{\tau 1} \quad v_{\tau 2} \quad \phi(v_{\tau 2}, v_{\tau 1}) \quad v_{\tau 3} \quad \phi(v_{\tau 3}, v_{\tau 1}) \quad \phi(v_{\tau 3}, v_{\tau 2}) \quad \phi(v_{\tau 3}, v_{\tau 2}, v_{\tau 1})]'$$

where ϕ is recursively defined by

$$\begin{aligned} \phi(v_{\tau 2}, v_{\tau 1}) &= -2v_{\tau 2}v_{\tau 1} + v_{\tau 2} + v_{\tau 1} && (1)\\ \phi(v_{\tau 3}, v_{\tau 2}, v_{\tau 1}) &= \phi\big(v_{\tau 3}, \phi(v_{\tau 2}, v_{\tau 1})\big) && (2)\\ &= -2v_{\tau 3}\phi(v_{\tau 2}, v_{\tau 1}) + v_{\tau 3} + \phi(v_{\tau 2}, v_{\tau 1}) && (3) \end{aligned}$$

A graph showing the dendritic encoder is given in Figure 1. For example, $[1 \quad 0 \quad 1]'$ and $[0 \quad 1 \quad 1]'$ are encoded into the codes $[0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0]'$ and $[0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0]'$ respectively.

Let u denote a scalar and $v = [v_1 \quad v_2 \quad \cdots \quad v_k]$ a k-dimensional vector. Define a k-dimensional vector $\phi(u, v)$ of polynomials by

$$\phi(u, v) = [\phi(u, v_1) \quad \phi(u, v_2) \quad \cdots \quad \phi(u, v_k)]$$

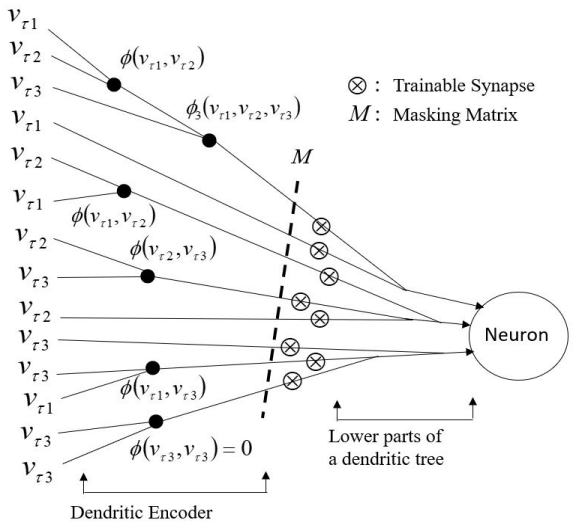

Figure 1. A dendritic encoder. The dendritic encoder is a dendritic tree that encodes its 3 binary inputs, v_1, v_2, v_3, into a 2^3-bit dendritic code $\breve{v}'(1,2,3)$ constructed with Equations (4)-(7) where m = 3. The entries of $\breve{v}'(1,2,3)$ are the inputs to the synapses.

The 2^m different functions that can be defined by compositions of the binary operation ϕ on the input set $\{v_1, v_2, ..., v_m\}$ are generated and organized into a 2^m-dimensional column vector $\breve{v}$ by recursively generating row vectors $\breve{v}(1, \cdots, k)$, for $k = 1, 2, ..., m$, as follows:

$$\begin{aligned} \breve{v}(1) &= [0 \quad v_1] && (4)\\ \breve{v}(1,2) &= \left[\breve{v}(1)\phi\left(v_2, \breve{v}(1)\right)\right] \\ &= [0 \quad v_1 \quad v_2 \quad -2v_2v_1 + v_2 + v_1] && (5)\\ \breve{v}(1, \cdots, k+1) &= \left[\breve{v}(1, \cdots, k+1)\phi\left(v_{k+1}, \breve{v}(1, \cdots, k)\right)\right] && (6)\\ \breve{v} &= \breve{v}'(1, \cdots, m) && (7) \end{aligned}$$

Denoting the k-th component of $\breve{v}$ by $\breve{v}_k$, the vector $\breve{v} = [\breve{v}_1 \quad \breve{v}_2 \quad \cdots \quad \breve{v}_{2^m}]'$ is called the dendritic expansion of v. It is proven in that given two m-dimensional binary vectors, v and u, their dendritic expansions, $\breve{v}$ and $\breve{u}$, satisfy

$$\begin{aligned} \left(\breve{v} - \tfrac{1}{2}\mathbf{I}\right)'\left(\breve{u} - \tfrac{1}{2}\mathbf{I}\right) &= 0, \text{ if } v \neq u && (8)\\ &= 2^{m-2}, \text{ if } v = u && (9) \end{aligned}$$

where $\mathbf{I} = [1 \quad 1 \quad \cdots \quad 1]'$ with $\dim \mathbf{I} = 2^m$. Note that $\mathbf{I}$ is not the identity matrix I.

3. Unsupervised and supervised learning rule and the resultant memory in synapses

The unsupervised covariance rule that updates the strength D_{ij} of the synapse receiving $\breve{v}_{tj}$ and feeding spiking neuron i whose output is u_{ti} follows:

$$D_{ij} \leftarrow \lambda D_{ij} + \Lambda(u_{ti} - \langle u_{ti}\rangle)\left(\breve{v}_{tj} - \langle\breve{v}_{tj}\rangle\right) \quad (10)$$

where Λ is a proportional constant, λ is a forgetting factor that is a positive number less than one, and $\langle\breve{v}_{tj}\rangle$ and $\langle u_{ti}\rangle$ denote, respectively, the average activities of the presynaptic dendritic node j and postsynaptic spiking neuron i over some suitable time intervals. The unsupervised covariance rule is shown in Figure 2. Note that this learning rule is actually a Hebbian-type learning rule [1-3].

Note that as u_{ti} and v_{ti} are stochastice neuron spikes with values, 1 and 0, with probabilities tending to approach ½, the averages $\langle\breve{v}_{tj}\rangle$ and $\langle u_{ti}\rangle$ approach ½, which makes $\left(\breve{v}_{tj} - \langle\breve{v}_{tj}\rangle\right)'\left(\breve{v}_{tj} - \langle\breve{v}_{tj}\rangle\right) = 0$. In many machine learning applications, the averages, $\langle\breve{v}_{tj}\rangle$ and $\langle u_{ti}\rangle$, over time are unnecessary and they are set equal to ½ to make use of the orthogonality property in (8) and (9). This is done in our experiments in Section 8.

The outputs u_{ti}, $i = 1, \ldots, R$, of the R spiking somas can be assembled into a vector, $u_t = [u_{t1} \quad u_{t2} \quad \cdots \quad u_{tR}]'$, and the strengths D_{ij} into a matrix D whose $i \times j$-th entry is D_{ij}. The vector u_t is the label of the input vector v_t that is selected in accordance with a probability distribution or membership function by the neurons for unsupervised learning . Such selection creates a vocabulary for the neurons themselves.

For supervised learning, the output u_{ti} from the spiking soma i in (10) is replaced with a component w_{ti} of the label of the input vector v_t provided from outside the CLM. Note that the label is that of the feature vector in the receptive field of soma i.

Soma i needs the output c_t of a nonspiking soma to generate its output u_{ti}. The synaptic strengths c_{ti} on the connections from the output terminals of a dendritic encoder to soma i form a row vector C and are updated by the following unsupervised accumulation rule, where λ and Λ are the forgetting and normalization factor respectively:

$$C \leftarrow \lambda C + \frac{\Lambda}{2}(\breve{v}_t - \langle\breve{v}_t\rangle)' \quad (11)$$

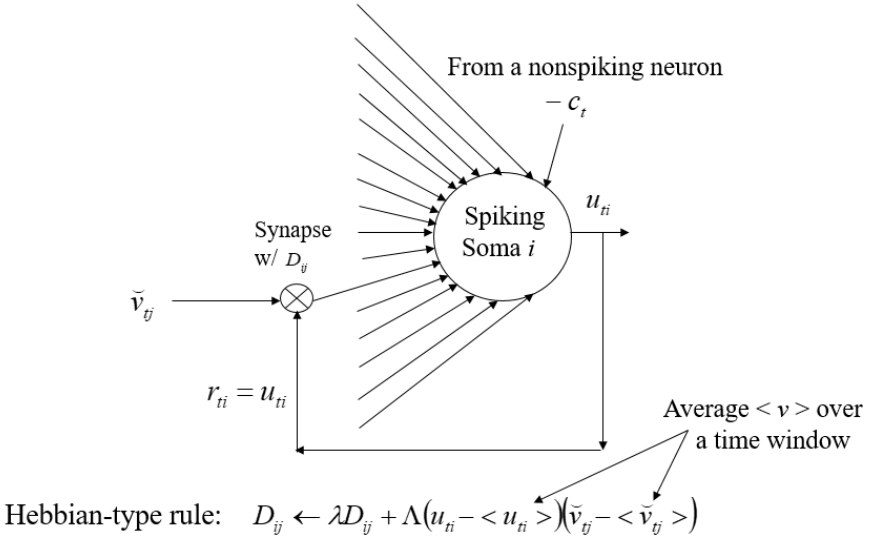

Figure 2. The unsupervised covariance learning rule. The output u_{ti} from soma i is learned jointly with the output $\breve{v}_{tj}$ from a synapse by the above Hebbian-type rule to update the synapse strength D_{ij}.

4. Masking Matrices for Binary, Real-Valued and Ranked Data

Let a vector v_τ that deviates from each of the vectors v_s, that have been learned by the synapses on a dendritic encoder due to corruption, distortion or occlusion, be presented to the dendritic encoder. The dendritic tree and its synapses are said to have a maximal generalization capability in their retrieval of information, if they are able to automatically find the largest subvector of v_τ that matches at least one subvector among the vectors v_s stored in the synapses and enable post-synaptic neurons to generate the empirical probability distribution of the label of the largest subvector. Such a maximal capability is achieved by the use of a masking matrix described in this section.

In [1, 3], the following orthogonal expansion of bipolar binary vectors $a = [a_1 \quad a_2 \quad \cdots \quad a_m]'$ was discovered:

$$\begin{aligned} \hat{a}(1) &= [1 \quad a_1]' \\ \hat{a}(1,\ldots,j+1) &= [\hat{a}'(1,\ldots,j) \quad a_{j+1}\hat{a}'(1,\ldots,j)]' \text{ for } j = 1,\ldots,m-1 \\ \hat{a} &= \hat{a}(1,\ldots,m) \end{aligned} \tag{12}$$

where $\hat{a}$ is called the orthogonal expansion of a. For example, if $a = [a_1 \quad a_2 \quad a_3]$, then

$$\hat{a} = [1 \quad a_1 \quad a_2 \quad a_2a_1 \quad a_3 \quad a_3a_1 \quad a_3a_2 \quad a_3a_2a_1]$$

Let us denote the vector $a = [a_1 \quad a_2 \quad \cdots \quad a_m]'$ with its i_1-th, i_2-th, ..., and i_j-th components set equal to 0 by $a(i_1^-, i_2^-, \ldots, i_j^-)$, where $1 \le i_1 \quad i_2 \quad \ldots \quad i_j \le m$, and the dendritic and orthogonal expansions of $a(i_1^-, i_2^-, \ldots, i_j^-)$ by $\breve{a}(i_1^-, i_2^-, \ldots, i_j^-)$ and $\hat{a}(i_1^-, i_2^-, \ldots, i_j^-)$, respectively. Denoting the m-dimensional vector $[1 \quad 1 \quad \cdots \quad 1]'$ by $\mathbf{I}$, the vector $\mathbf{I}$ with its i_1-th, i_2-th, ..., and i_j-th components set equal to 0 is $\mathbf{I}(i_1^-, i_2^-, \ldots, i_j^-)$ and the orthogonal expansion of $\mathbf{I}(i_1^-, i_2^-, \ldots, i_j^-)$ is denoted by $\hat{\mathbf{I}}(i_1^-, i_2^-, \ldots, i_j^-)$. For the vector $a = [a_1 \quad a_2 \quad \cdots \quad a_m]'$, $\breve{a}(i_1^-, i_2^-, \ldots, i_j^-) = \left(\text{diag}\hat{\mathbf{I}}(i_1^-, i_2^-, \ldots, i_j^-)\right)\breve{a}$. Notice that $\text{diag}\hat{\mathbf{I}}(i_1^-, i_2^-, \ldots, i_j^-)$ eliminates (i.e., masks) components in $\breve{a}$ that involve a_{i_1}, a_{i_2}, ..., a_{i_j}. Therefore, $\text{diag}\hat{\mathbf{I}}(i_1^-, i_2^-, \ldots, i_j^-)$ is called a masking matrix.

An important property of the masking matrix $\text{diag}\,\hat{\mathbf{I}}(i_1^-, i_2^-, \ldots, i_j^-)$ is the following: Assume that v_s and v_τ are binary vectors. If

$$\left(\text{diag}\,\hat{\mathbf{I}}(i_1^-, i_2^-, \ldots, i_j^-)\right)\left(\breve{v}_s - \frac{1}{2}\hat{\mathbf{I}}\right) = \left(\text{diag}\,\hat{\mathbf{I}}(i_1^-, i_2^-, \ldots, i_j^-)\right)\left(\breve{v}_\tau - \frac{1}{2}\hat{\mathbf{I}}\right)$$

then

$$(\breve{v}_s - \langle \breve{v}_s \rangle)' \left(\text{diag } \hat{\boldsymbol{I}}(i_1^-, i_2^-, \dots, i_j^-) \right) (\breve{v}_\tau - \langle \breve{v}_\tau \rangle) = 2^{m-2-j}. \tag{12}$$

If

$$\left(\text{diag } \hat{\mathbf{I}}(i_1^-, i_2^-, \dots, i_j^-) \right) \left(\breve{v}_s - \frac{1}{2}\hat{\mathbf{I}} \right) \neq \left(\text{diag } \hat{\mathbf{I}}(i_1^-, i_2^-, \dots, i_j^-) \right) \left(\breve{v}_\tau - \frac{1}{2}\hat{\mathbf{I}} \right)$$

then

$$(\breve{v}_s - \langle \breve{v}_s \rangle)' \left(\text{diag } \hat{\boldsymbol{I}}(i_1^-, i_2^-, \dots, i_j^-) \right) (\breve{v}_\tau - \langle \breve{v}_\tau \rangle) = 0$$

In the CLM in [1-3], the masking matrix is intended for learning a binary image, not like a binary number, whose bits are of the same significance. The label of a learned image stored in the synapses with the most bits equal to their corresponding bits in the input (or query) image should be recalled. For example, if a learned image is identical to the input image, the label of the learned image is recalled. If no such a perfect match, then the label of a learned image that matches the input image except one bit is recalled. If the learned images match the input image except at least j bits, then the labels of those learned images that matches the input image with exactly j bits should be recalled with confidence proportional to $2^{-j\eta}$ for some positive constant η. Therefore, the least number of bits in the stored images are masked to yield a match with the input image with the highest confidence. The number j of masked bits is called the masking level.

Because the masking level to be needed for an input image is unknown, we include masks of reasonable number of levels in the masking matrix, but use masking level weights to reduce the effects of masking unnecessary number of bits. Following this idea, we obtain the following masking matrix where $2^{-j\eta}$ are called the masking level weights for reducing the effects of masking level j and η is called the masking level parameter:

$$M = diag\, \hat{\boldsymbol{I}} + \sum_{j=1}^{J} 2^{-j\eta} \sum_{i_j=j}^{J} \cdots \sum_{i_2=2}^{i_3-1} \sum_{i_1=1}^{i_2-1} diag\, \hat{\boldsymbol{I}}(i_1^-, i_2^-, \cdots, i_j^-) \tag{13}$$

For real-valued data and ranked data to be processed by the dendrited trees, the real numbers and ranks are first converted into a binary vector such as the standard binary number and the standard unary code (or thermometer code), whose entries are naturally ordered. In a binary vector, different bits have different significances. In general, the higher bits (those in positions more to the left) are more signigicant than the lower bits. To take into consideration of such relative significances to enhance the generalization capability of the CLM (cortical learning machine), we introduce the position weights in addition to the level weight discussed above.

Let F real-valued or ranked features, $f = 1, \dots, F$, be represented by binary numbers or unary codes, which can be written as binary vectors, v_f, $f = 1, \dots, F$. We concatenate them into a vector $v = [v_1{}' \quad \cdots \quad v_F{}']'$, where the prime $'$ denotes the vector transposition. Let v_{fn} denotes the n-th entry in the m-th feature in v. Then, $v_f' = [v_{f1} \ \cdots \ v_{fdimv_f}]'$. For example, if $v_f = [v_{f1}\ v_{f2}\ v_{f3}]'$ for $f = 1, 2$, then $v = [v_{11}\ v_{12}\ v_{13}\ v_{21}\ v_{22}\ v_{23}]'$, v_{21} is the first entry of the second feature, and $\boldsymbol{I}(i_{21}^-) = [1\ 1\ 1\ 0\ 1\ 1]$.

Note that the entries in v_f have different significance. Assume that the vector v input to a PU. The masking matrix $M(f)$ for the feature vector v_f and the masking matrix M for the input vector v with both masking level weights $2^{-j\eta}$ and masking position weghts $2^{-\xi n}$ follows:

$$M(f) = diag\, \hat{\boldsymbol{I}} + 2^{-\eta} \sum_{i_{f1}=1}^{dim\, v_f - s_f} 2^{-\xi i_{f1}} diag \left(\hat{\boldsymbol{I}}(i_{f1}^-) \right) + 2^{-2\eta} \sum_{i_{f2}=2}^{dim\, v_f - s_f} \sum_{i_{f1}=1}^{i_{f2}-1} 2^{-\xi(i_{f1}+i_{f2})} diag \left(\hat{\boldsymbol{I}}(i_{f1}^-, i_{f2}^-) \right)$$

$$+\cdots+2^{-J_f\eta}\sum_{i_{fJ_f}=J_f}^{dim\,v_f-s_f}\cdots\sum_{i_{f2}=2}^{i_{f3}-1}\sum_{i_{f1}=1}^{i_{f2}-1}2^{-\xi\left(\sum_{k=1}^{J_f}i_{fk}\right)}diag\left(\hat{\boldsymbol{I}}\left(i_{f1}^{-},i_{f2}^{-},\cdots,i_{fj}^{-}\right)\right)$$

$$M(f)=diag\,\hat{\boldsymbol{I}}+\sum_{j=1}^{dim\,v_f-s_f}2^{-j\eta}\sum_{i_{fj}=j}^{dim\,v_f-s_f}\cdots\sum_{i_{f2}=2}^{i_{f3}-1}\sum_{i_{f1}=1}^{i_{f2}-1}2^{-\xi\left(\sum_{k=1}^{j}i_{fk}\right)}diag\left(\hat{\boldsymbol{I}}\left(i_{f1}^{-},i_{f2}^{-},\cdots,i_{fj}^{-}\right)\right) \tag{15}$$

$$M=\prod_{f=1}^{F}M(f)$$

where dim $\boldsymbol{I}\left(i_{f1}^{-},i_{f2}^{-},\cdots,i_{fj}^{-}\right)=\sum_{f=1}^{F}dim\,v_f$, η is called the masking level parameter, ξ is called the masking position parameter, and the number of the left-most bits (or positions) not to be masked is denoted by s_f. For example, changing the 2 left-most bits of a binary number changes greatly the value of the binary number. To avoid the generalization from such a different binary number, we can set $s_f=2$. The non-negative real-valued ξ is called the masking position parameter. In most application involving ranked data and real-valued data, the level weights are unneccessary and hence $\eta=0$.

Equation (15) shows the idea of using masking level and position weights. The masking level weights and masking position weights in (14) can deviate from those specified in (15), depending on the application, to achieve the best accuracy of the CLM. We may start with (15) and then use the trial and error method of tweaking the masking level and position weights to maximize the accuracy of the CLM. For example, in the Iris classification example in Subsection 8.2, the optimal masking position weights of the two right most bits of the binary numbers turned out to be equal to one.

If the input vector or input matrix represents an image whose entries are not ordered, then position weights are unnecessary. In this case, by setting $\xi=0$, we obtain a general formula of the masking matrix for input vectors with separate feature vectors, which are masked separately with level weights. Dividing the input image into subimages with separate masking matrices allows us to have the masked bits more evenly distributed and avoid a large number of masked bits to concentrate in a small area on the imput image.

Note that $diag\left(\hat{\boldsymbol{I}}\left(i_1^{-},i_2^{-},\cdots,i_j^{-}\right)\right)=\prod_{k=1}^{j}diag\left(\hat{\boldsymbol{I}}(i_k^{-})\right)$, which helps reducing the computation of the masking matrix. Upon the arrival of $\breve{v}_\tau$ at time τ, the synapses in Fig. 2 compute the following products, d_τ *and* c_τ.

$$d_\tau=DM(\breve{v}_\tau-\langle\breve{v}_\tau\rangle) \tag{16}$$

$$c_\tau=CM(\breve{v}_\tau-\langle\breve{v}_\tau\rangle) \tag{17}$$

which are then input to and processed by nonspiking and spiking somas in Fig. 3 to produce the probabilities for spiking somas to generate a pulse v_τ. The retrieving rules (16) and (17) are variants of the retrieving rules of the associative memory [9-11].

5. Estimation of Labels by Somas

Once a vector v_τ is received and encoded by a dendritic encoder into $\breve{v}_\tau$, it is made available to synapses for learning as well as retrieving of the information about the label of the input v_τ. In response to $\breve{v}_\tau$, the masking matrix M computes $M_{jj}\left(\breve{v}_{\tau j}-\left\langle\breve{v}_{\tau j}\right\rangle\right)$ for all j.

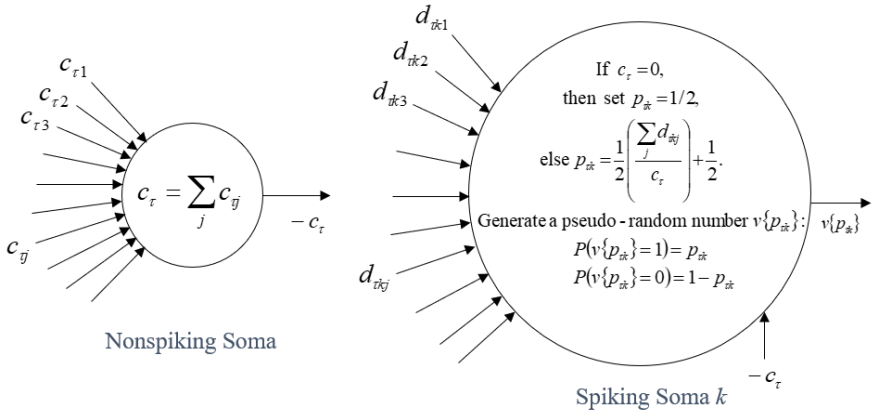

Figure 3. Nonspiking soma and spiking soma k. The former simply add up its inputs received from its preceding synapses. The latter computes the empirical probability $p_{\tau j}$ and generate a spike with the probability.

The computations in the nonspiking and a spiking soma in a PU to generate an estimate of a component of the l abel input to the PU are shown in Figure 3. Synapse j for a nonspiking soma then computes $c_{\tau j} = C_j M_{jj}\left(\breve{v}_{\tau j} - \langle \breve{v}_{\tau j} \rangle\right)$, for all j, where M_{jj} is the j-th diagonal entry of M. The model nonspiking soma sums up $c_{\tau j}$ to obtain the graded sign al c_τ. Note that the synaptic weight vector C is defined in (11). Because of the orthogonality property of $\breve{v}_\tau$, t = 1, ..., T; c_τ is an estimate of the total number of times $\breve{v}_\tau$ has been encoded and stored in C. The inhibitory output $-c_\tau$is a graded signal transmitted to each of the mentioned R spiking neurons that generate a point estimate of the label r_t of v_τ. The entries of the jth row D_j of D are the weights or strengths of the synapses for the jth spiking neuron. I n response to $\breve{v}_\tau$ produced by the dendritic encoders, the masking matrix M and synapses for the jth spiking neuron co mpute $M_{jj}\left(\breve{v}_{\tau j} - \langle \breve{v}_{\tau j} \rangle\right)$ and $d_{\tau kj} = D_{kj} M_{jj}\left(\breve{v}_{\tau j} - \langle \breve{v}_{\tau j} \rangle\right)$, respectively. The jth spiking neuron (a model spiking neuron) s ums up $d_{\tau kj}$ to obtain the graded signal $d_{\tau k} = \sum_j d_{\tau kj}$.

Because of the orthogonality property of $\breve{v}_\tau$; $d_{\tau k}$ is an estimate of the total number of times v_τ has been encoded and stored in D_k with the kth component $r_{\tau k}$ of r_τ being 1 minus the total number of times v_τ has been encoded and stored in D_k with the kth component $r_{\tau k}$ of r_τ being 0. The effects of M, λ and Λ are included in computing said total numbers, which make $d_{\tau k}$ only an estimate.

Recall that c_τ is an estimate of the total number of times v_τ has been learned regardless of its labels. Therefore, $(c_\tau + d_{\tau k})/2$ is an estimate of the total number of times v_τ has been encoded and stored with the kth component $r_{\tau k}$ of r_τ being 1. Consequently, $(d_{\tau k}/c_\tau + 1)/2$ is the empirical probability $p_{\tau k}$ that $r_{\tau k}$ is equal to 1. The kth spiking neuron then uses a pseudo-random generator to generate 1 with probability $p_{\tau j}$ and 0 with probability $1 - p_{\tau j}$. This 1 or 0 is the output of the kth spiking neuron. Biological justification of the model nonspiking and spiking somas are provided in [10].

Note that the vector $p_\tau = [p_{\tau 1} \quad p_{\tau 2} \quad \cdots \quad p_{\tau R}]'$ is a representation of a empirical probability distribution of the label r_τ of the input vector v_τ. A pseudorandom ternary number generator in the jth spiking neuron uses $p_{\tau j}$ to generate an output denoted by $v\{p_{\tau k}\}$ as follows: $v\{p_{\tau k}\}$ = 1 with probability $p_{\tau k}$, and $v\{p_{\tau k}\} = -1$ with probability $1 - p_{\tau k}$. Note also that the outputs of the R spiking neurons in response to v_τ form a binary vector $v\{p_\tau\}$, which is a point estimate of the label r_τ of v_τ.

6. Processing Units (PUs)

CLM organizes a biological neural network into a recurrent network of PUs (processing units). A schematic diagram of a PU is shown in Figure 4 below, which shows how dendritic encoders, synapses, a nonspiking soma, R spiking somas, and learning and retrieving mechanisms are integrated into a processing unit (PU). The vector v_τ input to a PU is first expanded by dendritic encoders into a dendritic code $\breve{v}_\tau$. $\breve{v}_\tau$ is used to compute $c_{\tau j}$ and $d_{\tau kj}$ using the synaptic weights C_j and D_{kj} respectively.

The nonspiking soma in the PU computes the sum $\sum_j c_{\tau j}$, and the kth spiking soma computes $\sum_j d_{\tau kj}$ and $p_{\tau k} = \left(\sum_j d_{\tau kj} / \sum_j c_{\tau j} + 1\right)/2$, which is the relative frequency that the kth digit of the label of v_τ is +1. By a pseudo-random generator, the kth spiking soma outputs $v\{p_{\tau k}\}$, which is +1 with probability $p_{\tau k}$ and is -1 with probability $1 - p_{\tau k}$, for k = 1, ..., R. $v\{p_\tau\}$

$= [v\{p_{\tau 1}\} \quad \cdots \quad v\{p_{\tau R}\}]'$ is a point estimate of the label of v_τ. Note that use of weights in the masking matrices can facilitate max-pooling of dendritic encoders in the computation of $p_{\tau k}$.

The lever in a hand-drawn circle indicates that the handcrafted r_τ is used for supervised learning, and the PU is a supervised PU (SPU). If the lever is placed at the short dashed line, then the estimated label $v\{p_\tau\}$ is used for unsupervised learning. In this case, the PU is a unsupervised PU (UPU). UPUs in the lowest layer cluster and recognize the lowest level of pattern elements such as variants of a hyphen, a pipe, a slash, a back slash, and so on. These pattern elements are integrated from layer to layer into larger and larger pattern elements and patterns. As long as a vector v is input to the SPU or UPU, it learns with or without supervision.

The nonspiking soma in the PU computes the sum $\sum_j c_{\tau j}$, and the kth spiking soma computes $\sum_j d_{\tau kj}$ and $p_{\tau k} = \left(\sum_j d_{\tau kj} / \sum_j c_{\tau j} + 1\right)/2$, which is the relative frequency that the kth digit of the label of v_τ is +1. By a pseudo-random generator, the kth spiking soma outputs $v\{p_{\tau k}\}$, which is +1 with probability $p_{\tau k}$ and is -1 with probability $1 - p_{\tau k}$, for $k = 1, ..., R$. $v\{p_\tau\}$ $= [v\{p_{\tau 1}\} \quad \cdots \quad v\{p_{\tau R}\}]'$ is a point estimate of the label of v_τ. Note that use of weights in the masking matrices can facilitate max-pooling of dendritic encoders in the computation of $p_{\tau k}$.

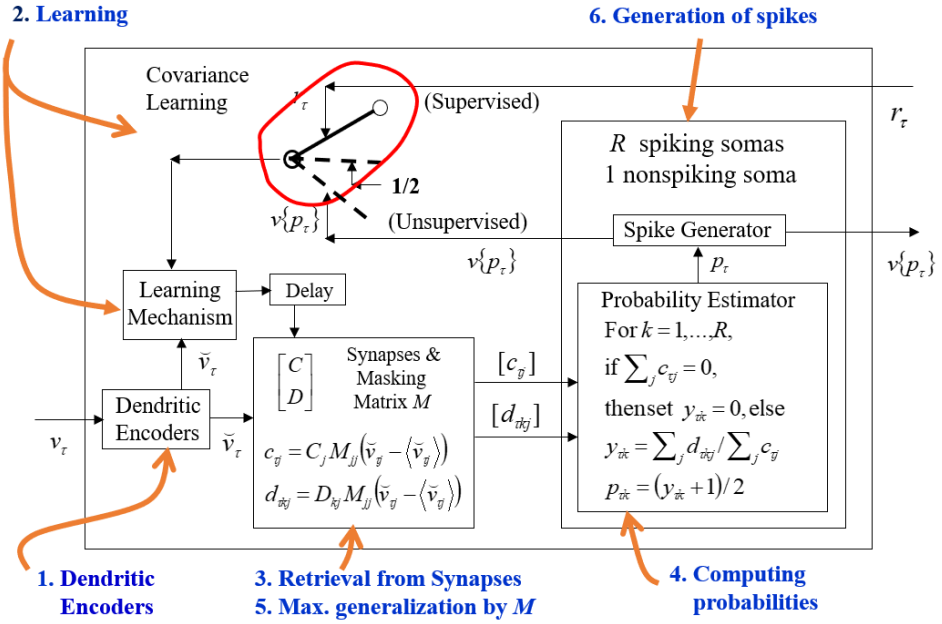

Figure 4. Supervised Processing Unit (SPU). An SPU comprises dendriteic encoders, synapses with strength *C/D*, a masking matrix *M*, *R* spiking somas, 1 nonspiking soma, and the supervised covariance learning mechanism. Their coordinated functions produce R empirical probabilities $p_{\tau k}$, which are used to generate *R* spike trains $v\{p_\tau\}$.

The nonspiking soma in the PU computes the sum $\sum_j c_{\tau j}$, and the kth spiking soma computes $\sum_j d_{\tau kj}$ and $p_{\tau k} = \left(\sum_j d_{\tau kj} / \sum_j c_{\tau j} + 1\right)/2$, which is the relative frequency that the kth digit of the label of v_τ is +1. By a pseudo-random generator, the kth spiking soma outputs $v\{p_{\tau k}\}$, which is +1 with probability $p_{\tau k}$ and is -1 with probability $1 - p_{\tau k}$, for $k = 1, ..., R$. $v\{p_\tau\}$ $= [v\{p_{\tau 1}\} \quad \cdots \quad v\{p_{\tau R}\}]'$ is a point estimate of the label of v_τ. Note that use of weights in the masking matrices can facilitate max-pooling of dendritic encoders in the computation of $p_{\tau k}$.

The green lever circled with the red solid line indicates that the estimated label $v\{p_\tau\}$ is used for unsupervised learning. In this case, the PU is a unsupervised PU (UPU). If the green lever is placed in the position circled with the blue dashed line, then the handcrafted r_τ is used for supervised learning, and the PU is a supervised PU (SPU). UPUs in the lowest layer cluster and recognize the lowest level of pattern elements such as variants of a hyphen, a pipe, a slash, a back slash, and so on. These pattern elements are integrated from layer to layer into larger and larger pattern elements and patterns. As long as inputs are provided to an UPU by sensors or other parts of the CLM, the UPU learns (without supervision).

By the maximal/adjustable generalization capability (or more specifically, masking matrix *M*), each UPU acts as a cluster of its input vectors v_τ. If v_τ or a close version has not been learned by an UPU, The UPU generates the label of v_τ at random. This enables the UPU to act as a pattern recognizer by itself. Whenever a handcrafted label r_τ is available to an SPU, the SPU learns its input vector v_τ with r_τ. By the maximal/adjustable generalization capability, the entire cluster(s) constructed by the UPU(s) that provide v_τ is assigned the same label r_τ. This minimizes the amount of handcrafted labels required.

7. Clustering and Interpreting

The version of CLM proposed herein consists of a hierarchical network of UPUs (unsupervised processing units) with feedbacks connections, acting as pattern recognizers, and a number of offshoot SPUs (supervised processing units), translating the self-generated labels from UPUs into human language, which are called the clusterer and interpreter, respectively. An example clusterer in its entirety for clustering spatial and temporal data is shown in Figure 5. The feedback connections in the clusterer have delay devices of different durations make CLM suitable for recognizing temporal patterns in video and movie. However, they will not be used in the proposed project.

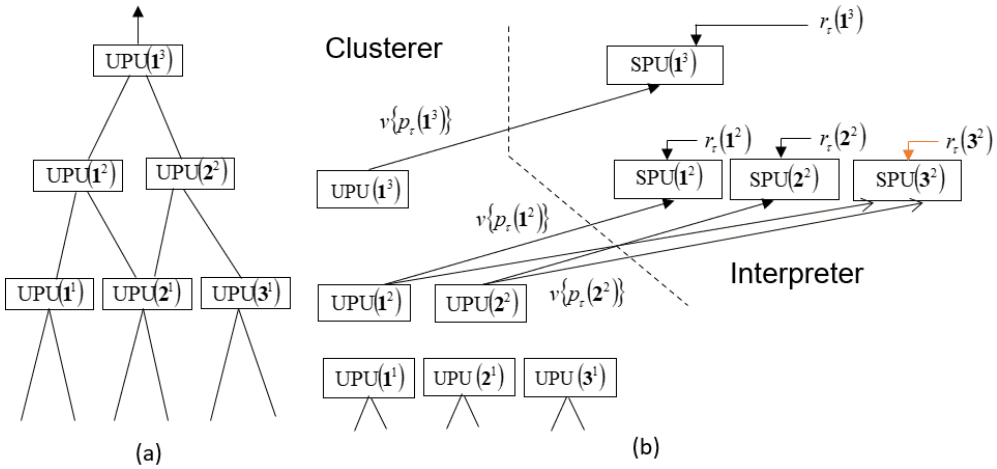

Figure 5. (a) A network of UPUs called a Clusterer and (b) a network of SPUs called an Interpreter. The clusterer performs unsupervised learning and the interpreter supervised learning to translate a self-generated vocabulary into a common vocabulary.

Once an exogenous feature vector is input to the clusterer, the UPUs perform retrieving and/or learning from layer to layer starting with layer 1, the lowest layer. After the UPUs in the highest layer complete performing their functions, the clusterer is said to have completed one round of retrievings and/or learnings (or memory adjustments). For each exogenous feature vector, the clusterer will continue to complete a certain number of rounds of retrievings and/or learnings.

The clusterer in Figure 5(a) is also shown in Figure 5(b) with the connections and delay devices removed. The three UPUs in the lowest layer of the clusterer do not branch out, but each of the three UPUs in the second and third layers branches out to an SPU. UPU($\mathbf{1}^2$) and UPU($\mathbf{2}^2$) in the second layer have feedforward connections to SPU($\mathbf{1}^2$) and SPU($\mathbf{2}^2$) respectively, and UPU($\mathbf{1}^3$) in the third layer has feedforward connections to SPU($\mathbf{1}^3$). The labels, $r_\tau(\mathbf{1}^2)$, $r_\tau(\mathbf{2}^2)$ and $r_\tau(\mathbf{1}^3)$, which are used for supervised learning are provided by the human trainer of the CLM.

8. Three Numerical Experiments on Real-Valued and Ranked Data

The CLM is tested on the iris classification, car evaluation, and congressional voting dataset from the UCI Machine Learning Repository [12]. The continuous-valued data and rank data are converted into the standard binary numbers or unary codes. The trainings and testings of 10,000 CLMs with a single processing unit (PU) were done on 10-fold cross validation datasets from each of the three datasets on a UMBC parallel computer version with 86 nodes.

The numerical test confirmed the feasibility and excellent performanc of only a single PU of the CLM trained by its supervised learning scheme. However, to confirm the unsupervised learning scheme of CLM, which is usually used for a larger dataset, and to get CLM accepted so as to contribute to the progress of machine learning, this proposal is seeking funding to conduct numerical experiments for CLM on large and popular datasets for direct comparison with CNN.

8.1. Experimental Procedure

Each of the three datasets is used as follows: The standard 10-fold cross-validation tesing is performed to test the CLM on the dataset used: The dataset is randomly partitioned into 10 subsets. One of them is used as the test dataset, and the other nine subsets are combined into the training dataset. An CLM is trained on the training dataset, tested on the test dataset, and the classification

rate (i.e., rate of correct classification) is recorded. Repeating this with each of the 10 data subsets as the test dataset, we obtain 10 classification rates.

We repeatedly partition the dataset into 10 subsets 1,000 times to get 1,000 partitionings of the dataset, and repeatedly performing the standard 10-fold cross-validation testing 1000 times to get 10,000 classification rates altogether. The average and variance of the 10,000 classification rates are computed and reported.

8.2. Iris classification

The UCI Iris Classification dataset [12] is concerned with using four real-valued attributes of an iris flower: sepal length in cm; sepal width in cm; petal length in cm; and petal width in cm to classify the flower into 3 classes: iris setosa; iris versicolour; and iris virginica. 150 instances are given.

We rank the different values of 150 measurements of each attribute, find the maximum and minimum of them, and determine the number of bits required to represent the different values by a binary number for the attribute. It turns out that we need to assign 6-bit binary numbers to the values of the sepal length and petal length and 5-bit binary numbers to the values of the sepal width and petal width values. Thus there are four attribute binary numbers with a total of 22 bits forming the input to the PU of the CLM. Using the 10-fold cross-validation testing procedure described in Subsection 8.1, 10,000 CLMs were implemented each to learn a data subset of $150 \times (9/10)$ instances and to be tested on a data subset of $150 \times (1/10)$ instances for classifying the four attributes presented to it into a type of iris. Note that the training data and the test data for each CLM do not share any instance.

As discussed in Section 4, Equation (15) is intended to show the idea of using masking position weights and its variants should be considered in selecting one for the application. A selected variant of (15) with more flexibility follows:

$$M(f) = \text{Diag}\,\hat{\mathbf{I}} + \sum_{j=1}^{J_f} 2^{-j\eta} \sum_{i_{fj}=j}^{\dim v_f - s_f} \cdots \sum_{i_{f2}=2}^{i_{f3}-1} \sum_{i_{f1}=1}^{i_{f2}-1} 2^{-\xi\left(\sum_{k=1}^{j} r_{fk}\right)} \text{diag}\left(\hat{\mathbf{I}}\left(i_{f1}^{-}, i_{f2}^{-}, \cdots, i_{fj}^{-}\right)\right) \tag{18}$$

where $r_{fk} = 0$ if $k \le m$ and $r_{fk} = i_{fk}$ if $k > m$ for a selected integer m. This variant sets the position weights of the m least significant bits to be 0, allowing more freedom to mask these least significant bits than to mask the bits with more significance.

For the iris classification dataset, $v_f =$ sepal length, sepal width, petal length, and petal width in binary form, respectively; $\dim v_f = 6, 5, 6, 5$ for $f = 1, 2, 3, 4$, respectively; $J_f = 6, 5, 6, 5$; and $s_f = 0$. It is found by trial and error that the best parameters are $\eta = 0$, $\xi = 5$, and $m = 2$ in (18).

According to the procedure described in Subsection 8.1, the average of the 10,000 classification rates are 96.15% and the maximum and minimum are 98.00% and 91.33%, respectively.

Although historical test results by the procedure described in Subsection 8.1 applied 10,000 times to another prediction method cannot be found, some results on other methods provide an approximate comparison. The Waikato Environment for Knowledge Analysis (WEKA) implementations of the J48 decision tree, Naive Bayes, and multilayer perceptron are reported to achieve 10-fold cross-validation accuracy rates of 96%, 96%, and 97:33%, respectively [4]. The C4.5 decision tree is reported to achieve 10-fold cross-validation accuracy ratings of 94.33% and 94.67% with boosting and bagging, respectively [11]. There are other accuracy rates obtained by RBF and SVM. All such historical results are comparable to mean accuracy rate of 96.15% of a single PU (processing unit) in the CLM (cortical learning machine). Such performance of a single PU shows the potential of a CLM that is a multilayer network of PUs as a learning machine. Recall that the CLM has the capabilities of real-time, photographic, unsupervised and hierarchical learning discussed in Section 1.

8.3. Car Evaluation

The UCI Car Evalustion dataset [12] is concerned with using six categorical attributes: buying price; maintenance cost; number of doors; number of passengers; luggage boot; and safety to classify the car into four demands: unacceptable; acceptable; good; very good. 1,728 instances are given.

Each of the six attributes has 4 categories. We use 3-bit unary codes (i.e., thermometer codes) {000, 001, 011, 111} for the buying price; maintenance cost; and number of doors, and 2-bit unary codes {00, 01, 11} for the number of passengers; luggage boot; and safety. Altogether 15 bits are input to the CLM. The four demands are represented by four 1-hot binary vectors that are output from the PU.

For the car evaluation dataset, we set $v_f =$ buying price; maintenance cost; number of doors; number of passengers; luggage boot; and safety in unary codes, respectively; $\dim v_f = 3, 3, 3, 2, 2, 2$ for $f = 1, 2, 3, 4, 5, 6$, respectively; $J_f = 3, 3, 3, 2, 2, 2$; and $s_f = 0$. It is found by trial and error that the best parameters are $\eta = 0$, $\xi = 30$ in (15).

Following the procedure described in the Subsection 8.1, 10,000 CLMs were implemented each to learn a data subset of $1{,}720\times(9/10)$ instances and to be tested on a data subset of $1{,}720\times(1/10)$ + 8 instances for classifying the six attributes presented to it into one of the four classes. The average, the maximum, and minimum of the accuracy rates of the 10,000 implemented PUs are 96.70%, 97.6273%, and 95.54%, respectively.

Although historical test results by the procedure described in Subsection 8.1 applied 10,000 times to another prediction method cannot be found, some results provide an approximate comparison. Several well-known learning algorithms have been applied to the Car UCI Evaluation dataset and reported in other publications for comparison, analysis, and development. Using 10 times repeated 10-fold cross-validation, the C4.5 and C5 decision tree algorithms are reported to achieve accuracy ratings of $92.2 \pm 2.2\%$ and $92.2 \pm 2.1\%$, respectively. The Waikato Environment for Knowledge Analysis (WEKA) implementation of the Naive Bayes classifier is reported to achieve 85.706% 10-fold cross-validation accuracy. The same publication also reports a 10-fold cross-validation accuracy rating of 99:537% using the WEKA implementation of the multilayer perceptron.

It is evident that the PU outperforms the aforementioned historical results pertaining to the C4.5 and C5 decision trees and the Naive Bayes classifier, but is outperformed by the multilayer perceptron. We believe given more time to adjust the masking position weights or perhaps use the binary numbers, the difference can be reduced or even eliminated.

8.3. Congressional Voting Prediction

The UCI Congressional Voting Records data set [12] consists of binary voting histories of elected representatives in the 98th US congress on key issues identfied by the Congressional Quarterly Almanac. Of the 435 data instances, there are 267 democrats and 168 republicans. There are 16 features, each of which identifies whether the corresponding representative voted yay or nay on the respective bill. Although the features are binary, there is technically a third category denoting an unknown position due to abstention. The voting histories are used to predict the party membership of each representative, whether they be democrat or republican.

The 16 features and their respective categorical values, yay and nay, are represented 1 and 0, respectively. The unknown is represented by ½ to make use of the orthogonal property in (8) and (9). Let **I** denote the 16-bit vector whose entries are all equal to 1. If we want to mask the *f*-th feature, Then

$$\mathrm{M(f)} = \mathrm{diag}\,\hat{\mathbf{I}} + 2^{-\xi \mathrm{i_{f1}}}\mathrm{diag}\left(\hat{\mathbf{I}}(\mathrm{i}_{\mathrm{f1}}^{-})\right)$$

$$M = \prod_{f=1}^{16} M(f) = \prod_{f=1}^{16}\left[diag\,\hat{\boldsymbol{I}} + 2^{-\xi i_{f1}} diag\left(\hat{\boldsymbol{I}}(i_{f1}^{-})\right)\right] \tag{19}$$

An alternative way to construct a masking matrix *M* is to treat the 16 features as a single feature with 16 components of the same significance, like those in an image. Then we have

$$M = diag\,\hat{\boldsymbol{I}} + \sum_{j=1}^{16} 2^{-j\eta} \sum_{i_j=j}^{16} \cdots \sum_{i_2=2}^{i_3-1} \sum_{i_1=1}^{i_2-1} diag\left(\hat{\boldsymbol{I}}(i_1^{-}, i_2^{-}, \cdots, i_j^{-})\right)$$

which is equal to (19) if the masking level parameter η is equal to the masking position parameter ξ.

The masking position or level weight parameter η is set equal to 4 above. Using the 10-fold cross-validation testing procedure described inSubsection 8.1, 10,000 CLMs were implemented. The average, maximum and minimum of their 10,000 accuracies are respectively equal to 93.33%, 94.4828% and 91.95%.

Although historical test results by the procedure described in Subsection 8.1 applied 10,000 times to another prediction method cannot be found, some results provide an approximate comparison. With 10 times repeated 10-fold cross validation, the WEKA implementations of the J48 decision tree and Naive Bayes algorithms are reported to achieve mean accuracies of 96:46±0:17% and 90:18±0:07%, respectively [1]. The default WEKA version 3.8.1 implementation of the multilayer perceptron achieves a 10-fold cross-validation accuracy of 94:7126%, configured with 16 input nodes, 9 hidden nodes, and 2 output nodes.

9. Conclusion

The CLM (cortical learning machine) reported herein has the capabilities of real-time, photographic, unsupervised a nd hierarchical learning, which are highly desirable of learning machines for applications and AI. After all, humans and even animals learn with such capabilities. Being a low-order computational model of the biological neural networks, th e CLM is a hierarchical multilayer network of processing units (PUs) each comprising novel models of dendrites (for e ncoding), synapses (for storing code covariance matrices), spiking/nonspiking somas (for evaluating empirical probabilitie s and generating spikes), unsupervised/supervised Hebbian learning schemes, and a masking matrix of the interneuron co nnection (for generalization).

Numerical experiments of a single PU of the CLM on small benchmark datasets result in performances comparable to those of leading learning algorithms, showing the promises of the CLM as a learning machine for real-world applications and as a true model of the biological neural networks for answering the holy grail questions on how the brain encodes, learns, memorizes, recalls and generalizes.

Author Contributions: Contributions are listed as follows: conceptualization, J.L.; methodology, J.L. and B.C.; software, B.C.; validation, B.C.; formal analysis, J.L. and B.C.; investigation, J.L. and B.C.; writing—original draft preparation, J.L. and B.C.; writing—review and editing, J.L.; visualization, J.L.

Funding: The work was supported in part by the U.S.A. National Science Foundation under Grant ECCS1028048 and Grant ECCS1508880, but does not necessarily reflect the position or policy of the U.S.A. Government..

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lo, J. T.-H. Functional model of biological neural networks. *Cognitive Neurodynamics* **2010**, *4*(4):295–313.
2. Lo, J. T.-H. A low-order model of biological neural networks. *Neural Computation* **2011**, *23*(10):2626–2682.
3. Lo, J. T.-H. A cortex-like learning machine for temporal hierarchical pattern clustering, detection, and recognition. *Neurocomputing* **2012**, *78*:89–103.
4. Fox, C. A.; Barnard, J. W. A quantitative study of the purkinje cell dendritic branches and their relationship to afferent fibers. *Journal of Anatomy* **1957**, *91*:299–313.
5. George, D.; Hawkins, J. Towards a mathematical theory of cortical micro-circuits. *PLoS Computational Biology* **2009**, *5*(10):1–26.
6. Granger, R. Engines of the brain. The computational instruction set of human cognition. *AI Magazine* **2006**, *27*:15–31
7. Grossberg, S. Towards a unified theory of neocortex 2007, Laminar cortical circuits for vision and cognition. *Progress in Brain Research* **2007**, *165*:79–104.
8. Martin, K. A. C. Microcircuits in visual cortex. *Current Opinion in Neurobiology* **2002**, *12*(4): 418–42.
9. Hassoun, M. H. Associative Neural Memories. *Theory and Implementation*, Oxford University Press, New York, New York, 1993.
10. Hinton, G. E.; Anderson, J. A. *Parallel Models of Associative Memory*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989.
11. Kohonen, T. *Self-Organization and Associative Memory*, Springer Verlag, New York, New York, 1988.
12. UC Irvine Machine Learning Repository. Available online: https://archive.ics.uci.edu/ml/index.php.