Temporally Local Unsupervised Learning: The MaxIn Algorithm for Maximizing Input Information

Randall C. O'Reilly
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213 USA
`oreilly@cmu.edu`

There are many appealing aspects of self-organizing learning rules, among them the notion that they are more "biologically plausible" than supervised learning algorithms. This plausibility usually derives from the ability to compute the algorithm with variables available locally to the unit or "neuron", typically using some variant of a Hebbian learning rule. Ironically, however, the locality *in time* of the variables that determine the learning is often ignored. This temporal non-locality presents a problem both from a biological and a psychological standpoint. In this paper, I present an alternative objective function for self-organizing algorithms that is local in both space and time, and results in a simple learning rule that can be implemented with properties of neuronal synaptic modification.

The issue of temporal non-locality takes several forms, some of which can be subtle. However, the central problem that lies behind the use of variables that are non-local in time is that they depend upon seeing a uniform random sample of the environment over the time period during which learning is to take place. While it is almost always possible to engineer the training set for a simulation so that the uniformity condition holds, the real world might not be so obliging. To instantiate the problem more fully, imagine the task of classifying different types of trees. In the real world, the kinds of trees one sees across different climates are not very uniformly distributed. Each climate zone tends to have associated with it a different set of trees, from palm trees to pine trees. Thus, if one were to spend several month's vacation in the tropics, where palm trees abound, the units representing these kinds of trees would be firing quite often, and those for the coniferous trees would be silent. Then, upon traveling to the Rocky Mountains for the remainder of the year, the reverse would occur. If one imagines that the entire space of trees is to be represented, then the appropriate time period to sample variables over is an entire year. However, this is clearly too long to wait to learn something new. With shorter time samples, the clustering of category instances over time will bias estimates of the mean and variance of unit firing rates, which might compromise the learning algorithm.

In its most obvious form, temporal non-locality is evident in the variables appearing directly in the learning rule. For example, the most general (and optimal with respect to the signal-to-noise ratio under certain conditions (Gaussian signal and noise, etc.) [16]) formulation of the Hebb rule is in terms of the covariance between pre and postsynaptic unit activities x_i and y_j:

$$\Delta w_{ij} \propto (x_i - \bar{x_i})(y_j - \bar{y_j}) \tag{1}$$

This rule requires the average unit activities $(\bar{x_i}, \bar{y_j})$, which in general must be estimated by measuring the activities of the units over time. This is clearly a violation of temporal locality, and it would suffer from the kind of temporal clustering described in the tree-classification example. A common solution to this problem is to assume that the means are constant and known in advance. This simply replaces the problem of temporal non-locality with that of introducing *a priori* assumptions about the environment and unit activities.

A more subtle example of temporal non-locality is that of Oja's learning rule [11], which extracts the first principle component of the input distribution:

$$\Delta w_{ij} \propto y_j(x_i - y_j w_{ij}) \tag{2}$$

This appears to be completely local in both space and time. However, Linsker's analysis [8] of this learning rule in terms of the Infomax principle of this learning rule shows that it is approximately maximizing the output variance. Thus, the objective function that this algorithm is optimizing is non-local in time, since the variance of the output signal can only be defined over time. I would like to argue that even this kind of temporal non-locality presents a problem from a psychological, if not biological, perspective.

The problem is that the stochastic estimate of unit variance implicit in the Oja algorithm would be distorted by the clustering of categories in time. In the tree-classification example, all of the units in the network would end up representing subspecies of palm trees in the tropics and subspecies of pine trees in the Rockies, in order to maintain high levels of output variance over time spans shorter than half a year. Clearly, people are capable of maintaining representations even after years without using them, and extended exposure to a particular environment does not automatically lead to the kind of fine representation of the categories or features of that environment that one would expect if all of one's neurons were maintaining a high variance of activity. However, some movement in these directions is to be expected, so that there should be some sensitivity to the relevance of a category over time.

One implication of this argument is that any reliance on the information content of a unit's output signal constitutes a temporal non-locality. Under this definition, a wide range of unsupervised algorithms suffer from the problem, including the BCM rule and its derivatives [3, 6], decorrelation algorithms [4], and algorithms that maximize mutual information between units [2]. In Linsker's [8] proposal of the Infomax principle, which clearly depends on the information content of the output signal, he acknowledges the assumption of a uniformly distributed input environment, but claims that Infomax is more general than variance maximization because it is really about maximum information *preservation*, which can be defined even for non-uniform input environments. Instead of attempting to conditionalize something like the Infomax principle on various non-uniform assumptions about the input environment, the approach presented in this paper is to formulate the objective of an unsupervised algorithm in a way that avoids the use of the output signal information entirely.

The principle that I would like to suggest is called "MaxIn" for MAXimizing INput INformation. The objective of MaxIn is to develop units in a network which maximize the quality of information they receive from input patterns, as opposed to the amount of information they transmit over time. The central idea behind MaxIn is that the units can be viewed as something like matched filters [5, 15] which are optimally tuned to respond to a signal in the presence of noise. When matched filters are used in signal processing, they typically respond to a temporal signal like a pulse of light in a fiber-optic channel. In applying the concept to neural networks, the signal is instead the instantaneous pattern of activity over the input units, and the filter is parameterized by the weights of the unit from these input units. The objective of MaxIn is to adapt these weights so as to maximize the signal-to-noise ratio (SNR) of a given input. More precisely, it seeks to maximize the divergence between the probability of a unit becoming active when the signal is present and the probability when the signal is not present.

Derivation of the MaxIn Algorithm

To formalize this notion, consider a network having input units $\vec{x}$ whose activity is governed by an environment which contains regularities that are to be abstracted. A particular state of the input vector is denoted by the subscript k. For a given input vector, the output unit activations reflect the similarity between the input vector and the weight matrix (i.e. filter) for the output unit. For the sake of analytical simplicity, we model the activation of the output unit states y_j in probabilistic terms. Adopting a *generative model* with spherical Gaussian basis functions [10], it is useful to consider the activation as reflecting the probability that the input vector is generated by a given output unit whose mean is given by the weight vector $\vec{w}_j$. This probability is:

$$P_j(\vec{x}_k) = \frac{1}{K\sigma_j} e^{-\frac{\|\vec{x}_k - \vec{w}_j\|}{2\sigma_j^2}} \tag{3}$$

For the moment, we consider a single output unit, and compute the MaxIn function for it. In this case, the activation state of the output unit y_j is just $P_j(\vec{x}_k)$. The MaxIn objective function can be specified as the divergence between the probability of the unit having generated the input vector and the probability of the same unit (with the same mean $\vec{w}_j$) having generated an input vector of random noise:

$$M_s = P_j(\vec{x}_k) ln \frac{P_j(\vec{x}_k)}{P_j(\vec{n}_k)} \tag{4}$$

where the noise vector $\vec{n}_k$ is a vector of normally distributed random numbers with variance σ_k^2 and mean μ_k.

This function is maximized when the probability of generating the input vector is large relative to the probability of generating the random noise vector. Put another way, the unit will respond maximally to the input vector signal relative to its response to the noise. This is the same underlying principle as the G-Max algorithm proposed by Pearlmutter & Hinton [13], which maximizes the information gain of a unit from structured input patterns relative to the case where the input units are statistically independent (i.e. "unstructured"). Recently, Linsker [7] proposed a formulation that includes an idea similar to G-Max, which also makes use of the difference between structured and unstructured input "phases".

The critical difference between MaxIn and these G-Max style algorithms is the way in which the noise is estimated and used in the weight update. In G-Max, the network is run in two different phases, one with structured input and another with unstructured input, in order to sample both the actual environment and the case of independent input units. This makes the learning rule non-local in time, since two phases must be run to update the weights. In MaxIn, the unstructured input or "noise vector" is estimated using variables which are local to the unit both in space and time. Instead of sampling many different noise vectors to obtain the necessary estimate, the expected value of the noise vector is used. Thus, assuming normally distributed noise, this is just a vector having each element equal to the mean μ_k. Further, the value of μ_k for the noise is estimated based on the mean input activity level for the current input vector:

$$\mu_k = \frac{1}{N_i} \sum_i x_{ik} \tag{5}$$

providing the necessary locality of information. Thus, the noise is assumed to be of the same overall strength as the signal, only it is uniformly distributed across the input lines. MaxIn causes the weights to accentuate any deviation from this uniform distribution present in the signal.

The derivative of the MaxIn objective function with respect to the weights provides the gradient which can be used for learning. The complete derivative is somewhat complicated, and it is difficult to imagine a biological neuron computing it:

$$\frac{\partial M_s}{\partial w_{ij}} = \frac{1}{\sigma_j^2} P_j(\vec{x}_k) \left[(x_{ik} - \mu_k) + (x_{ik} - w_{ij}) ln \frac{P_j(\vec{x}_k)}{P_j(\vec{n}_k)} \right] \tag{6}$$

We label this the "Single-unit Full Derivative" or SFD MaxIn. However, there is a simplification which results in a very simple and biologically plausible weight update rule. This simplification is derived by assuming that the weight update is only supposed to optimize the log SNR term, and not the overall probability of generating the input vector. Under this assumption, the first $P_j(\vec{x}_k)$ term is treated as a constant with respect to the weights, resulting in the following approximation of the full derivative:

$$\frac{\partial M_s}{\partial w_{ij}} \approx \frac{1}{\sigma_j^2} P_j(\vec{x}_k)(x_{ik} - \mu_k) \tag{7}$$

The weight update rule that results from the simplified derivative is just:

$$\Delta w_{ij} = \frac{\epsilon}{\sigma_j^2} y_j (x_{ik} - \mu_k) \tag{8}$$

which we label "Single-unit Simplified Derivative" or SSD MaxIn.

SFD MaxIn can be thought of as the combination of the simpler SSD MaxIn learning rule with a learning rule that closely resembles standard competitive learning (CL) [14]. In our notation, CL can be written as:

$$\Delta w_{ij} = \epsilon y_j (x_{ik} - w_{ij}) \tag{9}$$

while a learning rule based on the full derivative contains equation 8 plus:

$$y_j (x_{ik} - w_{ij}) ln \frac{y_j}{y_j'} \tag{10}$$

where y_j' denotes the response of the unit to the noise input vector. Interestingly, if the unit is responding the same to the signal and the noise, this term goes to zero, and only the SSD equation is effective. However, as the SNR improves, this term gets stronger. The relevance of this additional term is explored in simulations reported below. Further, a hybrid algorithm that is a simple combination of SSD MaxIn and CL (without the log SNR weighting term of SFD) is tested as well.

One consequence of the SSD equation is that individual weights can increase or decrease without bound. Thus, in order for a MaxIn algorithm to be used in simulations, a weight bounding procedure must be used. While it is possible to simply clip the weights at 0 and 1, a more natural form of weight bounding multiplies weight increases by $(1 - w_{ij})$ and weight decreases by w_{ij}, resulting in an exponential approach to the limits of 0 and 1. This was used in all MaxIn simulations reported below.

Extension to Multiple Output Units

The algorithm described above was derived for a single unit functioning in isolation. Such a unit would have a tendency to respond to all the input patterns. Usually it is desirable to have a layer of units which specialize on different clusters of the input patterns. One way this can be done is by making the activation of the unit reflect the likelihood that it generated the input vector as compared to all the other units. This could be thought of as the probability of the output unit having generated the given input vector [10]:

$$y_j = P(j|\vec{x}_k) = \frac{P_j(\vec{x}_k)}{\sum_l P_l(\vec{x}_k)} \tag{11}$$

where the index l goes over all the output units. Thus, the total probability of generating the input vector, summed over all units, is one.

Given the network just described, the MaxIn objective becomes:

$$M_m = P(j|\vec{x}_k) ln \frac{P(j|\vec{x}_k)}{P(j|\vec{n}_k)} \tag{12}$$

This can be expanded into a form that more easily shows the relationship between this multi-unit version and the previous single-unit version:

$$M_m = P(j|\vec{x}_k) \left(ln \frac{P_j(\vec{x}_k)}{P_j(\vec{n}_k)} - ln \frac{\sum_l P_l(\vec{x}_k)}{\sum_l P_l(\vec{n}_k)} \right) \tag{13}$$

Taking the derivative of this function is simplified by noting that $\frac{\partial (ln \sum_l P_l(\vec{x}_k))}{\partial w_{ij}} = P(j|\vec{x}_k)(x_i - w_{ij})$. This results in the "Multi-unit Full Derivative" (MFD):

$$\frac{\partial M_m}{\partial w_{ij}} = \frac{1}{\sigma_j^2} P(j|\vec{x}_k) \left([(x_{ik} - \mu_k) - (P(j|\vec{x}_k)(x_{ik} - w_{ij}) - P(j|\vec{n}_k)(\mu_k - w_{ij}))] + \right.$$

$$\left[(x_{ik} - w_{ij})(1 - P(j|\vec{x}_k) ln \frac{P(j|\vec{x}_k)}{P(j|\vec{n}_k)}\right]\Bigg) \tag{14}$$

Obviously, this is a rather more complicated function than the simple single-unit MaxIn functions derived above. However, a simplification can be made by ignoring the effect of weight changes on the sum terms from equation 13 under the assumption that the effect of a single weight change on a sum over all the output units will likely be small. This results in the "Multi-unit Constant-sum Full Derivative" (MCFD):

$$\frac{\partial M_m}{\partial w_{ij}} \approx \frac{1}{\sigma_j^2} P(j|\vec{x}_k) \left[(x_{ik} - \mu_k) + (x_{ik} - w_{ij})(1 - P(j|\vec{x}_k) ln \frac{P(j|\vec{x}_k)}{P(j|\vec{n}_k)}\right] \tag{15}$$

which resembles the SFD equation (6).

In addition, the same arguments made for simplifying the SFD into the SSD equation can be made here (i.e. maximizing only the SNR term, while treating the $P(j|\vec{x}_k)$ as a constant with respect to the weights), resulting in a MSD (Multi-unit Simplified Derivative) equation:

$$\frac{\partial M_m}{\partial w_{ij}} \approx \frac{1}{\sigma_j^2} P(j|\vec{x}_k)(x_{ik} - \mu_k) \tag{16}$$

which produces the same weight update function as SSD (see equation 8) when $y_j = P(j|\vec{x}_k)$. This simple learning rule has the form of a noise differentiation term $(x_{ik} - \mu_k)$ which is gated by a kind of "competitive success" term $(P(j|\vec{x}_k))$, so that only those units which are likely to generate a given input vector have their weights updated. This allows the representational space to be partitioned off by the different units.

Biological Computability

The critical feature of the simplest MaxIn algorithm (equation 8) in terms of biological computability is that the weight change is driven by a simple difference between the input from one unit and the average of all other inputs to the unit. It is well known that the passive dendrite on a neuron acts like a diffusion process, so that discrete synaptic inputs in different spatial locations will result in a moderate elevation of the membrane potential over a wide region of the dendrite. Thus, the potential at any given synapse will be a sum of both the local input from the synapse and a more global potential elevation due to other synaptic inputs. The μ_k parameter in the MaxIn algorithm plays the role of this diffuse, global potential elevation. Thus, we can write the local synaptic potential as $V_{ij} = x_{ik} + \mu_k$.

Recent findings in long-term potentiation (LTP) and depression (LTD) [1] suggest that whether a weight experiences potentiation or depression depends upon two thresholds. If the cell is not potentiated at all, no weight change occurs. If the cell is somewhat potentiated (e.g. above a low threshold) weight decrease occurs. However, if the cell is potentiated above a higher threshold, weight increase occurs. In the context of the synaptic potential V_{ij} given above, the low threshold might be right around μ_k, so that a synapse receiving no additional input while the cell is being excited by other inputs will experience LTD. If the high threshold were around $2\mu_k$, then inputs around the average would fall between LTD and LTP, and would not experience synaptic modification. However, inputs stronger than μ_k would be above the LTP threshold, and would result in potentiation.

Evaluation of the MaxIn Algorithm

There are a multitude of other algorithms which MaxIn can be compared to, but the simplest is probably the competitive learning (CL) algorithm [14]. In particular, the "Soft" version of this algorithm (SCL) as presented by Nowlan [10] shares the same probabilistic assumptions as those used in the derivation of the MaxIn algorithm. SCL uses the following weight update rule:

$$\Delta w_{ij} \quad = \quad \frac{1}{k_j(t-1) + y_j} y_j (x_{ik} - w_{ij})$$

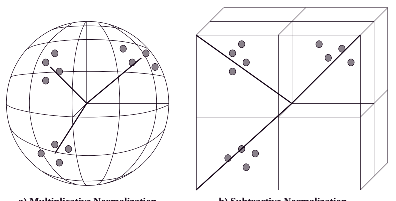

Figure 1: **a)** Multiplicative weight normalization yields (n-dimensional) spherical weight vectors pointed at the center of cluster of input patterns. **b)** Subtractive weight normalization yields weight vectors pointing at the corner of a (n-dimensional) hypercube closest to the input cluster.

$$k_j(t) \quad = \quad k_j(t-1) + y_j \tag{17}$$

where the parameters y_j, w_{ij} and x_{ik} are defined as in the multi-unit MaxIn derivation. This learning rule is the same as CL (equation 9) using a learning rate that decreases over time. This is the "statistically correct" way to perform iterative (on-line) weight updates, but in practice k_j is initialized to $.5N/M$ and bounded by N/M, where N is the number of input patterns and M is the number of output units. Thus, it is a reasonable simplification to replace this adaptive learning rate with a constant, which is what we do in our simulations for reasons that are explained below.

In comparing MaxIn with SCL, the issue of multiplicative *vs* subtractive normalization is relevant. This is explored first, followed by simulations which show what happens to these algorithms when the input patterns are clustered in time.

Subtractive *vs* Multiplicative Normalization

Competitive Learning is closely related to the Oja rule (equation 2), which, with linear output units, normalizes the weight vector to unit length. This causes each weight to reflect the weighted average (over all input units and input patterns) of each input's tendency to be active. This can be seen in a simple way by replacing y_j with the linear activation function: $\sum_i x_{ik} w_{ij}$, and computing the equilibrium weight value for a single input vector. When $\Delta w_{ij} = 0$, $x_{ik} = y_j w_{ij}$, and:

$$\breve{w}_{ij} = \frac{x_{ik}}{\sum_l x_{lk} w_{lj}} \tag{18}$$

where ($\breve{\ }$) denotes equilibrium, and l is an index over all the input units. The pattern-wise equilibrium is just a similar weighted sum over all input patterns. This makes it clear that the Oja rule is performing what Miller & MacKay [9] term "multiplicative" (i.e. dividing by the sum) normalization.

In contrast, the MaxIn learning rule (in the simplified form) performs a kind of subtractive normalization. Indeed, the MaxIn rule of equation 8 is identical to the Zero-Sum Hebbian learning rule (ZSH) [12], which causes the total weight increase to a unit to be balanced by an equal amount of weight decrease, yielding a net change of zero. This should be evident from equation 8 since μ_k is the mean of the elements in the vector $\vec{x}_k$ (see equation 5). While Miller & MacKay [9] discuss the differences between multiplicative and subtractive normalization at length, a simple figure captures the essence. The multiplicative normalization in the Oja rule causes the weight vector to move to the *center* of the clusters in the data as can be seen in the hypersphere representation in figure 1. In contrast, the subtractive normalization in MaxIn causes the weights to go towards the *nearest corner* of the hypercube to the data cluster. Note that this effect is independent of the activation function.

While SCL does not have the normalization properties of the Oja rule, it does result in a weight vector pointing at the center of the cluster of input patterns the unit represents. The importance of the corner *vs* the center distinction is that since the weights under MaxIn have a pressure to end up in a corner of weight space, they have an intrinsic preferred direction for distinguishing between input clusters. As is shown in figure 2

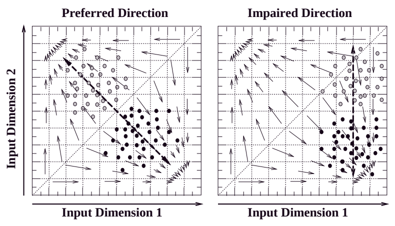

Figure 2: Subtractive normalization like that present in MaxIn results in a preferred direction (indicated by the vector field lines) for the weight changes, enhancing distinctions along this direction and impairing them otherwise. The second panel shows an example of an impaired dimension.

for the two-dimensional case, this direction lies perpendicular to the diagonal between input dimensions. Interestingly, this direction corresponds to the case of the input clusters having their centers at points where the coordinates sum to the same value (i.e. they have the same mean in the sense in which μ_j is computed for the MaxIn algorithm).

One would expect that the subtractive normalization in MaxIn would enhance its ability to distinguish categories along the preferred direction and impair distinctions along other directions. Algorithms that put the weight vectors at the center of the clusters, like SCL, on the other hand, should not have a preferred direction. These predictions were tested by constructing two-dimensional input patterns drawn from two Gaussian generators located at different points in the input space, but with a common σ of .2. In one case, the centers of the Gaussians were located along the preferred direction for MaxIn. As noted above, the coordinates of these centers have the same mean for the two Gaussians, so this is called the Same Mean (SM) condition. The coordinates used were [(.1,.9) (.9,.1)], [(.2,.8) (.8,.2)], [(.3,.7) (.7,.3)], [(.4,.6) (.6,.4)], and [(.45,.55) (.55,.45)]. In contrast, the Different Mean (DM) Gaussians were located at a fixed coordinate for the first dimension (either .8 or .2) with the coordinate for the second dimension varied as in the SM case. For example, DM8 is [(.8,.1) (.8,.9)], [(.8,.2) (.8,.8)], [(.8,.3) (.8,.7)], [(.8,.4) (.8,.6)].

Temporal Clustering of Input Patterns

A distribution of input patterns from the same category clustered over time presents a problem for algorithms which assume a uniform distribution in time. The relationship between SCL and Oja's algorithm, which approximately maximizes the variance of a unit's output signal under the assumption that input samples are distributed uniformly in time, indicates that temporal clustering should affect SCL. However, since MaxIn requires no such assumptions, it should be relatively less affected by clustering. These predictions were tested using the input clusters described above by either presenting all instances of a given cluster together, followed by all of those from the other cluster, or by permuting the presentation order so that the instances were uniformly distributed in time. Any differences between these two conditions should indicate a sensitivity to temporal clustering. Obviously, the learning rate parameter and the number of patterns from each cluster presented in a row will affect the sensitivity observed. If the learning rate is low relative to the number of patterns, then the stochastic estimate of output variance will not be dramatically affected by temporal clustering.

Methods and Results

The networks for the MaxIn and SCL algorithms were the same except for the learning rule used. There were 2 input units and 2 output units receiving connections from both input units. The output units computed their activations based on the multi-unit equations described above, with a fixed σ_j parameter of .2. The learning rate ϵ for the MaxIn simulations was either .1 or .02, and .05 or .01 for SCL. The difference is due to the use of weight bounding for MaxIn that can result in smaller weight changes (if the weight was at .5, the weight change would be half as strong in either direction compared with no weight

Model	SM $\Delta\mu$	%	PSM $\Delta\mu$	%	DM8 $\Delta\mu$	%	PDM8 $\Delta\mu$	%	DM2 $\Delta\mu$	%	PDM2 $\Delta\mu$	%
SFD MaxIn	.1	100	.1	100	.2	76	.2	68	.2	96	.2	44
lrate=.02	.1	100	.1	100	.2	80	.2	72	.2	88	.2	52
MCFD MaxIn	.1	100	.1	100	.2	52	.2	68	.2	88	.2	20
lrate=.02	.1	100	.1	100	.2	64	.2	20	.2	12	.2	12
HYB MaxIn	.1	100	.1	100	.2	100	.2	52	.8	0	.2	40
lrate=.02	.1	100	.1	100	.2	100	.2	36	.8	100	.2	28
SSD MaxIn	.1	100	.1	100	.6	100	.8	24	.8	36	.8	0
lrate=.02	.1	100	.1	100	.8	16	.8	28	.8	0	.8	0
Soft CL	.6	100	.1	72	.8	100	.2	100	.8	100	.2	100
lrate=.01	.1	84	.1	100	.2	100	.2	100	.2	100	.2	100

Table 1: Results of simulations comparing several versions of the MaxIn algorithm (see text for a description) with the Soft Competitve Learning (Soft CL) algorithm. The centers of two clusters in the 2-dimensional input data were separated by $\Delta\mu$ in each dimension. The means of these clusters were either the same distance away from the origin (SM) or different distances from the origin (DM, with X=.8 or X=.2). The results show the % of networks (sample N of 25) that successfully discriminated these clusters at the lowest $\Delta\mu$ for which they had a non-zero %. For SM, the minimum $\Delta\mu$ was .1 and the maximum .8, while DM had a min of .2 and the same max. The input patterns were presented sequentially by cluster, except in the Permuted (P) cases. See text for a description of SM vs. DM.

bounding). However, since strong weights decrease faster than weak ones (and weak weights increase faster) with the weight bounding than without, this would tend to impair the ability to maintain extreme weights under pressure to move to the other corner of weight space. Thus, the robustness of MaxIn under temporal clustering of input patterns cannot be attributed to the weight bounding procedure.

Simulation results are shown in table 1. The manner in which the centers of the input Gaussians were selected is denoted across the top (see description above), with 200 points per cluster used in training. The learning algorithms tested are listed in order of performance. In addition to the flavors of MaxIn described previously, a hybrid algorithm consisting of SSD plus the SCL algorithm was tested. For each cell in the table, the $\Delta\mu$ indicates the distance between the means of the input Gaussians along each dimension, and the % indicates the percent of trained networks having each unit's weight vector closest to the center of a different cluster (i.e., each unit represents one of the clusters).

Discussion

Using just the SM results, it is clear that SCL was quite impaired with the .05 learning rate in the temporally clustered condition, as it failed to distinguish Gaussian categories centered closer than .6 on each dimension. However, when the patterns were presented in permuted order (PSM), or when the learning rate was lowered to .01, the performance improved significantly, so that it was able to distinguish clusters only .1 apart on each dimension. In comparison, all of the MaxIn algorithms achieved this level of performance even in the temporally clustered, high learning rate condition. Also, even in the PSM condition, MaxIn exhibited better performance at the .1 distance than SCL for the high learning rate, indicating that MaxIn is better able to distinguish clusters along its preferred direction.

The DM conditions reveal that the SSD MaxIn rule is quite impaired at distinguishing clusters along directions other than its preferred one, even with permuting and low learning rate. However, the more complete derivative formulations of MaxIn, including a simple hybrid of SSD MaxIn and SCL, retain the insensitivity to temporal clustering but also enable the representation of clusters that are along impaired directions for SSD. The best overall performance appears to come from SFD MaxIn, but the difference between it and MCFD are probably not significant. However, both of these algorithms require the computation of $P(j|\vec{n}_k)$, which complicates the algorithm and makes it less biologically plausible. Thus, the hybrid probably represents the best balance between simplicity and overall performance.

References

[1] A. Artola, S. Brocher, and W. Singer, "Different voltage-dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex," *Nature*, vol. 347, pp. 69–72, 1990.

[2] S. Becker and G. E. Hinton, "Learning mixture models of spatial coherence," *Neural Computation*, vol. 5, pp. 267–277, 1993.

[3] E. L. Bienenstock, L. N. Cooper, and P. W. Munro, "Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex," *The Journal of Neuroscience*, vol. 2, no. 2, pp. 32–48, 1982.

[4] P. Földiák, "Forming sparse representations by local anti-Hebbian learning," *Biological Cybernetics*, vol. 64, no. 2, pp. 165–170, 1990.

[5] R. Hecht-Nielsen, *Neurocomputing.* Redwood City, CA: Addison-Wesley, 1990.

[6] N. Intrator, "Feature extraction using an unsupervised neural network," *Neural Computation*, vol. 4, pp. 98–107, 1992.

[7] R. Linsker, "Local synaptic learning rules suffice to maximize mutual information in a linear network," *Neural Computation*, vol. 4, pp. 691–702, 1992.

[8] R. Linsker, "Self-organization in a perceptual network," *Computer*, pp. 105–117, March 1988.

[9] K. D. Miller and D. J. MacKay, "The role of constraints in hebbian learning," CNS Memo 19, Caltech, 1992.

[10] S. J. Nowlan, "Maximum likelihood competitive learning," in *Advances in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), San Mateo, CA: Morgan Kaufmann, 1990.

[11] E. Oja, "A simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, vol. 15, pp. 267–273, 1982.

[12] R. C. O'Reilly and J. L. McClelland, "The self-organization of spatially invariant representations," Parallel Distributed Processing and Cognitive Neuroscience PDP.CNS.92.5, Carnegie Mellon University, Department of Psychology, 1992.

[13] B. A. Pearlmutter and G. E. Hinton, "G-maximization: An unsupervised learning procedure for discovering regularities," in *Neural Networks for Computing* (J. Denker, ed.), pp. 333–338, New York: American Institute of Physics, 1986.

[14] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," in *Parallel Distributed Processing. Volume 1: Foundations* (D. E. Rumelhart, J. L. McClelland, and PDP Research Group, eds.), ch. 5, pp. 151–193, Cambridge, MA: MIT Press, 1986.

[15] M. Schwartz, *Information Transmission, Modulation, and Noise.* New York: McGraw-Hill, 4 ed., 1990.

[16] D. Willshaw and P. Dayan, "Optimal plasticity from matrix memories: What goes up must come down," *Neural Computation*, vol. 2, pp. 85–93, 1990.