Exploring the Role of Context and Sparse Coding on the Formation of Internal Representations

David A. Medler (medler@cnbc.cmu.edu)
James L. McClelland (jlm@cnbc.cmu.edu)
Center for the Neural Basis of Cognition; Carnegie Mellon University
Pittsburgh, PA 15213 USA

Abstract

Recently, Bayesian principles have been successfully applied to connectionist networks with an eye towards studying the formation of internal representations. Our current work grows out of an unsupervised, generative framework being applied to understand the representations used in visual cortex (Olshausen & Field, 1996) and to discover the underlying structure in hierarchical visual domains (Lewicki & Sejnowski, 1997). We modified Lewicki and Sejnowski's approach to study how incorporating two specific constraints—context and sparse coding—affect the development of internal representations in networks learning a feature based alphabet. Analyses of the trained networks show that (1) the standard framework works well for limited data sets, but tends to poorer performance with larger data sets; (2) context alone improves performance while developing minimalistic internal representations; (3) sparse coding alone improves performance and actually develops internal representations that are somewhat redundant; (4) the combination of context and sparse coding constraints increases network accuracy and forms more robust internal representations, especially for larger data sets. Furthermore, by manipulating the form of the sparse coding constraint, networks can be encouraged to adopt either distributed or local encodings of surface features. Feedback connections in the brain may provide context information to relatively low-level visual areas, thereby informing their ability to discover structure in their inputs.

Introduction

Bayesian principles have been regaining popularity within cognitive science, both in the more traditional approaches to cognitive psychology (e.g., Anderson, 1990) and within the connectionist approach to cognition (e.g., MacKay, 1995; McClelland, 1998). Our current work is a preliminary investigation of incorporating two specific constraints, *context* and *sparse coding*, into an existing Bayesian unsupervised learning paradigm for multilayered architectures (Lewicki & Sejnowski, 1997). The concept underlying the original framework is that higher order internal representations can be formed by exploiting the statistical structure of simple features within an input stream. Indeed, Lewicki and Sejnowski were able to show that their networks could extract hierarchical structure from simple visual domains.

In this paper, we modify and expand the original framework to explore the internal representations of networks trained on feature-based letters. We first modified the framework to directly incorporate contextual information into the deep structure of the network. In the current experiments, "context" is defined as unique information that is presented to the network concurrently with an input pattern. Therefore, context can be used to uniquely identify input patterns and, thus, provide hints about which collection of simple features constitute higher-order representations.

The second—and more substantial—manipulation was to place prior constraints on the base probabilities of unit activations within the networks. This "sparse coding" constraint encourages a network to use relatively few units to represent any specific input pattern. That is, a sparsely coded network uses only a relatively small proportion of units to encode the internal representation for a given pattern. Sparse encoding within neural networks has previously been shown to create more biologically plausible receptive fields (e.g., Olshausen & Field, 1996, 1997).

Three different experiments were carried out. The first two experiments used a reduced stimulus set to study the base effects of independently manipulating the context and sparse coding constraints; Experiment 1 manipulated context with the simplest sparse coding constraint, while the second experiment specifically focused on different forms of the sparse coding constraint. In Experiment 3, the context and sparse coding constraints were investigated using the full alphabet. Networks were analyzed both in terms of their ability to reproduce the training set and in terms of their internal structure via weight analysis.

Networks and Bayesian Theory

It is assumed that the internal representations used by a system must come to represent the external world in some manner. In other words, internal representations could be thought of as hypotheses about the external world. Thus, the problem of defining these internal representations can be reformulated as computing the probability of a given hypothesis (internal representation) given the observed data (external world)—a potentially difficult task.

Fortunately, a relatively simple theoretical framework exists for computing this probability. In its simplest form, *Bayes' theorem* (see Equation 1) states that for a given hypothesis, $\mathcal{H}$, and observed data, $\mathcal{D}$, the posterior probability of $\mathcal{H}$ given $\mathcal{D}$ is computed as

$$P(\mathcal{H}|\mathcal{D}) = \frac{P(\mathcal{D}|\mathcal{H}) \times P(\mathcal{H})}{P(\mathcal{D})} \quad (1)$$

where $P(\mathcal{H})$ is regarded as the prior probability of $\mathcal{H}$ before observing the data $\mathcal{D}$, $P(\mathcal{D})$ is the probability of the data, and $P(\mathcal{D}|\mathcal{H})$ is the probability of the data given the hypothesis. Thus, by specifying $P(\mathcal{D}|\mathcal{H})$ and $P(\mathcal{D})$, the mechanisms of

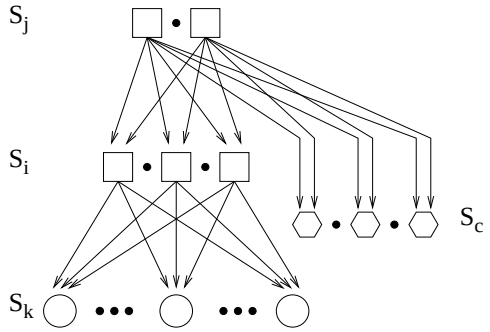

Figure 1: The basic network configuration for a three-layer network. S_k are surface units; S_i are mediating layer units; S_j are deep layer units; S_c are context units.

Bayesian theory provide a solution to the problem of learning from data (Bernardo & Smith, 1994; MacKay, 1995; McClelland, 1998).

We can also rearrange the model to predict the data given the hypothesis; in other words, this framework can be used to construct a *generative* model, such that the higher-order internal representations predict the lower-level simple features.

Network Architecture

To help explain the network dynamics, we will consider the simple case of a three-layer network (see Figure 1) consisting of an surface layer, S_k, a single mediating layer, S_i (in practice, there could be any number of mediating layers), and a deep layer, S_j. It is assumed that connections exist only between adjacent layers; that is, there are no direct connections between the surface and deep structure layers. Furthermore, the generative nature of the model means that connections are uni-directional and flow from the deep to the surface layer as indicated by the directed connections within Figure1. Thus, we can define three different relationships for a given unit; the parents ($pa[S_i]$; units contributing activation), children($ch[S_i]$; units receiving activation), and siblings ($sib[S_i]$; units within the same layer).

Units are assumed to be stochastic and are probabilistically active or inactive as determined by the summed activations being sent by their parents via weighted connections. Consequently, the network weight vector, $\mathbf{W}$, can be interpreted as encoding the underlying probabilities of the generative model. This means that weights are constrained to be zero or positive.

In the present studies, this basic network architecture has been expanded to include a *context*-layer, (S_c), as illustrated by the hexagonal units in Figure 1. This context layer is connected directly to the deep layer and thus provides contextual information to the deep layer only.

Learning Objective

Within this framework, the learning objective is to find the most probable explanation, $\mathcal{H}$, for the input patterns, $\mathcal{D}$, presented to the network. In other words, we wish to develop a generative model that encodes the probabilities of the input data within the network's weight structure. Therefore, the learning objective reduces to adapting the weight vector, $\mathbf{W}$, to find the most probable explanation for the input patterns.

If we knew the weight vector, we could calculate the probability of the input data as

$$P(\mathcal{D}_{1:N}|\mathbf{W}) = \prod_n P(\mathcal{D}_n|\mathbf{W}) \tag{2}$$

where

$$P(\mathcal{D}_n|\mathbf{W}) = \sum_m P(\mathcal{D}_n|\mathcal{S}_m, \mathbf{W})P(\mathcal{S}_m|\mathbf{W})$$

is the marginalization of all possible unit states, $\mathcal{S}_m$, of the network.

It should be noted that the number of all possible network states, $\mathcal{S}_m$, increases exponentially with the number of units in the network. Therefore, computing the exact sum becomes intractable as the networks become larger. One desirable property of generative models, however, is for most patterns to have one—or just a few—possible explanations; therefore, only a few terms, $P(\mathcal{D}_n|\mathcal{S}_m, \mathbf{W})$, will be non-zero and it becomes tractable to sample $\mathcal{S}_m$ according to $P(\mathcal{S}_m|\mathcal{D}_n, \mathbf{W})$.

Of course, we do not know the weight vector but must adapt it instead. One way of adapting the weight vector is to use a variation of the *expectation maximization* (EM) algorithm. EM is typically used to find parameter estimates in models where some variables are unknown or unobserved. The algorithm is composed of two steps: (i) an estimation (E) step that samples network states , and (ii) a maximization (M) step that adjusts weights. For our purposes, the E step can be accomplished by Gibbs sampling whereas the M step can use maximum-likelihood (ML) estimation.

Computing Network Probabilities

Being a generative model, the probability of any unit's state is directly computable from the states of its parents:

$$P(S_i = 1|pa[S_i], \mathbf{W}) = h(\sum_j S_j w_{ij}) \tag{3}$$

where S_j are the parents of S_i and w_{ij} is the weight from unit S_i to S_j.The function h in Equation 3 specifies how these underlying causes are to be combined to produce the probability of $S_i = 1$. One function that can be used for this is the "noisy OR" function:

$$h(u) = 1 - e^{-u} \tag{4}$$

where $u = \sum_j S_j w_{ij}$ is the causal input to S_i. Note that because weights are constrained to be positive, u is never negative, and therefore $0 \leq h(u) \leq 1$.

Thus, the joint probability density of a such a network can be computed as the product of the conditional probabilities

$$P(S_1 \ldots S_n|\mathbf{W}) = \prod_i P(S_i|pa[S_i], \mathbf{W}). \tag{5}$$

Sampling Network States

In Lewicki and Sejnowski's (1997) original formulation of the problem, each state of the network, $\mathcal{S}_m$, is updated iteratively according to the probability of each unit state, S_i, given the states of the remaining units in the network. This conditional probability is computed as

$$P(S_i|S_{j;j\neq i}, \mathbf{W}) \propto P(S_i|pa[S_i], \mathbf{W}) \prod_{j\in ch[S_i]} P(S_j|pa[S_j], S_i, \mathbf{W}) \tag{6}$$

Thus, the Gibbs equations as used in this framework can be interpreted in terms of a stochastic recurrent neural network, where the feedback from the higher (or deeper) layers influences the states at the lower (or surface) layers. Whereas Lewicki and Sejnowski (1997) computed the probability of a unit *changing* its state, the problem can be reformulated as one where the probability of a unit being active given the remaining states of the network is calculated.

Consequently, one can compute the probability of a unit being active, $S_i = 1$, given the remaining states of the network as

$$P(S_i = 1 | S_{j;j \neq i}, \mathbf{W}) = \frac{1}{1 + e^{-\Delta x_i}} \tag{7}$$

This function will produce a $P(S_i = 1) \approx 0$ for negative evidence, a $P(S_i = 1) \approx 0.5$ for inconclusive evidence, and $P(S_i = 1) \approx 1$ for positive evidence.

The variable Δx_i in Equation 7 indicates how much changing the unit state, S_i, to being active changes the overall probability of the network state. In multilayered networks (where the number of layers is greater than 2), this term will have both a feedback component from the parents in the deeper layers, and a feedforward component from the children in the more surface layers:

$$\Delta x_i = fb(pa[S_i]) + ff(ch[S_i]) \tag{8}$$

In networks with only two layers, or in the deepest layer of a multilayer network, this feedback term will drop out. Typically, the feedforward component of Equation 8 will dominate the term, but if the feedforward input is ambiguous, then the feedback component becomes important as it allows the more surface level units to use information computable only at the deeper layers.

The feedback term in Equation 8 is simply computed as the log probability of the unit being on minus the log probability of the unit being off. This is calculated as:

$$fb(S_i) = \log \frac{h(u)}{1 - h(u)} \tag{9}$$

where the function $h(u)$ is computed as described earlier.

Feedforward is computed as the probability of the unit being on given the activity of its children. Therefore, for a given unit, we want to sum the evidence of the unit being on minus the evidence of the unit being off. We also want to weight the evidence according to the number of other units contributing to the child's activity (the more units contributing, the less effect any one unit will have).

$$ff(S_i) = \sum_{k \in ch[S_i]} S_k \log \frac{h(u - S_i w_{ik} + w_{ik})}{h(u - S_i w_{ik})} + (1 - S_k) \log \frac{1 - h(u - S_i w_{ik} + w_{ik})}{1 - h(u - S_i w_{ik})} \tag{10}$$

Thus, if $S_i = 0$, then the weight from S_i is added to the top portions of Equations 10, whereas if $S_i = 1$ then the weight from S_i is removed from the bottom portion of the above equation. Furthermore, if $S_k = 0$ (indicating that the parent node should be off), then the first term of Equation 10 drops out, whereas if $S_k = 1$ (indicating that the parent node should be on), then the second term of Equation 10 drops out.

Adding Contextual Information

As defined earlier, context is the added information that can provide hints about which collection of simple features constitute higher-order representations, and thus helps constrain the internal representations developed at the deep-layer. Context can be added to the network dynamics simply by directly connecting a set of context units (denoted S_c in Figure 1) to the deep-layer units, S_j, via weighted connections w_{cj}.

$$ff(S_j) = \sum_{c \in cn[S_j]} S_c \log \frac{h(u - S_j w_{cj} + w_{cj})}{h(u - S_j w_{cj})} + (1 - S_c) \log \frac{1 - h(u - S_j w_{cj} + w_{cj})}{1 - h(u - S_j w_{cj})} \tag{11}$$

where $cn[S_j]$ are the context units directly connected to the deep-layer units. Thus, context information is directly added to the activation probabilities of the deep-layer units by summing the contributions of Equations 10 and 11.

Adding Sparse Coding Constraints

A further modification to the original framework is to add a sparse coding constraint on the unit activation probabilities. That is, all things being equal, sparse coding encourages a network to use relatively few units to represent any specific input pattern. In the standard framework, in the absence of any guiding information, a unit will be active with baseline probability $P = 0.5$. Sparse coding can be encouraged within the network by modifying Equation 8 to include a sparcity constraint:

$$\Delta x_i = fb(pa[S_i]) + ff(ch[S_i]) + \lambda \cdot sp(S_i) \tag{12}$$

where λ is equivalent to a gain function which modulates how much effect $sp[S_i]$ exerts on the rest of the equation.

Four sparse coding functions are defined. The first and simplest function is an implicit, independent prior constraint that reduces the baseline probability of a unit being active by a constant amount:

$$\text{Constant}: \quad sp_1[S_i] = log\tfrac{\phi}{1-\phi}\ , \ \ 0 \leq \phi \leq 1 \tag{13}$$

The three other functions defined encourage sparse coding in an explicit, dependent manner; that is, sparse coding is dependent on the number of sibling units co-active (excluding the current unit):

$$j = \sum_{n \in sib[S_i]} S_n\ , \quad \text{where } n \neq i \tag{14}$$

The first dependent sparse coding function (*Logistic*), uses a modified logistic function to probabilistically limit the number of units active from 0 to ϕn units.

$$\text{Logistic}: \quad sp_2(S_i) = log\tfrac{\ell_{\phi,\mu,n}(j+1)}{\ell_{\phi,\mu,n}(j)} \tag{15}$$

where

$$\ell_{\phi,\mu,n}(j) = 1 - \frac{1}{1 + e^{-\frac{(j/n) - \phi}{\mu}}} \tag{16}$$

The second dependent sparse coding function places a prior activation constraint on the units such that probabilistically ϕn units will be active at any given time. This is accomplished by sampling the unit activation states from the binomial distribution:

$$\text{Binomial}: \quad sp_3(S_i) = log \frac{b_{\phi,n}(j+1)}{b_{\phi,n}(j)} \tag{17}$$

where

$$b_{\phi,n}(j) = \frac{n!}{j!(n-j)!} \cdot \phi^j \cdot (1-\phi)^{n-j} \tag{18}$$

Finally, the last dependent sparse coding function is a mixture of poisson and binomial distributions.

$$\text{Pois + Bin}: \quad sp_4(S_i) = log \frac{\pi p_\gamma(j+1) + (1-\pi) b_{\phi,n}(j+1)}{\pi p_\gamma(j) + (1-\pi) b_{\phi,n}(j)} \tag{19}$$

where $b_{\phi,n}(j)$ is defined in Equation 18 and

$$p_\gamma(j) = e^{-\gamma} \cdot \frac{\gamma^j}{j!} \tag{20}$$

This mixture of distributions has the effect of probabilistically having 0 units active as determined by Equation 20 with probability π, and having ϕn units active with probability $1 - \pi$ as determined by Equation 18.

Weight Estimation

Once we have sampled the activation space, we are in the position to estimate the weights. To control the complexity of the model, a prior is placed on the weights. In using the "noisy OR" function where all weights are constrained to be positive, it is assumed that the weight prior is a product of independent gamma distributions parameterized by α and β. Hence, the objective function we wish the maximize becomes

$$\mathcal{L} = P(D_{1:N}|\mathbf{W})P(\mathbf{W}|\alpha,\beta)$$

Using the maximization step from the EM algorithm, we want to set $\partial\mathcal{L}/w_{ij} = 0$ and solve for w_{ij}. Lewicki and Sejnowski (1997) show this can be accomplished by using the transformations $f_{ij} = 1 - e^{-w_{ij}}$ and $g_i = 1 - e^{-u_i}$ and solving for

$$f_{ij} = \frac{\alpha - 1 + 2f_{ij} + \sum_n S_i^{(n)} S_j^{(n)} f_{ij}/g_j^{(n)}}{\alpha + \beta + \sum_n S_i^{(n)}} \tag{21}$$

It should be noted that in the equation, S_i is the *cause* of S_j. Furthermore, $S^{(n)}$ is the unit's state obtained via Gibbs sampling for the n^{th} input pattern. The sum in the above equation is simply the weighted average of the number of times unit S_i was active when unit S_j was active. The ratio f_{ij}/g_j weights each term in the sum inversely to the number of causes for S_j; if S_i is the sole cause of S_j (meaning that $f_{ij} = g_j$), then the term has full weight.

Method

The Alphabet We adopted Rumelhart and Siple's (1974) featured based alphabet; Each letter was composed of simple visual features such as horizontal, vertical, and oblique lines. We modified the original alphabet by breaking both the top and bottom horizontal line segments into two segments each in order to equalize all line segment lengths.

Figure 2 shows each of the 26 letters overlayed on the 16 base line segments; a "Space" character (no features active) was also presented to the network. A subset of these letters ('SPC',H, I, N, O, S, X, Z) was used in the first two studies and the full alphabet was used for the third study.

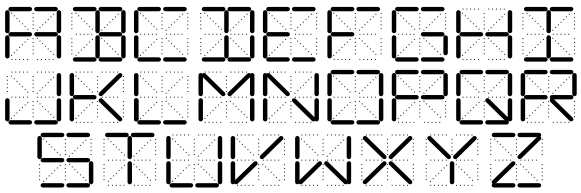

Figure 2: The full alphabet superimposed on the 16 features.

Each line segment was represented by a unary code; therefore, each letter was represented by turning on the appropriate bits in a 16 bit code. Context was also represented using a simple unary scheme; there was one unique unit active for each of the letters within the training set. Thus, there were 8 context representations for the reduced alphabet and 27 context representations for the full alphabet.

Network Architecture and Training The network architecture consisted of 16 surface units, no mediating units, and either 10 or 30 deep units for the reduced and full alphabet training sets respectively. If context was being tested, then the architectures included 8 or 27 context units in accordance with the training set.

For all networks, the weight prior was specified with $\alpha = 1.0$ and $\beta = 1.25$; weights were initialized between 0.05 and 0.15. Internal units were initialized with $P(S_i = 1) = 0.5$. Gibbs sampling was performed 15 times for each input pattern, or until the maximum state change probability was less than 0.05. For the sparse coding experiments, the parameters were set to $\lambda = 1.0$, $\mu = 1.0$, $\pi = 0.5$, and $\gamma = 0.1$; ϕ was set to 0.1 for the first two experiments and 0.05 for the third experiment. It should be noted that the parameters were chosen to maximize network performance (with all things being equal) and a more thorough exploration of parameter space will be required in the future. For each condition, 25 networks were trained with different randomized initial weights.

Results and Discussion

Network performance was analyzed via two methods. First, the generative nature of the models was tested in terms of their ability to reproduce the surface pattern presented. Each pattern was presented to the network and Gibbs sampling was performed to produce an internal pattern of activity at the deep layer. This internal activity was then propagated back to the surface layer units and the number of bits in error—either "Addition" (i.e., 1 instead of 0) or "Omission" (i.e., 0 instead of 1) errors—was calculated. This was performed 100 times for each pattern.

Second, the underlying weight structure of each network was analyzed both qualitatively and quantitatively. The first qualitative measure is based on the visual inspection of the weight matrix as in Lewicki and Sejnowski (1997). The weight for each input feature is passed through Equation 4 to produce a color code fading from "black" to "white" (representing $1 \rightarrow 0$) and then plotted as the appropriate line segment. This type of analysis is shown in Figure 4; unfortunately, it is restricted to single networks. The second is a quantitative measure that can be averaged over runs and is based on the number of weight vectors (i.e., the weights leav-

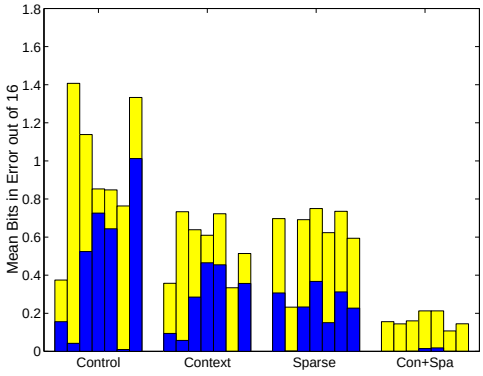

Figure 3: Mean number of errors for the 7 letters across the 4 conditions in Experiment 1. Bottom portion of the bars are 'Addition' errors and upper portions are 'Omission' errors.

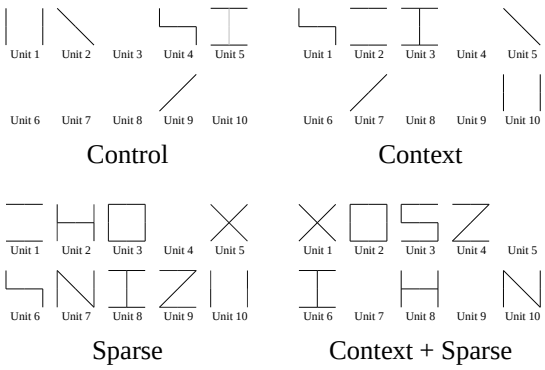

Figure 4: Typical network weights for the context and sparse coding manipulations.

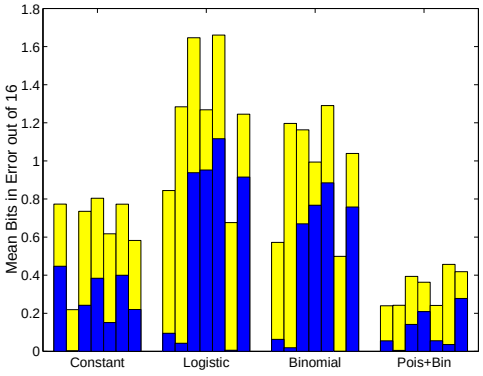

Figure 5: Mean number of errors for the 7 letters across the 4 conditions in Experiment 2. Graphical interpretation is the same as in Figure 3.

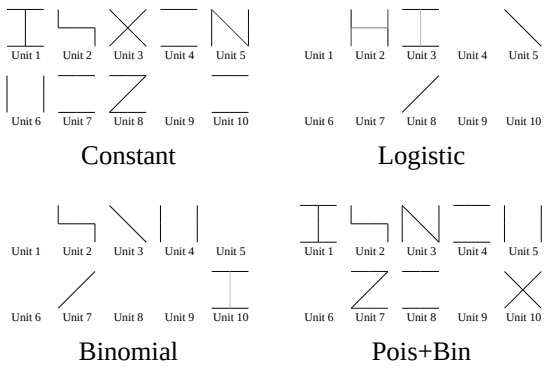

Figure 6: Typical network weights for the sparse coding manipulation.

ing a parent node) that have at least one non-zero element; this measure gives a rough estimation of how many units are being used to represent the data set and therefore local (one unit per pattern) versus distributed encoding.

Experiment 1: Figure 3 shows the mean number of bits in error for the reduced alphabet over the four conditions (only sp_1 with $\lambda = 1.0$, $\phi = 0.1$ was tested in Experiment 1). The mean number of bits in error collapsed over all the letters (excluding 'SPC') for the Control, Context, Sparse, and Context+Sparse conditions are 0.96 (*SD* = 0.36), 0.56 (*SD* = 0.16), 0.62 (*SD* = 0.18), and 0.30 (*SD* = 0.09) respectively.

As can be seen, the standard network performs fairly well on the reduced input set; it averages only 1 bit in error. The addition of the constraints, however, improves the performance of the networks, especially when applied in conjunction. Furthermore, it should be noted that variability in network performance (as indicated by standard deviations) is decreased when constraints are added.

The typical weight structures for the four conditions are shown in Figure 4. As can be seen, the Control, Context, and Sparse conditions tended to extract groups of individual features (indicating a distributed representation) whereas the other condition tends to pick out complete letters (a more localist representation). It should be noted, however, that the Sparse condition appears to redundantly encode information in terms of replicating line segments. Quantitative analyses show that on average, the number of non-zero weight vectors for each of the four conditions are 4.7 (*SD* = 0.8), 5.4 (*SD* = 0.9), 8.4 (*SD* = 1.0), and 7.32 (*SD* = 0.8) respectively. This analysis confirms that a combination of the context and sparse coding constraints encourages local representations of complete letters to develop.

Experiment 2: Figure 5 shows the mean number of error bits for the four different sparse coding functions (Constant $[sp_1]$, Logistic $[sp_2]$, Binomial $[sp_3]$, and Pois+Bin $[sp_4]$). The mean error for each of the four conditions are 0.64 (*SD* = 0.21), 1.23 (*SD* = 0.37), 0.96 (*SD* = 0.31), and 0.34 (*SD* = 0.09) respectively. The first thing to note is that two of the sparse coding functions (sp_2 and sp_3) are worse than or equal to the control condition in Experiment 1. The fourth function (sp_4), however, actually improves performance over the simple, independent sparse coding constraint.

Figure 6 show the typical weight structures for the four sparse coding conditions. It is interesting to note that sp_2 and sp_3 have similar structure (i.e., weaker weights pulling out individual lines) to the control condition in Experiment 1. The

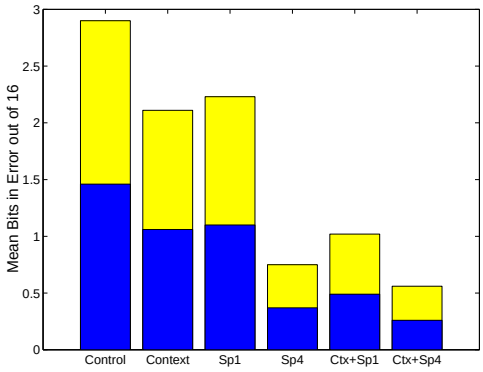

Figure 7: Mean number of bits in error collapsed across letters for the full alphabet plotted for the six conditions.

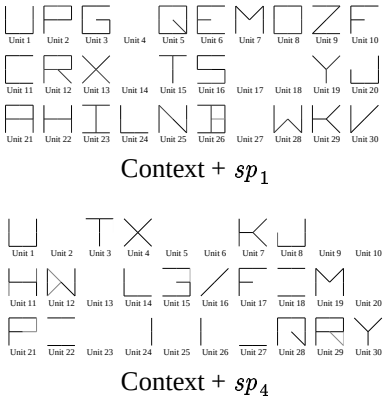

Figure 8: Weights for the fully constrained networks.

other two functions, sp_1 and sp_4, have again pulled out somewhat redundant codings of line segments. This distinction is supported by the quantitative analysis of the weight structure: the mean number of non-zero weight vectors for the four conditions are 8.7 (*SD* = 0.8), 4.3 (*SD* = 0.7), 4.9 (*SD* = 0.9), and 8.0 (*SD* = 0.9).

Experiment 3: In this final experiment, we tested the context and sparse coding constraints on the full alphabet. Only the sp_1 and sp_4 sparse coding constraints were tested. Furthermore, to improve performance, β was reduced to 1.05 and each pattern was only sampled five times per epoch.

Figure 7 shows the average number of bits in error collapsed across all 26 letters for the six conditions (Control, Context, sp_1, sp_4, Context + sp_1, Context + sp_4). Standard deviations for these six conditions were 0.58, 0.62, 0.42, 0.36, 0.34, and 0.27. Moving to the full data set was detrimental to Control, Context, and sp_1 networks; each network tended to have at least one 'Additive' error bit and one 'Omission' error bit for each letter. This was not the case for the three other conditions, with the best performance being produced by the combination of the Context and the sp_4 constraints.

The average number of non-zero weight vectors for each of the six conditions were 25.5 (*SD* = 3.7), 13.8 (*SD* = 6.4), 30.0 (*SD* = 0.0), 18.5 (*SD* = 1.6), 25.0 (*SD* = 1.5), and 19.5 (*SD* = 1.4). The network weight structures for the two fully constrained networks (i.e., Context + Sparse Coding) are shown in Figure 4 (the four other conditions were not graphed as they tended to have smaller weights). As can be seen, combining the context and sparse coding constraints produced networks with distinct weight structures. Once again, it appears that the Context + sp_1 function encourages local encoding. Interestingly, however, the Context + sp_4 function developed a more distributed, redundant encoding.

General Conclusions

The results from these three experiments suggest that a combination of context and sparse coding constraints are required for the formation of adequate internal representations, especially when the data set is large. Moreover, analysis of the weight structure suggests that more accurate performance is due to the development of internal representations that are both distributed and redundant, as opposed to purely local. Although these results are preliminary, they suggest future studies within this generative framework. Specifically, future research will consider networks with mediating layers, and networks trained on words using the feature based letters.

In terms of visual cognition, these results suggest that feedback connections in the brain may provide context information to relatively low-level visual areas, thereby aiding their ability to discover structure in their inputs. Furthermore, sparse coding may be required to create redundant representations that actually increase performance.

References

Anderson, J. R. (1990). *The adaptive character of thought.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Bernardo, J. M., & Smith, A. F. M. (1994). *Bayesian theory.* New York: John Wiley & Sons.

Lewicki, M. S., & Sejnowski, T. J. (1997). Bayesian unsupervised learning of higher order structure. In M. C. Mozer, M. I. Jordan, & T. Petsche (Eds.), *Advances in Neural Information Processing Systems* (Vol. 9, pp. 529–535). Cambridge, MA: MIT Press.

MacKay, D. J. C. (1995). Bayesian methods for supervised neural networks. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks* (pp. 144–149). Cambridge, MA: MIT Press.

McClelland, J. L. (1998). Connectionist models and Bayesian inference. In M. Oaksford & N. Chater (Eds.), *Rational models of cognition.* Oxford: Oxford University Press.

Olshausen, B. A., & Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, *381*, 607–609.

Olshausen, B. A., & Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, *37*, 3311–3325.

Rumelhart, D. E., & Siple, P. (1974). Process of recognizing tachistoscopically presented words. *Psychological Review*, *81*, 99-118.