SANDIA REPORT
SAND2010-6389
Unlimited Release
Printed September 2010

Modeling Cortical Circuits

Fredrick H. Rothganger
Stephen J. Verzi
Brandon R. Rohrer
Patrick G. Xavier

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.osti.gov/bridge

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online order:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2010-6389
Unlimited Release
Printed September 2010

Modeling Cortical Circuits

Fredrick H. Rothganger and Stephen J. Verzi
Cognitive Science and Applications

Brandon R. Rohrer
Intelligent Systems Controls

Patrick G. Xavier
Interactive Systems Simulation & Analysis

Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS1188

Abstract

The *neocortex* is perhaps the highest region of the human brain, where audio and visual perception takes place along with many important cognitive functions. An important research goal is to describe the mechanisms implemented by the neocortex. There is an apparent regularity in the structure of the neocortex [Brodmann 1909, Mountcastle 1957] which may help simplify this task.

The work reported here addresses the problem of *how* to describe the putative repeated units ("cortical circuits") in a manner that is easily understood and manipulated, with the long-term goal of developing a mathematical and algorithmic description of their function. The approach is to reduce each algorithm to an enhanced perceptron-like structure and describe its computation using difference equations. We organize this algorithmic processing into larger structures based on physiological observations, and implement key modeling concepts in software which runs on parallel computing hardware.

Acknowledgements

We thank our collaborators at the Beckman Institute / University of Illinois at Urbana-Champaign. Dr. Stephen Levinson and his students Luke Wendt, Alexander Duda, and Lydia Majure provided stimulating discussions and helpful insights on various approaches to modeling brain processes. Dr. Thomas Anastasio provided expertise in neural modeling and its computational implementation.

Contents

1. Introduction

The neocortex (also known as the cerebral cortex) is the large outer layer that surrounds most of the human brain, and together with its white-matter occupies about 80% of the brain by volume [Zhang & Sejnowski 2000]. It is widely regarded to be the seat of the uniquely powerful human mind. The goal of Sandia's Cognitive Science and Technology thrust is to develop models of human behavior that are based on a scientific understanding of the human mind. This work is motivated by the impact that the human factor has our national security mission, including the safety and surety of nuclear weapons, and how our nation interacts with other cultures and organizations in the world. A complete scientific description of the human mind must necessarily specify the mechanisms of the neocortex.

This task is challenging for a number of reasons. The neocortex is enormously complex, containing (by some estimates) 100+ distinct cell types organized into numerous interconnected structures. Fortunately there appear to be some regularities in its construction. We can begin to gain understanding by focusing on how an arrangement of neurons ("circuit") within a small region interact with each other. The neuroscience community is working intently on this problem, and has already made tremendous progress. At the molecular level, we have a basic understanding of how neurons transduce, transform and transmit signals. We also have some knowledge, at a statistical level, about the connectivity between various types of neurons. However, despite the sheer volume of studies, there are still large gaps in our knowledge in both of these areas. The consequence is that we cannot yet describe the exact structure of the complete cortical circuit. This prevents us from forming any accurate estimate of what function it implements.

The goal of the work presented here was to explore the function of the putative cortical circuit, particularly at the mathematical and algorithmic level. As a result, we have begun to develop techniques that may eventually make the modeling task tractable. Researchers can approach this task from two directions. A "bottom-up" approach looks at neurophysiological details, attempts to assemble a likely circuit, and then infers its functions. A "top-down" approach starts with an assumption about the function of the neocortex, develops a circuit that could implement this, and then correlates it with neurophysiological details. In this work, we tried to combine both approaches, but have emphasized the top-down path.

Why use the language of algorithms to describe circuit function? At any given modeling level, we want to make a transition from the underlying mechanisms to something that is functionally equivalent. A good functional equivalent is something that does the same job, but could possibly be implemented by a different set of mechanisms. In the case of cortical circuits, there is a *de facto* consensus that it is doing some kind of staged information processing. Therefore, a procedural description of the information processing seems to be a reasonable choice for functional equivalent.

The modeling approach consists of two key components: 1) A set of general principles about how models should be organized. These principles are derived from observations about the brain, and from comparisons between different brain subsystems. 2) A restricted language within which to describe the models. The constraints imposed by the language act to filter the plausibility of the models. In addition, we have developed a software framework to support testing models built within this approach. The software runs on parallel hardware in the form of either a shared-memory multiprocessor desktop system or a cluster computer system. The remainder of this report describes the modeling methodology (Section 2) and applies it to a number of published algorithms (Section 3). Finally, it describes the software implementation and tests performed on one of the models (Section 4).

2. Modeling Approach

The approach to modeling cortical circuits consists of two parts. At the most basic level, we represent algorithms as neural circuits that perform various computations on signals that flow through them. At a higher level, we organize these circuits into systems that communicate spatially structured information. In order to give context to the models, the following presentation with start with the larger scale structures and work towards the smaller scale specifics of neural circuits.

2.1. Organizing Cortical Models

2.1.1. Survey of gross brain structure

Figure 2.1 shows a diagram of the brain with some of the major regions identified. Although all the regions of the brain make a contribution to cognition, the particular focus in this work is on the neocortex. In a normal human brain, key functions such as auditory and visual perception, as well as decision making, occur in the neocortex, making an interesting area to study from the perspective of Cognitive Science.

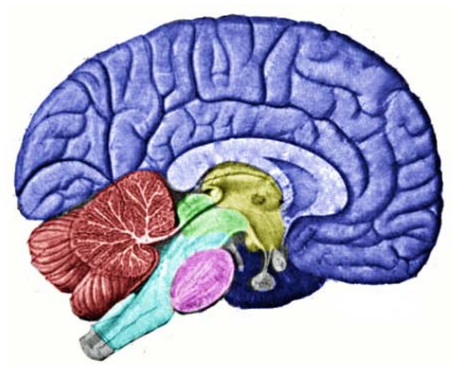


Figure 2.1. The brain, with major regions identified by color. Dark blue is neocortex. Dark red is cerebellum. Yellow is thalamus and hypothalamus. Other colors are brain-stem structures. [Wikipedia, public domain]

The two major cortices of the brain, the neocortex and the cerebellum, both share some comparable structures. They have a regularly organized outer sheet (the term “cortex” means “bark” like that on a tree) and a central core where connections are routed. In the case of the cerebellum, the core consists of the deep cerebellar nuclei and the inferior olive. In the case of the neocortex, the core is the thalamus. The neocortex contains other structures, mainly the basal ganglia, that do not map neatly to anything in the cerebellum.

The focus of this work is primarily on the outer sheet, or “bark” of the neocortex. This ~5mm thick sheet is what we normally refer to as the “gray matter” of the brain, whereas the myelinated axon bundles that interconnect regions within cortex form the “white matter”. The cortical sheet has two distinctive organizational characteristics. First, it appears the all the cells within a radial section (column) work together on a particular piece of the brain’s information processing.

Second, it seems that cells of various types tend to reside at characteristic depths within the sheet.

2.1.2. Layers

Figure 2.2 shows how cells at different depths within the sheet express different genes. The fact that a cell at a particular depth expresses a distinct set of genes suggests that it performs a distinct function from cells at other depths. We use the term "layer" to refer to a population of a particular type of cell and to the depth at which it occurs. There is a layer naming system in common use in the neurophysiology community which came into existence before genetic typing techniques were applied to the neocortex. This naming system still acts as a reference frame for specifying a layer. However, there are many more types of cells than there are named layers, and several types of cell may occur at the same depth.

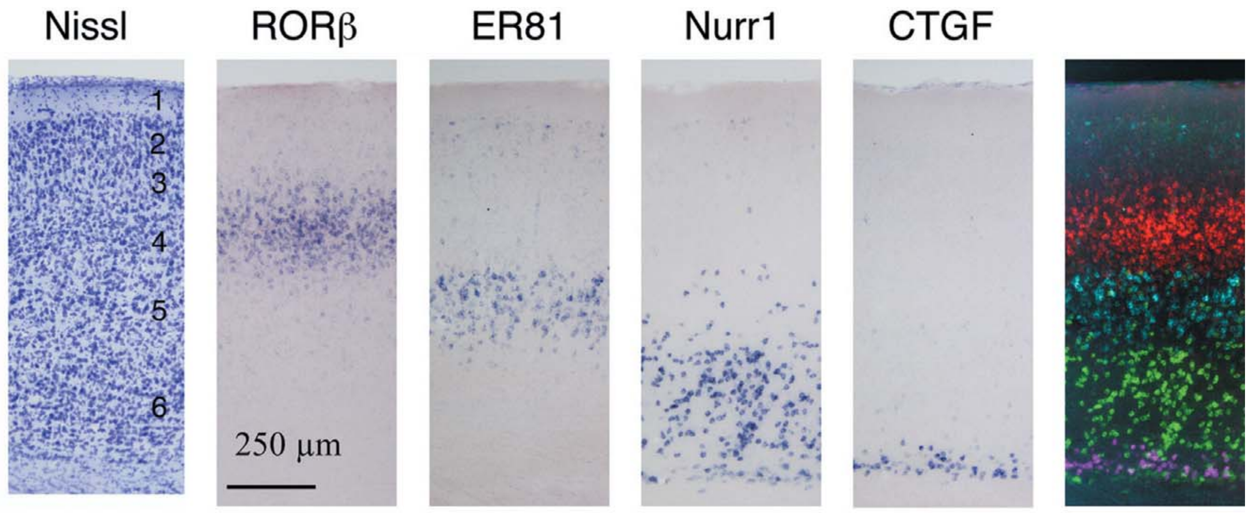


Figure 2.2. Layer-specific gene expression. [Watakabe 2009 – Figure 1a]

The quantity of particular cell types varies from region to region within the neocortex. These variations, and the effect they have on the bulk appearance of a slice of gray matter, are the basis for the Brodmann system of designating areas of the brain [Brodmann 1909]. These variations almost certainly have functional significance. The long-term goal of the kind of work presented here is to find the common algorithmic structures that these cellular elements implement, as well as the algorithmic significance of the variations in structure across the neocortex.

Although cell specialization is an important function of layering, it is not the only one. We propose a hypothesis that one function of layering is to provide pre-defined places for neural processes to meet and exchange information. To motivate this hypothesis, consider two points: the physical constraints on neurons, and an analogy with the structure of the retina.

Neurons are physical objects that have volume and thus use up space. They also need to be in close physical proximity to each other to exchange signals. These two constraints indicate that some volume must be used up making connections, and that all connections cannot occur at the same place. That is, once a connection is made, few or no other neurons can share that connection. These constraints also suggest that neural processes must "find" each other in order to make a connection. The job of finding the desired type of connection can be simplified if there is an agreed upon place where information of a particular type will be delivered/received.

The retina is an example of this type of organization. In particular, the inner-plexiform layer consists of roughly 5 sub-layers. The retinal ganglia project dendritic tufts that spread out in one or a few specific sub-layers. Meanwhile, bipolar cells carrying specific bands of rod/cone output deliver axons into specific sub-layers, associated with the kind of information. This arrangement strongly suggests that the ganglion cells select input layers based on the kind of information they compute.

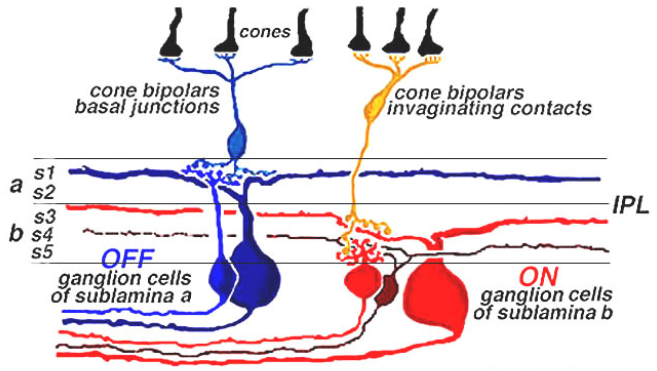


Figure 2.3. Retinal ganglion cells make connections in IPL sublaminae specific to the kind of information they output. [Nelson et al. 1978]

If the hypothesis (that one function of layering is to provide pre-defined places for neural processes to meet and exchange information) is correct, then we predict that the degree of elaboration in layering in a given region of the neocortex is proportional to the number of information channels that it needs to keep separate. This is consistent with what is known about layering in the neocortex: Sensory areas such as V1 have elaborate layering, and presumably keep a wide range of feature types segregated. "Association" areas, on the other hand, tend to have simpler layering, and presumably are fusing information.

Based on the notion that a layer is a place to exchange a channel of information, we model each one as a 2D matrix of numbers. However, since a computational model is never an exact reflection of cytoarchitecture, a matrix is not exactly equal to a physiological layer. In particular, the goal of a model layer is to represent a particular cell type population, so several matrices may be associated with the same physiological layer.

Each matrix is intended to map to a 2D sheet tangential to the cortical surface, that is, it has a spatial interpretation. In particular, suppose a model were processing visual input. The cortical sheet would be retinotopically organized (somewhat like a picture laid over the surface of the brain), and the associated model matrix would also be retinotopically organized. For convenience of organization, each element of a "layer" matrix is generally associated with the output of exactly one model neural unit, and the number of neural units determines the number of elements in the matrix.

2.1.3. Fuzzy Columns

Implicit in the use of matrices to represent layers is the fact that a neural unit must input from one or more matrices as well as output to at least one "home" matrix. In general, a real neuron will spread both its inputs and its outputs over a region, making multiple contacts. For simplicity in modeling, a model neural unit will only output to a single element of a matrix (or a single element in several different matrices in some cases). All spatial spread of signals between neurons is modeled on the input side: a model neural unit will read a region from each of its input matrices, rather than just a single element. The exact spatial arrangement of inputs is encoded in the synaptic weights of the model neural unit.

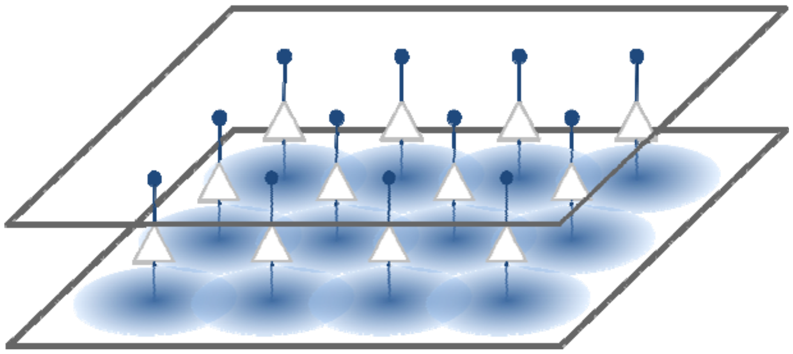


Figure 2.4. Layers and neural units. Spatial dispersion is implemented only on the input side.

The structure created by connecting several populations of model neural units to a set of model layers can be thought of as a model cortical column. The natural question to address next is where to draw boundaries around columns on the model cortical surface. A typical approach in neural network (NN) modeling is the break the area up into units that do not interact with each other except between stages of processing. IE: units may cover overlapping regions of input, and may feed into a common unit in the next stage, but will not share any processing components within a stage. This is a natural organization from a programmer's perspective, because it creates well-defined inputs and outputs for a function that implements the model neural unit.

However, this organization is rather unnatural when compared to a real brain. Each neuron interacts with other neurons within a neighborhood. This neighborhood spreads tangentially to the cortical surface for some limited distance in all directions [Costa & Martin 2010]. It is unlikely that the neighborhood of neurons fit entirely in the boxes defined by variables in a given algorithm. This observation has a profound impact on how we organize models and their software implementation. A model neural unit should contribute to all the computation within its neighborhood, not just a vertically regimented subset. This implies a lateral bleeding of signals in an algorithm that would otherwise keep the same set of values together over several operations.

$$\begin{bmatrix} 1 & 1 & 1 & & & & & & \\ 1 & 1 & 1 & & & & & & \\ 1 & 1 & 1 & & & & & & \\ & & & 1 & 1 & 1 & & & \\ & & & 1 & 1 & 1 & & & \\ & & & 1 & 1 & 1 & & & \\ & & & & & & 1 & 1 & 1 \\ & & & & & & 1 & 1 & 1 \\ & & & & & & 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & & & & & & & \\ 1 & 1 & 1 & & & & & & \\ & 1 & 1 & 1 & & & & & \\ & & 1 & 1 & 1 & & & & \\ & & & 1 & 1 & 1 & & & \\ & & & & 1 & 1 & 1 & & \\ & & & & & 1 & 1 & 1 & \\ & & & & & & 1 & 1 & 1 \\ & & & & & & & 1 & 1 \end{bmatrix}$$

Figure 2.5. Contrast between block diagonal (left) and band diagonal (right) matrices.

In terms of matrix algebra, this is analogous to the distinction between block diagonal and band diagonal matrices. For the sake of discussion, let the rows and columns of a matrix to represent individual variables, and the non-zero elements of matrix to represent interactions between those variables. Then a block diagonal matrix represents independent subsets of the variables, which do not interact with the other subsets. On the other hand, a band-diagonal matrix represents variables that interact with their near neighbors. If we multiply such a matrix by itself several times, which is equivalent to performing several iterations of interactions between the variables, then eventually all the variables interact with each other.

“Fuzzy columns” refers to arrangements in which each model neural unit contributes to multiple processing streams within its neighborhood. (The term “fuzzy column” was coined by Lydia Majure.) The closest analog of fuzzy columns that we are aware of in the literature is convolutional neural networks [LeCun 1999, Jarrett et al. 2009]. In convolutional NNs, the neural units in a given layer are modeled as a single set of weights that are convolved over all positions in the input. In contrast, we model each neural unit with a separate set of weights. On the surface, this would seem to imply that a fuzzy column arrangement would require far more data to learn a representation of the input. However, model neural units share data in overlapping regions, so in practice the difference in the required amount of data is not that much.

2.2. Representing Algorithms

The goal of this work is to represent the function of cortical circuits in a mathematical and algorithmic form. There are two obvious sources of information upon which to build such models. One is neurophysiological studies of the neocortex, and the other is published computational models. In this work, we looked for common themes in existing computational models in order to develop new insight and to avoid re-inventing an existing algorithm. This task was made more difficult by the fact that authors tend to illustrate their algorithms in different ways and to use very detailed differential equations to express the relationships between the variables. To simplify the task of analyzing and comparing algorithms, we developed two tools: a short-hand notation for neural networks, and a method for abbreviating the mathematical forms. One outcome of developing and using these methods was a realization that a similar approach could help us make sense of the neurophysiological evidence as well.

2.2.1. Graphical Notation

Figure 2.6 (bottom left) shows a typical example of a neural network diagram. These diagrams break out all scalar values as separate nodes and draw lines between them to represent connections. The drawback of this type of representation is a profusion of lines and circles that do not add much more to our understanding of the system beyond the first line or circle. Figure 2.6 (top) shows a more concise notation used by some neurophysiologists to represent connectivity between neuron types. This diagram has the right level of abstraction, but it represents shape and spatial information which we don't really need to understand the circuit.

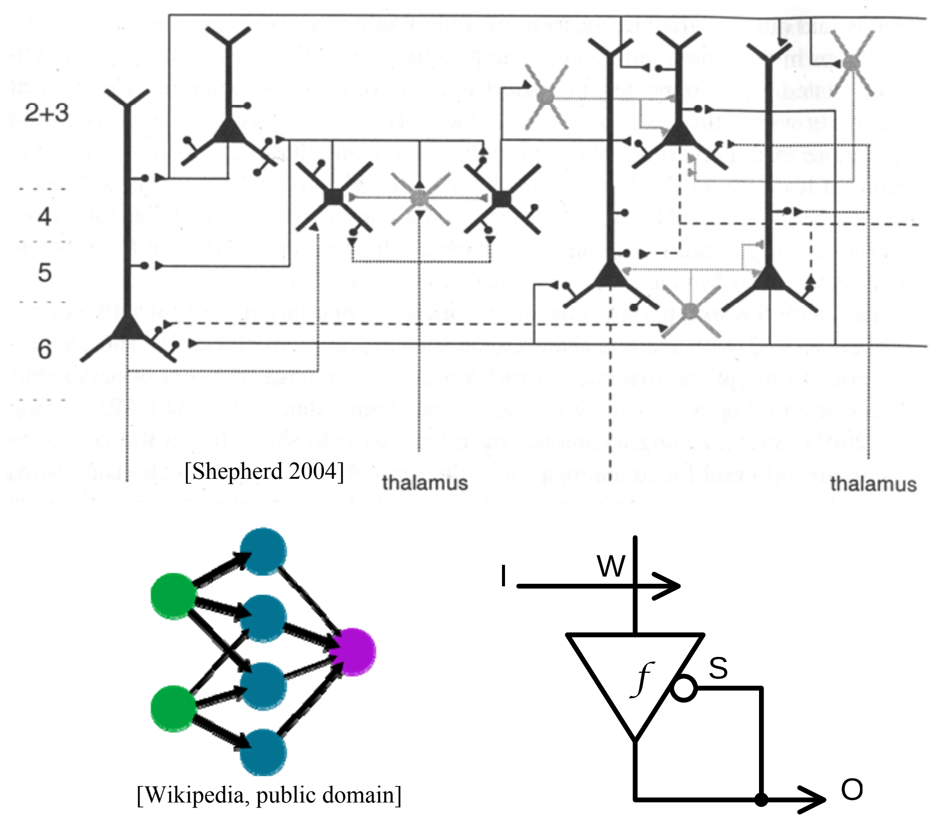


Figure 2.6. Three different neural network notations. Physiological connections (top), traditional perceptron (bottom left), and proposed short-hand (bottom right).

We propose a notation in which all units (the circles in Figure 2.6 bottom left) in a particular layer are grouped together and represented by a single triangle. That is, the triangle stands for an entire population of neural units. The triangle points in the nominal direction of signal flow (which is somewhat the opposite of the normal usage in physiological literature, where a triangle is typically used to represent a pyramidal cell). A line extends out the "back" of the triangle to represent the dendritic arbors of all the units in the population, and another line extends out the point to represent all the axonal outputs. An arrowhead appears on the "axon" line to remind the reader of the nominal signal direction. Lines crossing the dendritic arbor mark at a right angle represent inputs to the population. Such a crossing represents an all-to-all connection between the vector of input values and the set of units in the population. Clearly, this is equivalent to a 2D matrix of connections, and we can note a matrix variable at the crossing point to represent connections weights. Any connection that does not exist has a zero-valued element in matrix.

In real neural tissue, inhibitory inputs to a population can come in several forms. One form comes from local inhibitory cells that impinge on the dendritic arbor. Another form, shunting inhibition, comes from inhibitory cells that wrap multiple synapses around the bodies or axons of the output neurons. The second form of inhibition varies the gain of the whole neuron and can be modeled as a multiplicative factor applied to the total activation of a given unit. In the graphical notation described here, dendritic inhibition would appear simply as negative weights in the input matrix. Shunting inhibition appears as an open circle touching the side of the triangle. See Appendix A for a more complete description of the various neural structures that can be represented in this notation system.

2.2.2. Mathematical Notation

Our goal is to represent the basic relationships between variables with a minimum of notation and clutter. The point of minimizing notation is simply to minimize cognitive load in analyzing and comparing circuit models. The method for simplifying equations presented here works by throwing away carefully selected mathematical details in order to focus on the key underlying structure. The resulting equations are ***not*** generally equivalent to the original equations. Instead, they are an abstract form that can represent any number of different systems. A specific system can be created from the abstract form by filling in particular choices of those mathematical details.

As noted above, the connections from one set of neural units to another set can be represented as a matrix multiply. A typical perceptron unit has the basic mathematical form

$$\mathbf{O} = f(\mathbf{WI}),$$

where **I** is a vector of activities from the input units, **O** is a vector of output activities from the units represented by the equation, **W** is a matrix of connection weights, and *f*() is a non-linear "squashing" function. The job of the squashing function is to keep the output within some range, and to introduce non-linear behavior into an otherwise linear system. Introducing a non-linearity allows additional layers to increase the representational capacity of the system; otherwise, it would simply be a multi-layer linear system, which is equivalent to a single-layer linear system.

The first and most important simplification we propose is to assume an implied non-linearity is always present and therefore not write it explicitly. Effectively, this allows us to write most systems ***as if*** they were strictly linear:

$$\mathbf{O} = \mathbf{WI}.$$

For that reason, this notation could be called "pseudo-linear". Furthermore, a good deal of complexity can be removed from published equation sets by changing from multi-subscripted scalars to matrix notation.

Some models, particularly those that apply shunting inhibition, need to multiply the activity of each unit in a population by a corresponding value in a vector. To represent this in matrix notation, we use an extended form of the Hadamard operator:

$$(\mathbf{A} \circ \mathbf{B})_{rc} = A_{rc} \cdot B_{rc}$$
$$(\mathbf{v} \circ \mathbf{B})_{rc} = v_r \cdot B_{rc}$$
$$(\mathbf{B} \circ \mathbf{v})_{rc} = B_{rc} \cdot v_c$$

where **A** and **B** are matrices, and **v** is a vector. **A** and **B** must have exactly the same dimensions, and **v** must have the same number of rows or columns as **B**, depending on which side it is applied. The first form is the original Hadamard operator, which we rarely use. The second form says that pre-multiplying a matrix with a vector scales its rows, and the third form says that post-multiplying a matrix with a vector scales its columns. We define the Hadamard product to have higher precedence than regular matrix multiplication, but will usually add parenthesis to make the order clear.

Most published equations for neural algorithms appear as differential equations, or on occasion as difference (discrete time) equations. Both forms may be useful when developing a software implementation, depending on the numerical techniques used. However, we can write the equations in a more compact form by changing to *update equation* form. This is the kind of notation used by Donald Knuth and other computer scientists when writing out pseudo-code for algorithms. Our running example would be written as

$$\mathbf{O} \leftarrow \mathbf{WI},$$

where the term on the left (**O**) is at discrete time-step *t*+1 and all terms on the right are at the previous time-step *t*. The equation basically says that in a given execution cycle, the new value of **O** is computed from the current values of **W** and **I**.

In order to translate from differential equations to update equations, it is often necessary to take into account a decay rate in the variable being updated. We prefer representing this as a balanced update using a generic decay rate *L* (for "learning rate"). Here is another form of our running example, in all three notations to show how to move from one to the other:

$$\frac{\partial \mathbf{O}}{\partial t} = \varepsilon \mathbf{WI}$$
$$\mathbf{O}_{t+1} = \mathbf{O}_t + \varepsilon \mathbf{WI}$$
$$\mathbf{O} \leftarrow (1-L)\mathbf{O} + L\mathbf{WI}$$

Here we have translated ε to the generic constant *L*, and added the term -*L***O** to keep the gain of the update equation at unity. The justification for this is that we seek models that are stable under long-term operation. Differential equations in published models often add extra terms just for the purpose of gain control. Part of the simplification process is to recognize and remove these terms in favor of the generic decay rate *L*. Alternately, some of the gain control is folded into the implicit squashing function, whose job is to keep the values within a certain range.

See Section 3 for examples of simplified models, Appendix B for list of simplification patterns, and Appendix C for fully worked simplifications.

2.2.3. Analyzing Physiological Evidence

Up to now, we have been assuming that a restricted class of algorithms can be expressed as a set of differential/difference/update equations. It is also common to describe the dynamics of real neurons using differential equations. This suggests that a bridge can be built between the domain of neural dynamics and the domain of algorithms, and that this could be done in a systematic way. Here is an outline of one possible approach: Collect key pieces of published physiological evidence and translate them into difference equation form, where the key variables are levels of neural activity, synaptic strengths, second messenger activity, etc. The bulk structure of the neural system will be mapped to a set of model layers, and specialized neural types will be assigned to various layers. The difference equations will express interactions between these components. When enough structure is assembled, identify algorithmic motifs within the equation sets.

That difference equations are rich enough to express the key neural structures is evidenced by the fact that differential equations are used to describe the basic functioning of neurons (cf. the Hodgkin-Huxley model). It is worth noting that difference equations are also rich enough to express any finite Turing machine, meaning that they don't simply express a restricted class of algorithms, but rather ***all*** algorithms that might be implemented on a digital computer.

3. Analyzed Algorithms

This section summarizes the analysis of a number of well-known algorithmic models of cortical function. The emphasis here is on understanding the algorithms and comparing them to gain insight. Therefore, we do not present full derivations, but instead refer the reader to the original papers and to Appendix C. The models are presented roughly in order complexity, starting with the simplest.

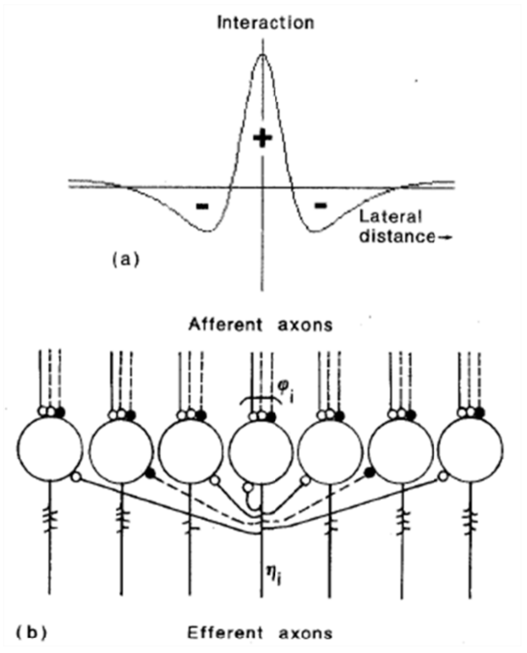
3.1. Self-Organizing Maps (SOM)

Kohonen [1982] proposes a general algorithm for learning feature maps in an on-line manner. These maps effectively perform dimensionality reduction by projecting from the (high) dimensionality of the input space to the (low) dimensionality of the map space. This algorithm is offered as a model of processing in various regions of the brain where features maps emerge. For example, area V1 contains oriented edge detectors that are arranged so that neighboring detectors have similar orientations, and the orientations vary as a function of position on the cortical sheet. Self-organizing maps (SOMs) naturally reproduce maps with these characteristics.

The SOM algorithm in its simplest form involves the following steps:

1) For a given input vector, select the vector in the map that is most similar.
2) Update that vector and its close neighbors to be more similar to the input.

Kohonen [1982] provides a neural implementation of this algorithm, which we reproduce below. Kohonen gives these equations at several different levels of detail. We select the simplest version of each that still conveys the full algorithm.



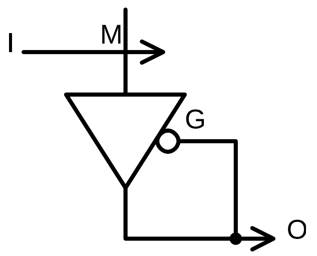
$$\varphi_i = \sum_{j=1}^{n} \mu_{ij} \xi_j$$

$$\eta_i = \sigma\left[\varphi_i(t) + \sum_{k \in S_i} \gamma_k \eta_k (t - \Delta t) \right]$$

$$\frac{d\mu}{dt} = \alpha(\xi - \xi_b)\eta$$

$\sum \mu_{\mathrm{ij}}$ is constant

Figure 3.1. [Kohonen 1982 – Figure 7] and original equations.



$$\mathbf{O} \leftarrow (\mathbf{MI}) \circ (\mathbf{GO})$$
$$\mathbf{M} \leftarrow (1-L)\mathbf{M} + L\mathbf{OI}^T$$

Figure 3.2. Translation and simplified equations.

Circuit operation: The weight matrix **M** stores a set of feature vectors, one per row where each row is associated with a neural unit in the population. When input **I** arrives, some units will be more activated than others due to a better match between their weights and the input pattern. Each unit produces negative feedback that is distributed via a Gaussian weighting to its neighbors. This produces a local winner-take-all effect, where units that closely match **I** remain active, and all others become silent. Furthermore, when multiple units respond they will be separated by some distance that is determined by the neighborhood size of the negative feedback. The circuit adapts its weights using a Hebbian rule: units that are most active will adapt their weights to become more similar to the input, while inactive units will make little or no change. Specifically, each unit updates its weights based on its own current activity (expressed as an element of **O**) and the input vector **I**. This is expressed succinctly as the product of **O** and the transpose of **I**, resulting in a matrix with the same shape and semantics as the weight matrix **M**.

3.2. Douglas & Martin Models

Douglas & Martin [1991, 2004, 2007] present greatly simplified models of the interactions between the major neocortical cell types. The emphasis in these models is on reciprocal and/or recurrent feedback among excitatory neurons, combined with broad regulation from inhibitory neurons. We present these models in graphical form only because the original papers do not offer a mathematical form.

3.2.1. Full interconnection between two populations of excitatory cells and a population of inhibitory interneurons, all driven by thalamic input.

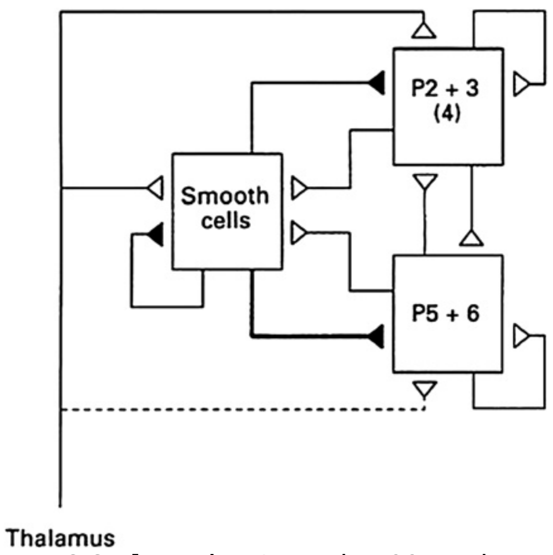


Figure 3.3. [Douglas & Martin 1991 - Figure 5]

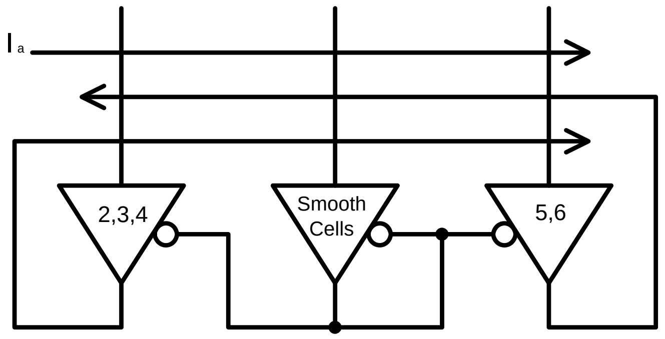
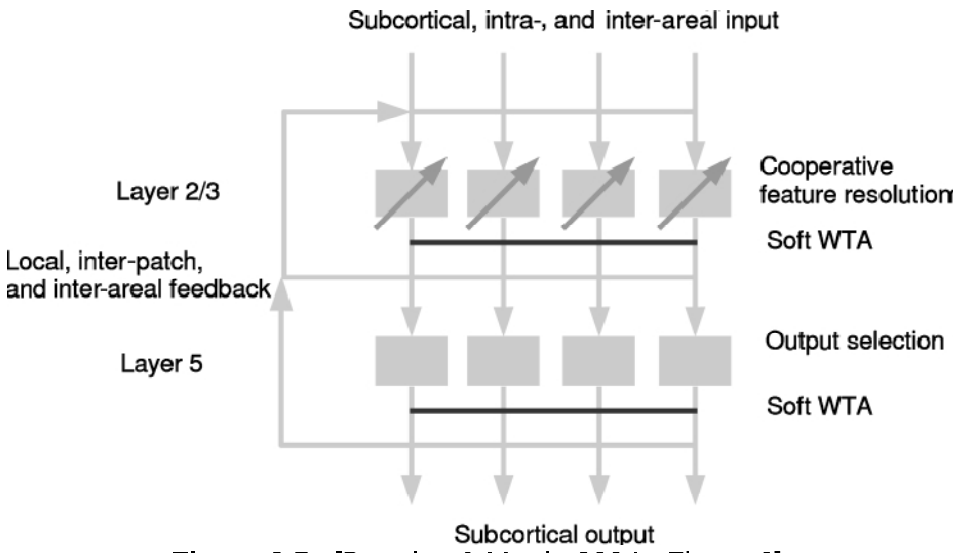


Figure 3.4. Translation.

3.2.2. Winner take all for feature resolution.



Subcortical, intra-, and inter-areal input
Layer 2/3
Cooperative feature resolution
Soft WTA
Local, inter-patch, and inter-areal feedback
Layer 5
Output selection
Soft WTA
Subcortical output

Figure 3.5. [Douglas & Martin 2004 - Figure 6]

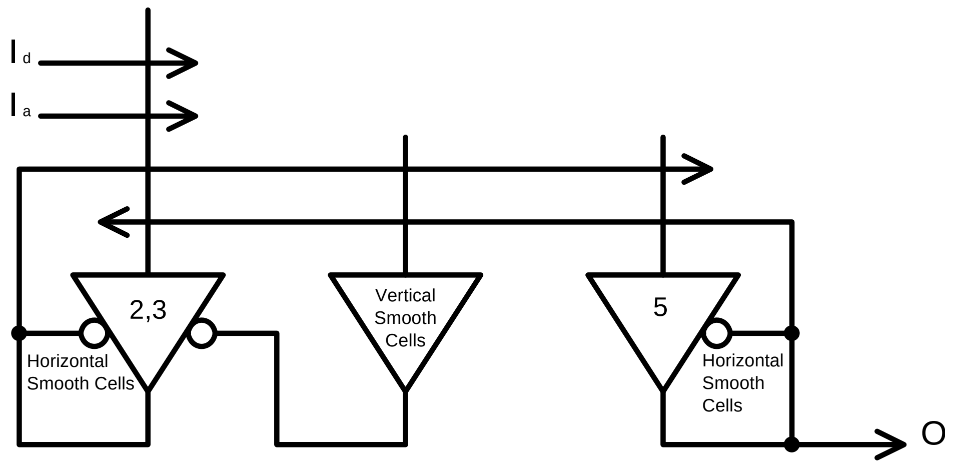


Figure 3.6. Translation.

3.2.3. Linear Threshold Neurons with inhibitory "pointer" neurons.

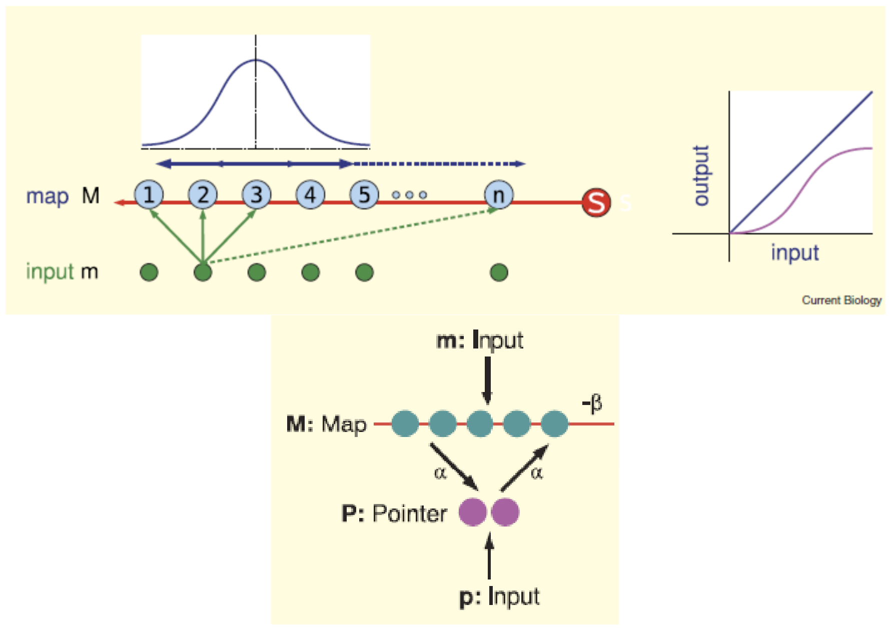
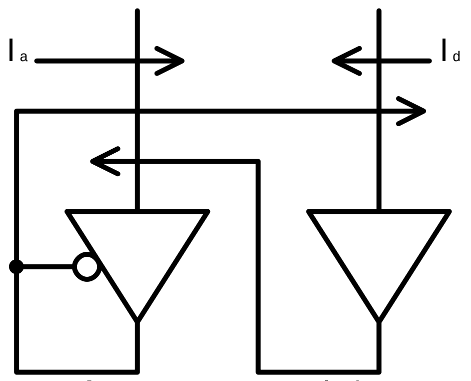


Figure 3.7. [Douglas & Martin 2007 – Figures 2 and 4]

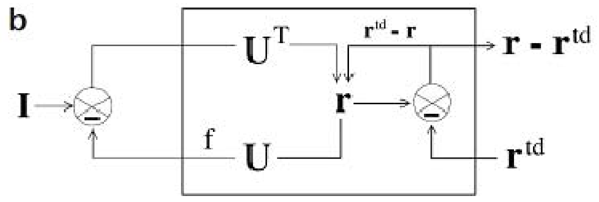


Fiugure 3.8. Translation.

3.3. Kalman Filters (R&B)

Rao & Ballard [1997, 1999] propose a method for learning a sparse code online. The idea of sparse coding [Olshausen & Field 1996] is to select a linear basis for an input space such that each input requires a minimum number of non-zero coefficients (linear mixing weights). Rao [1999] proposes a linear Kalman filter which not only optimizes current state but also the state transition and sensor matrices themselves. He then simplifies this filter by reducing all the associated covariance matrices to scalars. Rao and Ballard [1999] propose an even simpler model which does not use a state transition matrix, but rather updates the current state based solely on input. Here we present the simpler model first along with a small modification we made for better neural plausibility, and then the full model.

3.3.1. Learn a sparse basis set while maintaining an internal representation of input state.



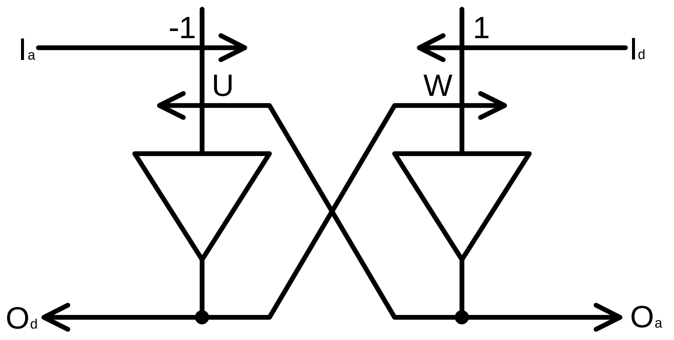
$$\frac{d\mathbf{r}}{dt} = -\frac{k_1}{2}\frac{\partial E}{\partial \mathbf{r}} = \frac{k_1}{\sigma^2} U^T \frac{\partial f}{\partial x}^T (\mathbf{I} - f(U\mathbf{r})) + \frac{k_1}{\sigma_{td}^2}(\mathbf{r}^{td} - \mathbf{r}) - \frac{k_1}{2} g'(\mathbf{r})$$

$$\frac{d\mathbf{U}}{dt} = -\frac{k_2}{2}\frac{\partial E}{\partial \mathbf{U}} = \frac{k_2}{\sigma^2}\frac{\partial f}{\partial x}^T (\mathbf{I} - f(U\mathbf{r}))\mathbf{r}^T - k_2 \lambda \mathbf{U}$$

$$f(x) = \tanh(x)$$

$$g(\mathbf{r}) = \alpha \sum_i r_i^2$$

Figure 3.9. [Rao & Ballard 1999 – Figure 1b] and original equations.



$$\mathbf{O}_d \leftarrow \mathbf{U}\mathbf{O}_a - \mathbf{I}_a$$

$$\mathbf{O}_a \leftarrow \mathbf{W}\mathbf{O}_d + \mathbf{I}_d$$

$$\mathbf{U} \leftarrow (1-L)\mathbf{U} + L\mathbf{O}_d\mathbf{O}_a^T$$

$$\mathbf{W} \leftarrow (1-L)\mathbf{W} + L\mathbf{O}_a\mathbf{O}_d^T$$

Figure 3.10. Translation and simplified equations.

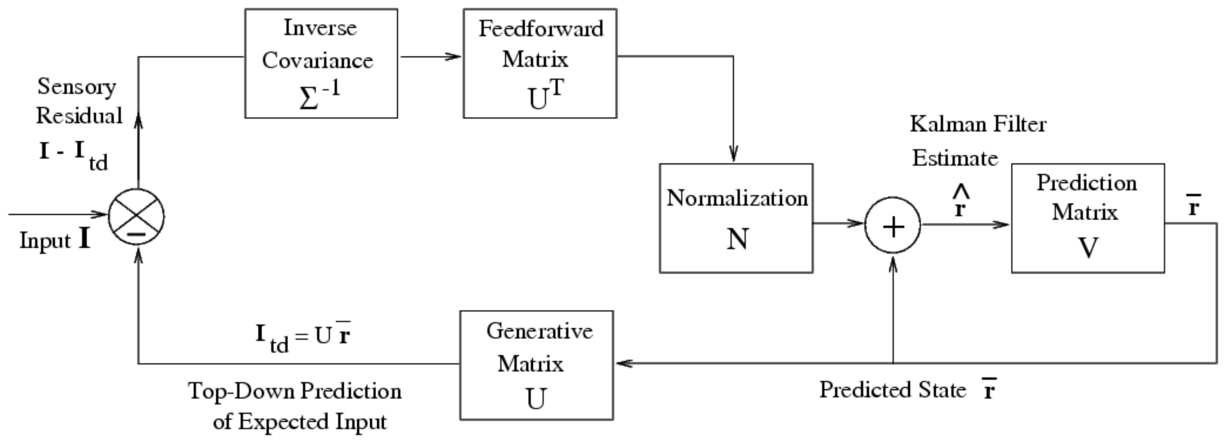
Circuit operation: Input enters the module from both "lower" ($\mathbf{I}_a$) and "higher" ($\mathbf{I}_d$) stages. Likewise, the module outputs to lower ($\mathbf{O}_d$) and higher ($\mathbf{O}_a$) stages. The left-hand population handles prediction generation and differencing with ascending input. It takes the current internal state $\mathbf{O}_a$ and transforms it via the basis vectors $\mathbf{U}$ into a prediction of the input activity. The same population also receives inhibitory input $\mathbf{I}_a$ from the previous stage, effectively subtracting it from the prediction. These differences $\mathbf{O}_d$ are fed back to the previous stage as well as to the right-hand population. The right-hand population handles internal state updates. It transforms the prediction error $\mathbf{O}_d$ into the internal state space via the basis vectors $\mathbf{W}$, and adds in the error feedback $\mathbf{I}_d$ from higher stages. Both populations follow a Hebbian rule for updating their weights. This circuit is essentially a Kalman filter that lacks any notion of tracking uncertainty/covariance between the variables.

The original model proposed by Rao & Ballard had a $\mathbf{U}^T$ in instead of a $\mathbf{W}$. We made this change because it seemed implausible that two neural populations could share the same set of weights, and that conveniently these weights were transposes of each other. Note that $\mathbf{W}$ can easily be learned using Hebb's rule, just as $\mathbf{U}$ can.

Remarkably, this circuit is almost completely symmetric. The key difference between its two halves is the negative weights on $\mathbf{I}_a$. These are also somewhat implausible, although structures such as triadic synapses do exist that could invert inputs. In order to provide error feedback between stages, it is in fact necessary for one of $\mathbf{I}_a$ and $\mathbf{I}_d$ to have negative weights and the other to have positive weights. However, the choice of which one to make negative is not critical. The original version by Rao & Ballard made the weights on $\mathbf{I}_d$ negative, but our version chooses $\mathbf{I}_a$ instead. A more biologically plausible implementation would need to pay attention to the actual types (excitatory versus inhibitory) of top-down and bottom-up connections, and perhaps introduce an extra population of inhibitory units that handle the differencing between predicted and actual inputs.

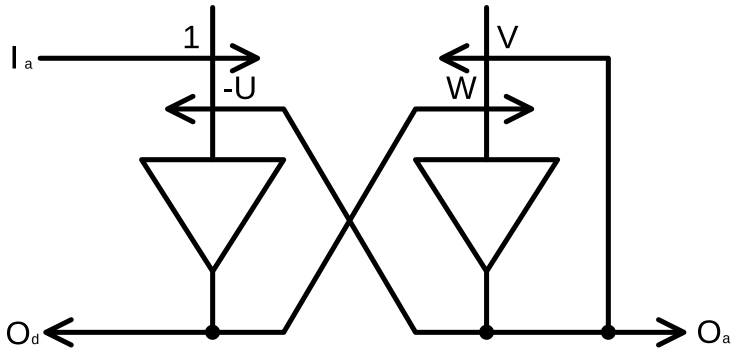
One of the original motivations for this work was to examine whether the Input Minimization learning rule [Anastasio 2001, Rothganger & Anastasio 2009] might apply in the neocortex. Our approach was to start with this circuit by Rao and Ballard, and adapt it to follow the InMin rule. The proof of concept would then be whether such a circuit could reproduce some neocortical phenomenon using the InMin rule. As it turns out, the circuit devised by Rao and Ballard is already an input minimizer. We can see this in two ways. First, notice that the sparse coding rule calls for a minimum number of active elements in the vector that encodes each input. This is effectively an output minimization rule. If the system consists entirely of such modules connected to each other, then output minimization implies that input will be minimized as well. Second, notice that the right-hand population minimizes the error fed back to it. Specifically, it updates its output vector $\mathbf{O}_a$ in the direction indicated by the error feedback vector $\mathbf{I}_d$. When it reaches the target value, $\mathbf{I}_d$ should reduce to zero. Despite these observations, notice that the left-hand population does not explicitly implement an InMin rule, so the Rao & Ballard module as a whole does not fully implement InMin. Because the Rao & Ballard module partially implements the InMin rule, we have not pursued this modeling question further. One possible extension would be to use the perturbative gradient descent mechanism proposed by Rothganger & Anastasio [2009] in the left-hand population.

3.3.2. Learn the state transition model as well as the sensor model.


Inverse Covariance Σ^{-1}
Feedforward Matrix U^T
Sensory Residual $\mathbf{I} - \mathbf{I}_{td}$
Input $\mathbf{I}$
Normalization N
Kalman Filter Estimate $\hat{\mathbf{r}}$
Prediction Matrix V
$\bar{\mathbf{r}}$
$\mathbf{I}_{td} = U\bar{\mathbf{r}}$
Generative Matrix U
Top-Down Prediction of Expected Input
Predicted State $\bar{\mathbf{r}}$

$$\hat{U}(t) = \overline{U}(t) + \alpha\left[\mathbf{I}(t) - \overline{U}(t)\mathbf{r}(t)\right]\mathbf{r}(t)^T$$
$$\hat{V}(t-1) = \overline{V}(t-1) + \beta\left[\mathbf{r}(t) - \mathbf{r}'(t)\right]\hat{\mathbf{r}}(t-1)^T$$
$$\hat{\mathbf{r}}(t) = \mathbf{r}'(t) + \frac{N_0}{\sigma^2}\overline{U}(t)^T\left(\mathbf{I}(t) - \overline{U}(t)\mathbf{r}'(t)\right)$$
$$\mathbf{r}'(t) = \overline{V}(t-1)\hat{\mathbf{r}}(t-1) + \overline{\mathbf{m}}(t-1)$$
with
$$\overline{U}(t) = \hat{U}(t-1)$$
$$\overline{V}(t-1) = \hat{V}(t-2)$$
$$\mathbf{r}(t) = \hat{\mathbf{r}}(t)$$
$$\mathbf{r}'(t) \text{ is synonymous with } \overline{\mathbf{r}}(t)$$

Figure 3.11. [Rao 1999 – Figure 2, and equations 18 – 21]



$$\mathbf{O}_d \leftarrow \mathbf{U}\mathbf{O}_a - \mathbf{I}_a$$
$$\mathbf{O}_a \leftarrow \mathbf{V}\mathbf{O}_a + \mathbf{W}\mathbf{O}_d$$
$$\mathbf{U} \leftarrow (1-L)\mathbf{U} + L\mathbf{O}_d\mathbf{O}_a^T$$
$$\mathbf{V} \leftarrow (1-L)\mathbf{V} + L\mathbf{W}\mathbf{O}_d\,delay(\mathbf{O}_a^T)$$
$$\mathbf{W} \leftarrow (1-L)\mathbf{W} + L\mathbf{O}_a\mathbf{O}_d^T$$

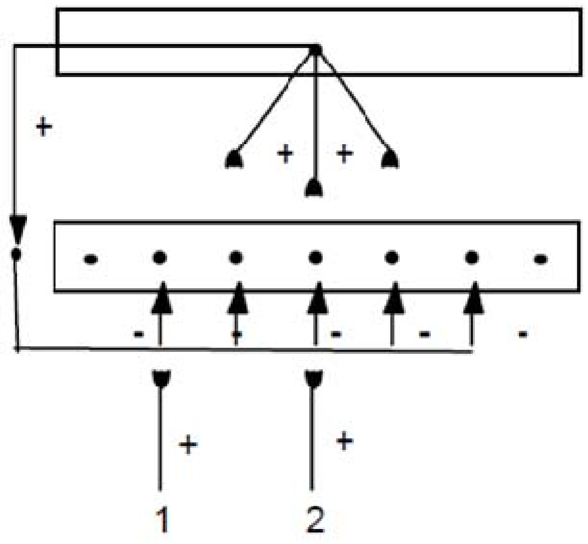
Figure 3.12. Translation and simplified equations.

Circuit operation: Operation is similar to the previous circuit model. The left-hand population does two jobs: It computes the predicted input vector based on current internal state $\mathbf{O}_a$ and the matrix of basis vectors **U**. Then it subtracts this value from the actual input $\mathbf{I}_a$ and sends error value $\mathbf{O}_d$ to the right-hand population and to any previous stage. The left-hand population learns the weights of **U** using a Hebbian rule based on correlations between state $\mathbf{O}_a$ and error $\mathbf{O}_d$. The right-hand population computes an updated state $\mathbf{O}_a$ based on error values $\mathbf{O}_d$ and an inverse basis set **W**. It also maintains its own activity level over time via feedback through the state transition matrix **V**. **V** will generally be close to identity, with some off-diagonal values that contain a model of how the underlying states evolve over time. The right-hand population learns the weights **W** using a Hebbian rule, similar to how the left-hand population learns **U**. The right-hand population also learns the state transition matrix **V** by comparing the computed update $\mathbf{WO}_d$ with the previous state $\mathbf{O}_a$. This is again a Hebbian learning rule, perhaps implemented with spike-timing dependent plasticity (STDP). A physiologically dubious aspect of our neural network interpretation is how the computed update is communicated to the synapses that implement **V**. If the outputs of the right-hand unit are randomly but spatially distributed back onto its own dendritic arbor, and if activities from the synapses implementing **W** can propagate to **V**, then STDP could strengthen those weights that are predictive of the outputs from **W**. We do not know if such an arrangement has ever been observed in the brain.

3.4. Adaptive Resonance Theory (ART)

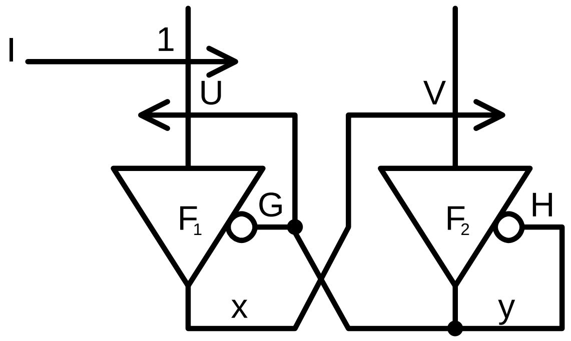
Grossberg, along with various collaborators in students, has developed a model of how neural circuitry may perform clustering in an on-line and stable manner. In that sense, it is another kind of self-organizing map (cf. Kohonen maps). A key characteristic of this theory is the need for positive feedback (resonance) between two populations of units in order to select and learn a particular category. Various versions of the model include other elements, such as a reset circuit to enable searching through multiple candidates. Grossberg [1999] indicates that one of the simpler forms of the ART network is well supported by neural evidence. We present this version here, and then present a more elaborate variant called LAMINART, which interprets observations about the laminar structure of the visual cortex in terms of ART.

3.4.1. Simple ART



$$\varepsilon \frac{d}{dt} x_i = -x_i + (1 - A_1 x_i) J_i^+ - (B_1 + C_1 x_i) J_i^-$$
$$\varepsilon \frac{d}{dt} x_j = -x_j + (1 - A_2 x_j) J_j^+ - (B_1 + C_1 x_j) J_j^-$$
$$V_i = D_1 \sum_j f(x_j) z_{ji}$$
$$J_i^+ = I_i + V_i$$
$$J_i^- = \sum_j f(x_j)$$
$$T_j = D_2 \sum_i h(x_i) z_{ij}$$
$$J_j^+ = g(x_j) + T_j$$
$$J_j^- = \sum_{k \neq j} g(x_k)$$
$$\frac{d}{dt} z_{ij} = K_1 f(x_j) \left[-E_{ij} z_{ij} + h(x_i) \right]$$
$$\frac{d}{dt} z_{ji} = K_2 f(x_j) \left[-E_{ji} z_{ji} + h(x_i) \right]$$
$$K_2 = E_{ji} = E_{ij} = 1$$

Figure 3.13. [Grossberg 1999 – Figure 2C] and equations from [Carpenter & Grossberg 1987]



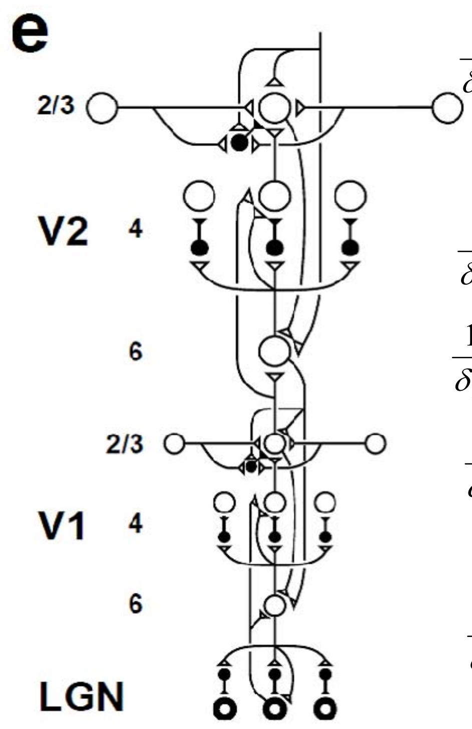
$$\mathbf{x} \leftarrow (1 - \tfrac{1}{\varepsilon})\mathbf{x} + \tfrac{1}{\varepsilon}\left[\mathbf{I} + \mathbf{U}\mathbf{y} - \mathbf{G}\mathbf{y}\right]$$
$$\mathbf{y} \leftarrow \mathbf{y} + \tfrac{1}{\varepsilon}\left[\mathbf{V}\mathbf{x} - \mathbf{H}\mathbf{y}\right]$$
$$\mathbf{V} \leftarrow (1 - \mathbf{y}) \circ \mathbf{V} + \mathbf{y}\mathbf{x}^T$$
$$\mathbf{U} \leftarrow \mathbf{U} \circ (1 - \mathbf{y}) + \mathbf{x}\mathbf{y}^T$$

Figure 3.14. Translation and simplified equations.

Circuit operation: Inputs directly activate their respective units. The weight matrix **V** transforms the current activity pattern **x** into categories **y**. Local inhibition implements a winner-take-all rule on **y**, so that only one member is strongly active. The weight matrix **U** transforms the selected category from **y** into predicted input values. Activity in **y** produces an equal amount of inhibition across all the F_1 units, effectively reducing their gain. Units in F_1 that receive strong input or that are reinforced by **y** will continue to output into **x**, while others will be suppressed. A Hebbian rule guides the learning of the weights of both **U** and **V**. The rate of decay in both **U** and **V** is determined by which category in y is active. The more active a category, the faster its associated weights decay, allowing weights of active categories to change quickly.

Note that the network diagram shows shunting (divisive) inhibition, while the equations show dendritic (subtractive) inihibition. We prefer modeling with shunting inhibition, but dendritic inhibition is the form that falls out of the ART equations. Also note that the variable learning rate in the rules for U and V could be reasonably implemented in real neurons using influx of calcium or other molecular mechanisms for varying level of plasticity based on activity.

3.4.2. Laminar ART (LAMINART)



$$F(z_{ijk},\Gamma) = \max(z_{ijk} - \Gamma, 0)$$

$$\frac{1}{\delta_C}\frac{d}{dt}x_{ijk} = -x_{ijk} + (1 - x_{ijk})\left(\alpha C_{ijk} + \phi F(z_{ijk},\Gamma) + V_{21}x_{ijk}^{V2} + att\right)$$

$$f(x) = \mu\frac{x^n}{\nu^n + x^n}$$

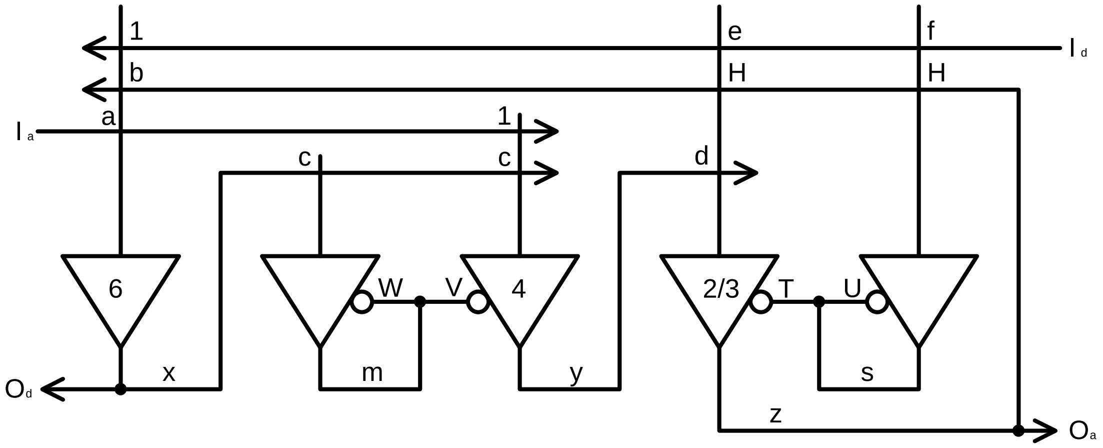
$$\frac{1}{\delta_C}\frac{d}{dt}y_{ijk} = -y_{ijk} + (1 - y_{ijk})(C_{ijk} + \eta^+ x_{ijk}) - (y_{ijk} + 1)\, f\left(\sum_{pqr} W^+_{pgrijk} m_{pqr}\right)$$

$$\frac{1}{\delta_m}\frac{d}{dt}m_{ijk} = -m_{ijk} + \eta^+ x_{ijk} - m_{ijk}\, f\left(\sum_{pqr} W^-_{pgrijk} m_{pqr}\right)$$

$$\frac{1}{\delta_z}\frac{d}{dt}z_{ijk} = -z_{ijk} + (1 - z_{ijk})\left(\lambda[y_{ijk}]^+ + \sum_{pqr} H_{pqrijk} F(z_{pqr},\Gamma) + a^{23}_{excit} att\right) - (z_{ijk} + \psi)\sum_r T^+_{rk} s_{ijr}$$

$$\frac{1}{\delta_s}\frac{d}{dt}s_{ijk} = -s_{ijk} + \sum_{pqr} H_{pqrijk} F(z_{pqr},\Gamma) + a^{23}_{inhib} att - s_{ijk} T^-_{rk} s_{ijr}$$

Figure 3.15. [Raizada & Grossberg 2000 – Figure 6e] and equations from [ibid. – Appendix]



$$\mathbf{O}_d \leftarrow (1-L)\mathbf{O}_d + L\left(a\mathbf{I}_a + b\mathbf{O}_a + \mathbf{I}_d\right)$$
$$\mathbf{y} \leftarrow (1-L)\mathbf{y} + L(\mathbf{I}_a + c\mathbf{O}_d - \mathbf{Vm})$$
$$\mathbf{m} \leftarrow (1-L)\mathbf{m} + L(c\mathbf{O}_d - \mathbf{Wm})$$
$$\mathbf{O}_a \leftarrow (1-L)\mathbf{O}_a + L(d\mathbf{y} + \mathbf{HO}_a + e\mathbf{I}_d - \mathbf{Ts})$$
$$\mathbf{s} \leftarrow (1-L)\mathbf{s} + L(\mathbf{HO}_a + f\mathbf{I}_d - \mathbf{Us})$$

Figure 3.16. Translation and simplified equations.

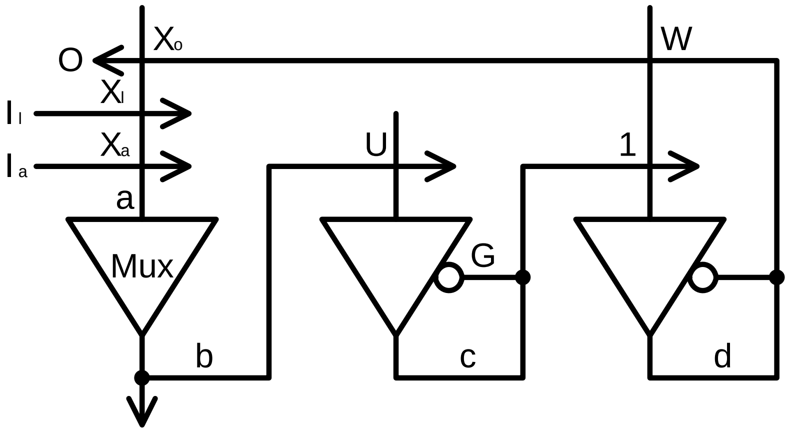
Circuit operation: Populations 6 and 4 both receive the primary input to the module. In addition, population 6 integrates cues from higher stages and from internal feedback. The job of population 6 is to convey this summed activity to population 4. If the constants are tuned correctly, input from population 6 to population 4 will remain under population 4's threshold, but will enhance weak direct input in population 4. Population 2/3 produces the primary output of the module, and also exchanges horizontal connections within itself. These connections spread activity within network, which may for example help complete interrupted contour lines.

All the weights on the connections within this network are constant. There are no on-line learning rules in [Raizada & Grossberg 2000] itself, although it does give reference to other papers that explain how the fixed kernels were calculated.

3.5. Brain Emulating Cognitive Control Architecture (BECCA)

The BECCA model [Rohrer et al. 2009, Rohrer 2010] builds up dictionaries of discrete sensory events and their sequences. BECCA issues action commands, takes in observations (which includes a copy of the actions), and receives rewards based on its performance. BECCA performs feature extraction from its observations using an online clustering method called kx-trees. It works by online divisive partitioning of the state space into regions, based on the local frequency of observations. Each region that contains a sufficiently large number of observations is a feature. As features are observed, they are fed to a sequence learning (S-learning) process. Over time, repeated sequences are weighted more heavily and unrepeated sequences are forgotten. The resulting sequence library constitutes the model of a robot and its environment. Given an initial state, the library also shows which states can be reached by which sequence of actions. Actions are selected based on the expected reward of the predicted states.

Here we present neuralBECCA, a version of BECCA that can be implemented using neural mechanisms. neuralBECCA is not just a model of cortical circuit operation, but also a cognitive architecture. A basic idea of this architecture is that all cortical activity is folded back into its input. This function is assigned to the thalamus and to white-matter connections between regions of the neocortex. The specifics of connectivity between cortical regions are ultimately due to the routing of related signals to the same place.



$$\mathbf{b} \leftarrow \mathbf{Xa}$$
$$\mathbf{c} \leftarrow \mathbf{Ub} - \mathbf{Gc}$$
$$\mathbf{f} \leftarrow \mathbf{c} + \mathbf{Wf}$$
$$\mathbf{X} \leftarrow (1-L)\mathbf{X} + L\mathbf{aa}^T$$
$$\mathbf{U} \leftarrow (1-L)\mathbf{U} + L\mathbf{cb}^T$$
$$\mathbf{W} \leftarrow (1-L)\mathbf{W} + L \cdot delay(\mathbf{f})\mathbf{c}^T$$

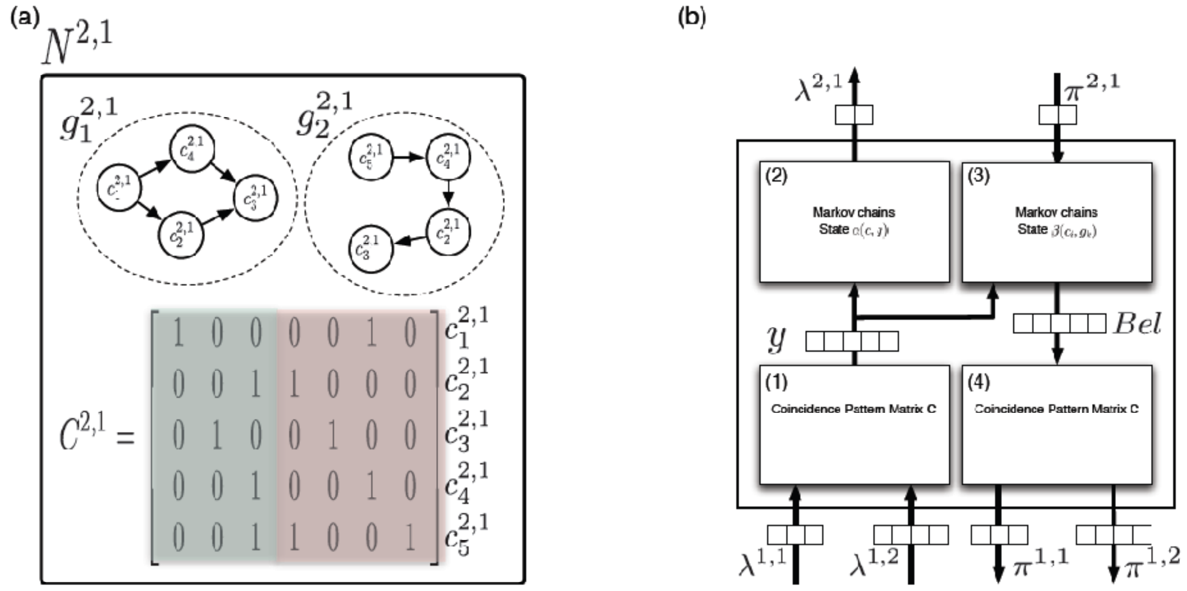
Figure 3.17. Diagram and equations.

Circuit operation: Input to the system come from various sources: folded output (**O**), folded input from peer modules (**I**$_l$), and external input (**I**$_a$). These are combined into a single input vector **a**. The "Mux" unit is not specifically a neural population. Instead, it represents a routing of subsets of the input vector to various modules. The remaining two populations belong to one such module. Notionally, the multiplexing is learned by finding correlations between inputs and routing bundles of correlated values to the same module. The weight matrix **X** represents this routing. **X** is a square matrix constructed from three blocks of weights (**X**$_o$, **X**$_l$, **X**$_a$) associated with the three input sources. The center population does the job of developing a basis set of features and transforming the input into that basis. This may be done any number of ways. Here we illustrate the simplest mechanism already discussed in this section: a Kohonen map. The resulting vector of feature activities **c** then enters the right-hand population, which combines **c** with a prediction of how state will evolve. The right-hand population uses a Hebbian rule to learn the state transition matrix **W** based on its input and a delayed version of its output. Finally, the output is folded back into the input vector.

3.6. Hierarchical Temporal Models (HTM)

Dileep George [2008] proposes a model the broadly follows the ideas of Jeff Hawkins [Hawkins and Blakeslee 2004] regarding a general purpose neocortical algorithm. A key characteristic of this approach is to treat groups of sensory events that tend to occur next to each other in time as a single feature. George observes that the temporal proximity of events is a powerful cue about the structure of the underlying cause. That is, if two events A and B always occur one after the other in a given order, then they are probably due to the same underlying world state.

The structure of the model consists of a step that extracts sensory features and a step that learns Markov chains of such events and groups them to form the temporal features. The sensory feature extractor is essentially a form of on-line clustering or dimensionality reduction. The temporal feature extractor consists of a standard Markov chain learning process (learning the transition matrix), followed by agglomerative clustering using the transition probabilities between events as a distance measure. Modules that implement this process and be assembled in hierarchies, and George proposes a Bayesian framework within which they communicate state information with each other.



$$y_t(i) = P({}^{-}e_t \mid c_i(t)) \propto \prod_{j=1}^{M} \lambda_t^{m_j}(r_i^{m_j})$$

$$\lambda_t^k(g_r) = P({}^{-}e_0^t \mid g_r(t)) \propto \sum_{c_i(t)\in C^k} \alpha_t(c_i, g_r)$$

$$\alpha_t(c_i, g_r) = P({}^{-}e_t \mid c_i(t)) \sum_{c_j(t-1)\in C^k} P(c_i(t) \mid c_j(t-1), g_r)\alpha_{t-1}(c_j, g_r)$$

$$\alpha_0(c_i, g_r) = P({}^{-}e_0 \mid c_i(t=0))P(c_i(t=0) \mid g_r)$$

$$Bel_t(c_i) \propto \sum_{g_r\in G^k} P(g_r \mid {}^{+}e_0)\beta_t(c_i, g_r)$$

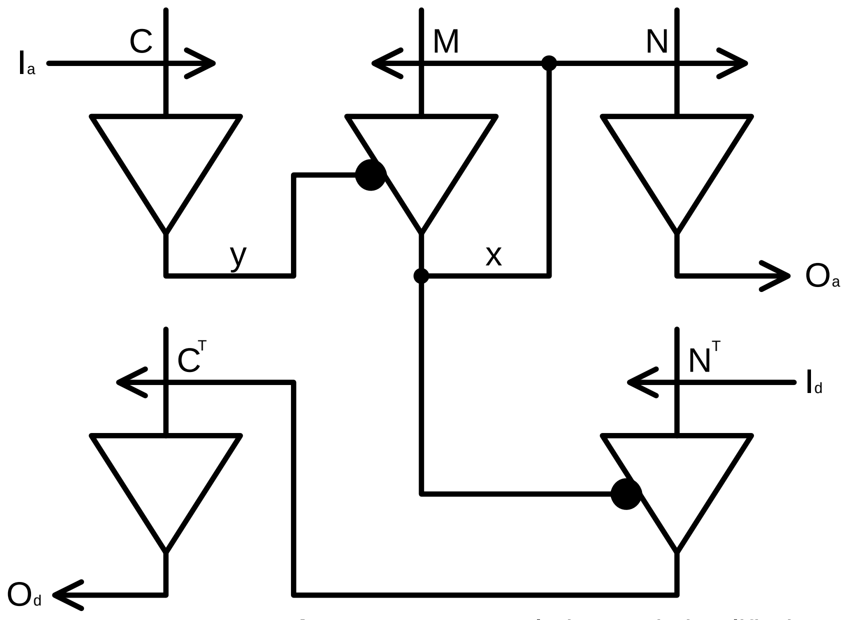
$$\beta_t(c_i, g_r) = P({}^{-}e_t \mid c_i(t)) \sum_{c_j(t-1)\in C^k} P(c_i(t) \mid c_j(t-1), g_r)\beta_{t-1}(c_j, g_r)$$

$$\beta_0(c_i, g_r) = P({}^{-}e_0 \mid c_i(t=0))P(c_i(t=0) \mid g_r, {}^{+}e_0)$$

$$\pi^{child}(g_m) \propto \sum_{i\forall i} I(c_i)Bel(c_i)$$

$$I(c_i) = \begin{cases} 1, \text{if } g_m^{child} \text{ is a component of } c_i \\ 0, \text{otherwise} \end{cases}$$

Figure 3.18. [George 2008 – Figure 4.4] and equations from [ibid. – Table 4.1].



$$
\begin{aligned}
&\mathbf{y} \leftarrow \mathbf{C}\mathbf{I}_a \\
&\mathbf{x} \leftarrow \mathbf{y} \circ \mathbf{M}\mathbf{x} \\
&\mathbf{O}_a \leftarrow \mathbf{N}\mathbf{x} \\
&\mathbf{O}_d \leftarrow \mathbf{C}^T(\mathbf{x} \circ \mathbf{N}^T\mathbf{I}_d) \\
&\mathbf{C} \leftarrow ??? \\
&\mathbf{M} \leftarrow (1-L)\mathbf{M} + L\mathbf{y}\mathbf{x}^T \\
&\mathbf{N} \leftarrow ???
\end{aligned}
$$

Figure 3.19. Translation and simplified equations.

Circuit operation: Inputs $\mathbf{I}_a$ are projected onto the set of basis vectors **C**. The resulting linear mixing weights **y** modulate the internal state **x**. The structure of **x** is a vector where each element represents the probability of being in a particular state in a Markov chain. The next value of **x** is determined by the transition matrix **M**, which encodes the weights of the Markov chain, as well as the values of **y**, which indicate which state(s) are likely based on the input. The weight matrix **N** groups subsets of the elements in **x** so that they act as a single feature. Each feature is represented as an element in the output vector $\mathbf{O}_a$. Top-down input essentially goes through the reverse of this process. $\mathbf{N}^T$ projects a top-down feature vector into subsets of the Markov states. The active subset of states in turn projects onto bottom-up features via $\mathbf{C}^T$.

In [George 2008], all learning is off-line and accomplished by processes that are independent of the run-time architecture. We have attempted to propose on-line learning rules based on the structure of the circuit. However, only a learning rule for the state transition matrix **M** is clearly plausible (based on a Hebbian mechanism). A learning rule for **C** would require reciprocal feedback between the upper-left population and the lower-left population, similar to a Rao & Ballard module (discussed earlier). The basis set **C** could also be learned using a Kohonen module (discussed earlier). It is not clear how the $\mathbf{C}^T$ would be learned in this setup. Finally, we have not yet determined a neurally plausible mechanism for learning the groupings among Markov states represented by **N**.

3.7. Comparisons and Contrasts

3.7.1. SOM and R&B and ART

An ART module is similar to a pair of SOM modules reciprocally connected.

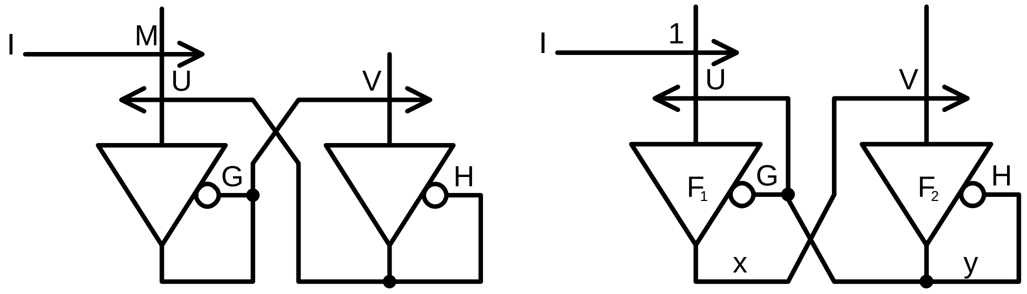


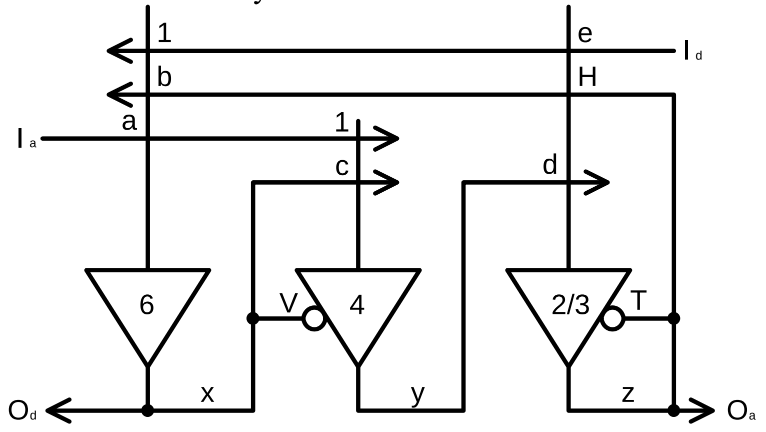
Figure 3.20. Reciprocal SOM (left) compared to ART (right).

The subtle difference between these two is that in ART, all inhibitory activity comes from F2, while in reciprocal SOM, inhibition comes from the other population.

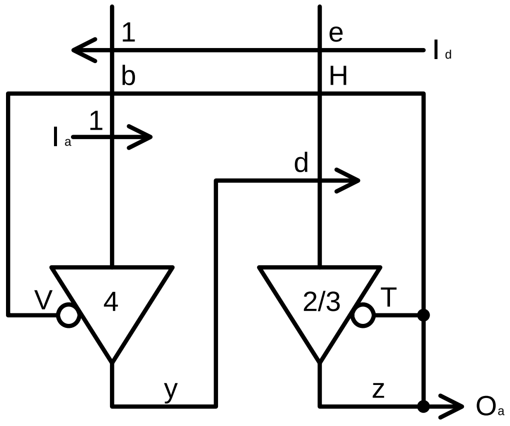
Likewise the R&B model is similar to a pair of reciprocally connected SOM units in that both sides apply a Hebbian rule to learn mappings between input and output. However, unlike either SOM or ART, R&B does not implement a winner-take-all rule via self-inhibition.

3.7.2. ART and LAMINART

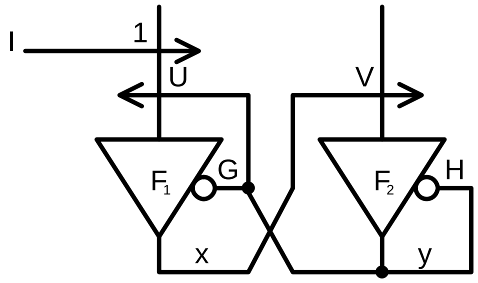
The authors [Raizada & Grossberg 2001] argue that LAMINART is indeed an implementation of the basic ART algorithm. We can see this by performing a series of simplification to the LAMINART circuit diagram. First, remove the inhibitory interneuron populations and treat them as direct inhibition. Note that direct inhibition in a circuit diagram is in general a short-hand for a population of local inhibitory interneurons.



Remove population 6 and route signals directly to population 4.



Compare with the simple ART circuit…



and we can see that the same basic structure of connections is present. The key differences are the addition of the attention signal $\mathbf{I}_d$ and direct positive feedback from population 2/3 to itself.

3.8. Synthesis

The basic characteristic of all the models is a feature extraction stage. Typically, this involves casting a higher-dimensional input space into a lower-dimensional feature space. The feature space is usually learned on-line by detecting correlations between the input and the currently selected feature held internally by the model. This type of processing can be alternately described as vector quantization or dimensionality reduction.

The more sophisticated models (R&B model 2, BECCA, HTM) also include a stage that learns sequential relationships between features. Based on the models studied here, the consensus model of the neocortical circuit would therefore be a feature extraction unit connected to a sequence detection/prediction unit. (Lydia Majure first made this observation regarding the HTM model.)

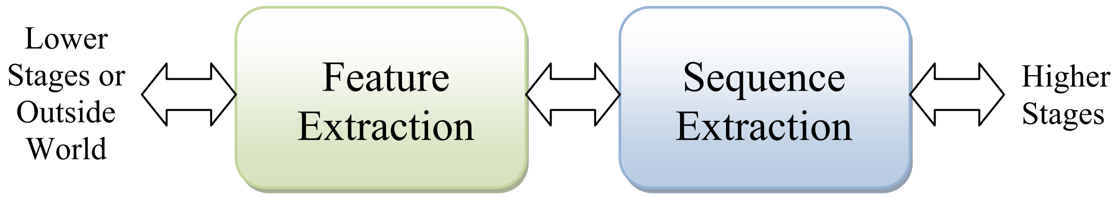


Figure 3.21. Synthesis of models.

Either one of these components is “plug and play” in the sense that a range of algorithms could be substituted into the box to achieve similar results. The simplest model presented here that has both components is R&B model 2.

4. Implementation and Results

We developed a sequence of increasingly sophisticated models, and implemented each one in software in order to characterize its behavior. We selected the Kalman filter model by Rao & Ballard (R&B) as the starting point for this process for a number of reasons. First, it is a simple model and therefore fairly direct to implement. As pointed out in Section 3, the model is the simplest one that incorporates all the elements of the consensus model. Second, it describes a clear role for top-down feedback. Finally, its authors have demonstrated the emergence of receptive field patterns similar to those measured in real animals.

Here we present four stages in the evolution of the model and associated implementation, along with results. We finish by describing the software architecture and the task of moving it to a parallel processing environment.

4.1. Reproduce the results of [Rao & Ballard 1999]

We requested the original software from the authors, but they were unable to provide us with it. Instead they pointed us to a dataset and code by Bruno Ohlshausen [Ohlshausen & Field 1996] which they had used as a starting point. We implemented the algorithm based on the appendix of [Rao & Ballard 1999], and reproduced their results on the emergence of receptive fields. Below is a qualitative comparison of our results against their paper.

Figure 4.1. R&B network architecture and results.

The above diagram illustrates the network architecture used in [Rao & Ballard 1999] and reproduced in our work. Each input image is converted to a "whitened" image (not illustrated), and then a 26x16 pixel patch is extracted from a random location in the image. The first stage of processing consists of three R&B units, which each process a 16x16 pixel sub-window of the patch. These overlapping windows are offset from each other by 5 pixels, which accounts for the total input patch width of 26 pixels (16+5+5). The outputs of the three first-stage units feed to a single second stage unit.

On the left of the figure are qualitative results from the first stage units. At the top is the output of some units in our implementation ("MCC" is an abbreviation for "Modeling Cortical Circuits" and labels the results of this project). In the middle is an image from the original paper. At the bottom is a figure from [Hubel & Wiesel 1959] which illustrates the receptive field of a *simple cell* in V1. These are sensitive to edges of a particular orientation. Such edges are formed by a bright field adjacent to a dark field in the image. Both MCC and R&B learn such receptive field patterns after exposure to natural images.

On the right of the figure are results for the second-stage unit. These reproduce patterns similar to the *complex cells* reported in [Hubel & Wiesel 1968]. In particular, these cells have an end stopping effect, where an edge of an ideal length will produce an optimal response, while an edge that is too long will produce a weaker response. We interpret this as the effect of a checker-board-like pattern of bright and dark field sensitivities. Note that in the original work of Hubel and Wiesel, the bars of light were always in motion. However, in both R&B and this work, we proceed as if the system will respond to stationary light patterns. This is a simplification for working with still images, but it probably reduces the validity of the models.

4.2. Add Layering and Fuzzy Columns

With a working implementation of R&B as a baseline, the next logical step was to implement the new modeling concepts developed in this work and evaluate them. We implemented the layering and fuzzy column structures in two separate steps. The baseline R&B code used direct connections between scalars (essentially pointers) to move data. We rewrote the code to instead communicate by filling and reading matrices. In terms of impact on circuit behavior, this was a fairly minor change. We then rewrote the code to assign each model neural unit to a particular 2D position. Each unit then reads from a sub-matrix surrounding its position. This had a substantial impact on behavior, which we discuss below.

Figure 4.2 shows a somewhat more detailed conceptual diagram of how an R&B module fits together in the new scheme. This illustration treats the input as 1-dimensional, so that all neighborhood relationships can be seen on a single row. All vector values are shown as bundles of lines, one per element. Black dots mark places where an element of some vector interacts with a unit. The inputs, both ascending and descending, map one-to-one with individual units via fixed weights. The interaction between the inputs of one population and the outputs of the other population form a grid which can be thought of as a weight matrix. These inputs are staggered, so that nearby units share some common inputs, while distant units do not. This is the effect of having each unit read from a neighborhood near its assigned position. If this example were extended to 2 dimensions, then the line bundles would represent layer matrices, and the connectivity pattern would be staggered in both dimensions. In the actual code implementation, there is no single weight matrix that covers the interaction between a population of units and one of its input layers. Instead, each individual unit has a representation of just its part of the weights. Effectively, this is a way of encoding a sparse weight matrix, that is, a weight matrix where many or most of the elements are zero.

Figure 4.2. Connection detail for a small R&B circuit, showing two populations interconnecting four layers. Black dots indicate connection, which are staggered to implement fuzzy columns.

Figure 4.3. Network architecture for R&B using layering and fuzzy columns, along with qualitative results.

Figure 4.3 shows the revised network architecture. This looks similar to the original R&B architecture. However, each element in one of the output vectors of the original model is now associated with a distinct unit. These units are spread evenly over the image space in both dimensions. Thus, the indicated number of units is much larger in this version, but each unit is simpler in form. For convenience, we do not exactly translate the dimensions of the original model into the new form. In particular, there are 32 units along the horizontal dimension, rather than 26; and the top level is simply a 2:1 downsample of the bottom level. On the right of the figure are qualitative results in the same format as the baseline. These are somewhat different than the original results, mainly due to the way the units share regions of input. However, they still retain the basic form. First level units detect oriented edges, or contrasts between a light and a dark region. Second level units develop a checkerboard-like pattern which can behave as an endstopped segment detector.

One of the milestones of this work was to test the model on an object recognition task. The first and most obvious approach was to create a deep hierarchy of R&B units using the same basic circuits as the two stage system described above. Each subsequent stage downsampled the image space by roughly 2 until we got to a layer with 100 units (10x10) in it, one per category in a very slightly reduced Caltech 101 dataset. On top of this layer operated a supervised learning module which reinforced one predefined output element per category.

Figure 4.4 (right side) shows a conceptual diagram of the network architecture. At the bottom is the current input image, which passes through successively smaller stages until it reaches the category stage. On the bottom left is a visualization of the Caltech 101 dataset (due to Antonio Torralba). For each category, it shows the average of all the images in that category. Conveniently, most of the images are well-registered, which makes spatially dependent feature recognition feasible. This makes the set well-suited for neural network style recognition systems. The top left shows a snapshot of the network in operation. The snapshot consists of 4 images side-by-side. The leftmost is the original input image. Next is a processed version of the input that is actually fed to the network. Third from the left is the "error" image coming down from the bottom-most stage of the hierarchy. These middle two images are direct visualizations of the first two layers in the model. They are exactly $\mathbf{I}_a$ and $\mathbf{O}_d$ of the first R&B stage (the one that is 300x300 units in size). The rightmost image in the snapshot is the top 10x10 layer ($\mathbf{O}_a$ from the top stage).

Unfortunately, this system was not able to develop stable representations, which is a prerequisite for associating network states with categories. There are a number of network parameters that we could adjust that may enable this arrangement to work. However, we chose instead to develop a more sophisticated model that incorporates saccades. The hope was that a model that could focus on preferred parts of the input would develop stable representations.

Figure 4.4. First approach to object recognition. (Top) Snapshot of network in operation. (Lower left) Summary of input data. (Right) Network architecture.

4.3. Generate Saccades

One of the motivations for developing a saccade model was the generation of stable representations, but it was not the only one. A saccade model is interesting in its own right as a model of perceptual saliency. Saccades are very high speed eye movements that humans produce when looking at a scene. These movements are nearly subconscious and often terminate on points that have explainable significance. The goal of the work presented here was not necessarily to reproduce human-like saccade behavior, but rather to experiment with some possible neural mechanisms that could underlie saccade generation.

The basic approach is to treat the error output ($\mathbf{O}_d$) from the bottom stage of the network as an indication of saliency. The error output is effectively a difference between the actual input image and the image predicted by the network. Presumably, any failure to predict a part of the image indicates that it is worthy of more attention.

In real brains, a region called the superior colliculus (SC) integrates signals from other areas of the brain in a 2D saccade map. When some part of the map becomes highly activated, and the system is not refractory from a previous saccade, it will initiate a new saccade towards the point in the eye's movement space indicated by the point in the map. When this happens, the activity in the map is quenched, at least partially. It then begins to integrate activity, leading up to another saccade.

We developed a very primitive model of this process, illustrated in Figure 4.5. Rather than inputting the whole image, we select a sub-window that is about 1/3 the size of the image in each dimension. We added a layer that accumulates the error output from the bottom stage in a leaky integrator. The first element in the matrix to exceed the saccade threshold triggers the system to re-center the working window over the associated point in the image. At the same time, the entire saccade layer is cleared to zero.

We implemented the saccade mechanism in a simpler version of the object recognition architecture. In particular, the model only had four stages, all of equal size. The input window was 100x100 pixels, and the input images were on average 300x300 pixels. The network stages themselves were also 100x100 to match the input window. Figure 4.6 shows some of the results from running this model on the Caltech 101 set. The input window is moved to the center of the image at the start of each presentation. After that, it is free to move around under the direction of the saccade map. The red line shows the path the input window followed. Note how the system moved to the center of the eye in the face image and then stayed there. The eye produces a very strong low-level saliency cue. It is also noteworthy that it did *not* move to the eye on the cougar image. This may be due to the nose producing an even stronger cue.

This model did not produce stable representations that are suitable for object recognition. However, it did produce fairly consistent behavior on similar images, suggesting that more development of the network configuration could lead to stable representations.

Figure 4.5. Network architecture for saccade generation.

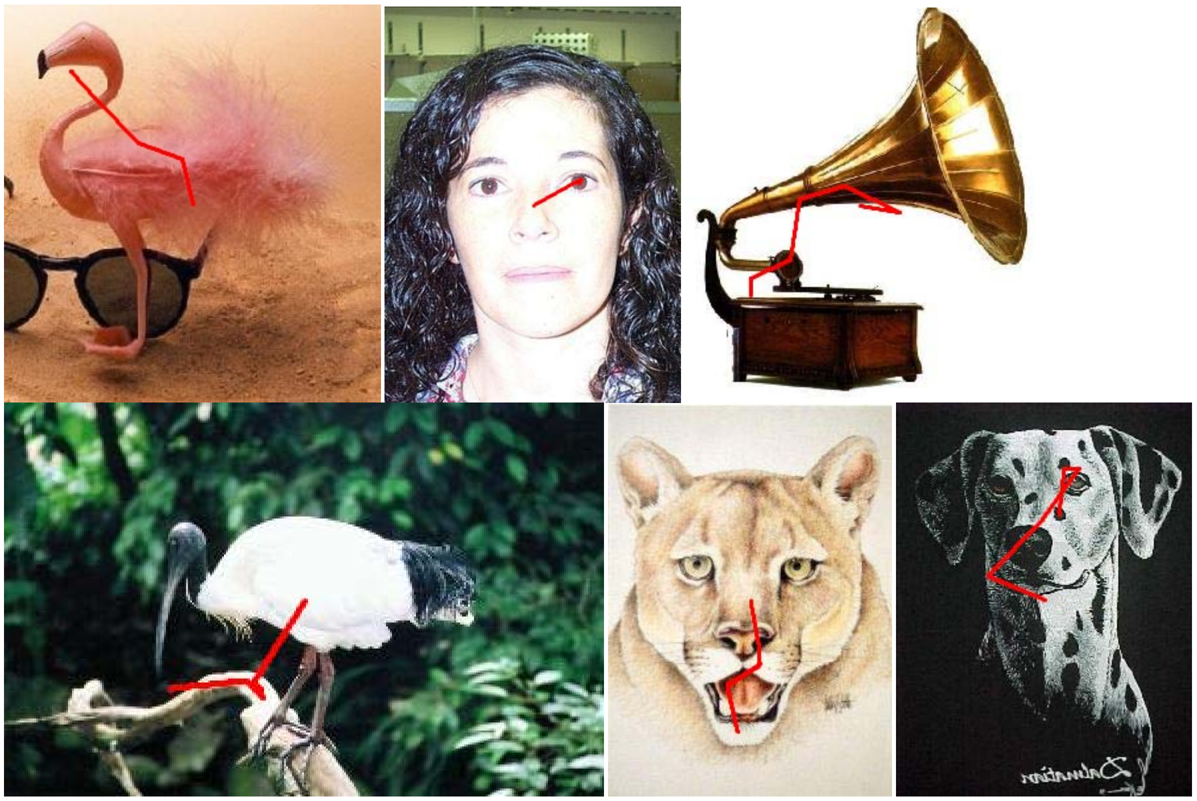


Figure 4.6. Saccade tracks on various input images.

4.4. Add Lateral Connections

In parallel with developing the saccade model, we also developed a model for lateral interactions between units. Again, part of the motivation for doing this was to learn stable representations. We supposed that if related feature could cue each other, than they system could identify an object even when it shifted out of canonical position. Lateral connections are an approach to achieve feature pooling, which enables a recognition system to develop some shift invariance [Jarrett et al. 2009]. In terms of physiology, it is well-known that neurons with particular receptive field types establish lateral connections with neurons of similar type that are in nearby patches [da Costa & Martin 2010].

Figure 4.7 shows the circuit diagram for the lateral connection model. This is an adaptation of both R&B models discussed in Section 3.3. The second R&B model discussed in that section is structured to find temporal sequences, and it lacks a top-down input. The model we present here is more similar to the first R&B model, in that it has a top-down input. We did not implement a delay in the lateral input, so this model does not learn temporal sequences, but rather coincidences between activities in nearby units. The distinction between these two modes is very slight, and should be viewed as a continuum in which one can select a behavior by varying the amount of delay.

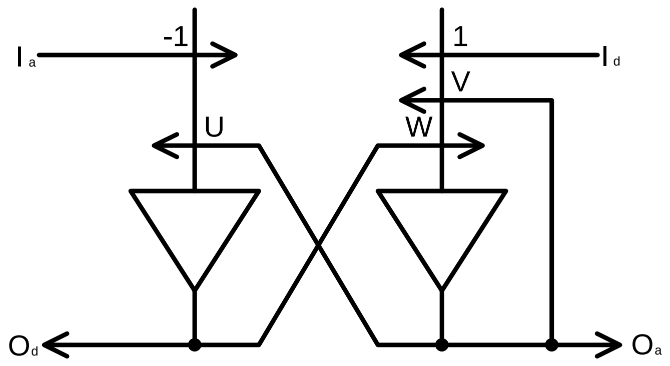


Figure 4.7. Circuit diagram for lateral connections added to R&B.

For a number of reasons, we use an annular kernel (Figure 4.8) to weight the lateral input to a unit. One motivation is to encourage the model to make more distant connections, and another is to avoid feedback. Finally, Kevan Martin and others report "daisy"-like patterns in the lateral connections of the cortex, where a neuron will be connected to other non-adjacent patches reminiscent of petals on a flower. The hope was that this kernel would encourage such a pattern in the model.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & & & & 1 & 1 & 1 \\ 1 & 1 & 1 & & & & 1 & 1 & 1 \\ 1 & 1 & 1 & & & & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 4.8. Annular input pattern.

We implemented and tested this model. Although it runs stably as a whole, after some time there emerge regions within a stage that oscillate at maximum amplitude. This suggests that laterally connected units are driving each other to saturation. This model therefore requires more development to be useful.

4.5. Software Architecture and Parallel Computation

The neural simulations are coded in C++ and follow an object-oriented design. The simulation infrastructure is a set of generic base-classes. Any specific neural unit types are implements as extensions of these classes, and any specific network architectures are built by extension code as well. This enables the simulation infrastructure to be used for a wide range of neural unit types and network structures, not just the ones discussed in this report. The base classes are:

- Network – Contains the entire network. Manages startup and shutdown of the simulation, as well as saving/restoring state.
- Unit – One model neural unit. All simulated neurons extend this class.
- Layer – A matrix of values that are exchanged between populations of neurons. Maintains time coherence of all data within the matrix. Double-buffered so data can be read and written at the same time.
- WorkUnit – A group of Units that execute on a single thread or process.

The Rao & Ballard networks described in this report were implemented by a number of specializations:

- RBnetwork (Network) – The network builder that creates all the various stages of various sizes and interconnects them.
- RBunit (Unit) – Implement half of an R&B module. A complete R&B module consists of two populations of these units reciprocally connected. (See Figures 3.10 and 4.2.)
- RBinput (Unit) – Reads images an inserts them into the bottom stage. Accumulates error feedback and generates saccades.
- RBcategory (Unit) – Implements reinforcement learning by imposing error feedback on top stage of network.
- RBvisualizePyramid (Unit) – Collects information from various Layers and displays it as an image.

The software is designed to run in two distinct parallel-computation environments. On multi-core desktop computers, the software breaks the work up into a number of threads that run in parallel. A separate WorkUnit exists for each piece of the simulation, and each WorkUnit runs on its own thread. A WorkUnit contains some subset of the Units in the simulation. The WorkUnit keeps track of the data dependencies for the Units and ensures that all the input and output Layers are at the same time-step before executing the code in the Units. The Layer objects ensure that all their data have been read or written before moving on to the next step in the simulation. Because there is little interprocess communication overhead on shared memory multiprocessor desktop systems, this code achieves significant speedup and can make use of all available cores.

On cluster computers, the software uses the Basic Linear Algebra Communication Subroutines (BLACS) message passing infrastructure, as opposed to running directly on top of MPI. In this version of the software, a WorkUnit manages all the work within a given process, which typically maps to exactly one compute node. Each process contains a copy of all the Layers that its Units read from or write to. For any given region in any given Layer, there is exactly one process in the system that owns the authoritative copy. At the end of each simulation cycle, that process sends an update message to each other processes that depends on that Layer.

The cluster computing implementation is currently written in a naïve way, and requires further development in order to achieve speedup. An alternate communication scheme may be to formulate the networks directly as sparse matrices and use sparse linear algebra libraries designed for cluster computers. Another future direction may be to run neural simulations on top of Xyce. A neural version of Xyce is currently being developed under another project.

5. Conclusions

We have examined a number of published models at various levels of complexity. Surprisingly, the methodology has exposed critical weaknesses in some highly acclaimed models (in particular the lack of learning rules in HTM and LAMINART). Many of the more sophisticated models examined have some aspect that is implausible, suggesting that there is still a good deal of work to be done in this area. However, the methodology has also shown that some of these models hold promise as both physiologically plausible and computationally effective. Our hope is to synthesize progressively better approximations of the neocortex.

We have implemented two forms of the Kalman filter model (by Rao & Ballard) in software, with significant restructuring to follow the layering and fuzzy column organization patterns, and have successfully reproduced the emergence of receptive fields in early visual processing. We extended the model to generate saccade sequences and experimented with a lateral connection architecture. We did not achieve the goal of object recognition, but see a number of directions forward that could lead to this. One possibility is to retune the network parameters so that oscillations between the higher and lower stages eventually damp out, resulting in stable representations. Another possibility is to treat cyclic trajectories in activity space as representations in their own right. (Alex Duda, one of our university collaborators, is developing general purpose neural models that use cyclic fixed points as representations.)

One of the outcomes of the modeling work is the realization that difference equations may provide a common representation for both “top-down” algorithmic descriptions of neocortical circuit function and “bottom-up” descriptions of neural structure and connectivity. As a common representation, it could provide a bridge between structural detail and the function that structure implements. The neuroscience community is working towards the goal of mapping all the connectivity within the neocortex and understanding its function. Their efforts produce a growing body of evidence about various structures and connections. As the subject of future work, we will develop a method for translating this information into difference equations and look algorithmic motifs in the result. We are hopeful that a new and deeper understanding of neocortical function will emerge.

Appendix A: Representation of Neural Structures

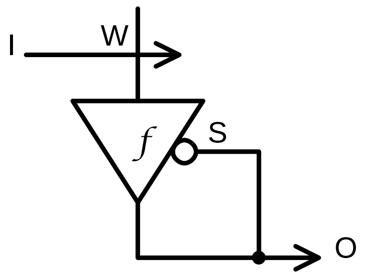


Figure A.1. Illustration of graphical notation.

Axon Bundle – Represented by a line with an arrowhead. The arrow indicates the traditional direction of action potential propagation. Mathematically, an axon bundle is a vector of the activities of the units within its associated population. Axon bundles may be marked with a vector variable.

Axon Collateral – Axon bundles can give off multiple branches. Represented by a line joined to the main axon bundle by a small dot.

Dendritic Arbor – Represented by a vertical line extending from the top of a population icon (triangle).

Dendritic Weights – Represented by an intersection between a dendritic arbor symbol (vertical line) and an axon symbol (horizontal line ending in an arrowhead). In diagrams where it is useful to allow some crossings that are not actual innervations, a large black dot indicates those intersections that are in fact innervations, while a simple crossing with no dot indicates that the arbor and axons do not interact.

The values of the weights are represented by a matrix variable placed near the intersection. It is permissible to have negative weights provided a physiological justification exists for them. Otherwise, avoid negative weights.

An identity matrix represents a one-to-one relationship between neural units and incoming axons. (Climbing fibers in the cerebellum approach this condition by being mutually exclusive, although the number of units innervated is generally on the order of 10 rather than 1.) Because "I" is already reserved for inputs, an identity matrix can be stated with the numeral "1" or "-1".

Local Inhibition – A principal output cell may have local collaterals that synapse onto local inhibitory interneurons. The inhibitory outputs of these neurons in turn synapse onto the principal output cells, sometime even the same ones that deliver the stimulus to the interneurons. Represented by a cell population symbol (triangle) for the interneurons that receives the output of the principal cell population and feeds back to it using an open circle on the side of the principal cell population icon. This can also be abbreviated as a direct feedback from the principal cell population icon to itself, in which case the presence of a population of inhibitory interneurons is generally implied.

Neural Population – A group of neurons with similar characteristics that work together at a particular stage of processing. Represented by a triangle whose point faces downward. The direction of the triangle indicates the forward flow of action potentials in a traditional view of neural behavior. That is, integration in the dendritic arbor followed by an action potential initiated at the hillock followed by an action potential that travels in one direction along the axon. This description is not consistent with observations about the more general behavior of neurons, which includes retrograde action potentials and reciprocal synapses, among other phenomena.

Neurotrophism / Neurotopism – Various patterns of growth, structuring and connection formation, some of which may be information driven. Represented by intersecting an axon bundle with all possible targets and associating a weight matrix with each one. The "learning rule" for each matrix would be based on neurotropic factors. Elements of the respective weight matrices would start at exactly zero, and only become nonzero if a connection is made. Any nonzero elements, however small, represent the existence of a connection.

Non-linear Response – Most details about the behavior of neurons in a given population are abstracted by the population icon (triangle). Notations inside the triangle can express some of these details. For example, a triangle may contain the name of a function which models the behavior of population.

Shunting Inhibition – An input that modulates the gain of a neuron, generally in an inverse manner. That is, the more active the input, the less responsive the neuron becomes. Represented by an open (non-filled) circle attached to the side of a triangle. A matrix variable next to the circle represents the neural weights mapping between the axonal bundle and the units in the population.

Retrograde Action Potential – An action potential that travels in from the soma towards the dendrites. May be involved in Hebbian learning mechanisms. Not represented explicitly in the graphic notation, but implied by mathematical forms that express learning based on relationships between the dendritic inputs and firing activities of a given population.

Spike Timing Dependent Plasticity (STDP) – A potential mechanism for Hebbian learning. If this mechanism plays a significant role in the behavior of a population, then the acronym "STDP" may be placed in small-type inside the triangle the represents the population.

Appendix B: Equation Simplification Patterns

Pattern 1 – Remove any subscripts whose only job is to iterate over elements of respective matrices or vectors.

Rationale – Subscripts add clutter, and thus tend to obscure the meaning of the math. In many cases, standard linear algebra operations directly imply which elements interact with each other. In those cases, the subscripts are truly unnecessary to readers familiar with such conventions. In other cases, it requires only a small abuse of notation to express something in vector form.

Pattern 2 – Remove decorations from the names of values. Attempt to rename to a single symbol.

Rationale – Complex names for a value add clutter. The choice to decorate a symbol is a judgment call. If there are a large number of values of essentially the same type, then it may make sense to use one symbol with a subscript to specify them.

Pattern 3 – Convert difference equations into update equation form.

Rationale – Update equations express the numerical relationships between variables while disposing of details about their time relationships, and are thus easier to read. If all the variables on the right-hand side of a difference equation are at the same time step, and the variable on the left-hand side is at the subsequent time step, then the update equation also preserves the mathematical meaning.

Pattern 4 – Remove any non-linear squashing function from a term.

Rationale – Summarizing the neural units as a linear operation simplifies the notation while preserving the relationships between the variables. Generally, we can assume that every group of neural units has some kind of non-linearity in their outputs, so it is sufficient to leave this non-linearity implicit.

Pattern 5 – Remove normalizations from a term.

Rationale – Normalizations are just another kind of non-linearity, and thus can be absorbed into the implicit non-linearity associated with each term.

Pattern 6 – Express weight updates with a gain of unity.

Rationale – Most weights should maintain some kind of normalization, so generally the determinant of a weight matrix should remain constant, rather than grow or shrink.

A simple form to express a gain of unity is $\mathbf{A} \leftarrow (1-L)\mathbf{A}+L\mathbf{B}$, where L is a scalar that means roughly "learning rate". L may be any value, but is generally a real number in (0,1).

Pattern 7 – Rename these common connections between a local circuit and its neighbors:

I_a – Input ascending, that is, input from a "lower" stage.
I_d – Input descending, that is, input from a "higher" stage
I_l – Input lateral
O_a – Output ascending, that is, output to a "higher" stage
O_d – Ouput descending, that is, output to a "lower" stage
O_l – Output lateral

Appendix C: Derivations

This appendix shows how to derive the various simplified mathematical forms given in Section 3.

C.1. Self-Organizing Maps

Original equations.

$$\boxed{\begin{aligned} &\varphi_i = \sum_{j=1}^{n} \mu_{ij}\xi_j \\ &\eta_i = \sigma\left[\varphi_i(t) + \sum_{k\in S_i} \gamma_k \eta_k (t-\Delta t)\right] \\ &\frac{d\mu}{dt} = \alpha(\xi - \xi_b)\eta \end{aligned}}$$

Change to update equation form. In the equation for η_i, we treat φ_i as if it occurred at t-Δt rather than t. This algorithm is not learning a temporal relationship between φ_i and η_i, so the time difference is mainly to distinguish η_i before and after update.

$$\boxed{\begin{aligned} &\varphi_i \leftarrow \sum_{j=1}^{n} \mu_{ij}\xi_j \\ &\eta_i \leftarrow \sigma\left[\varphi_i + \sum_{k\in S_i} \gamma_k \eta_k\right] \\ &\mu \leftarrow (1-L)\mu + L(\xi - \xi_b)\eta \end{aligned}}$$

Substitute φ into second equation.

$$\boxed{\begin{aligned} &\eta_i \leftarrow \sigma\left[\sum_{j=1}^{n} \mu_{ij}\xi_j + \sum_{k\in S_i} \gamma_k \eta_k\right] \\ &\mu \leftarrow (1-L)\mu + L(\xi - \xi_b)\eta \end{aligned}}$$

Translate matrix multiply to linear algebra form. Rename μ to **M**, γ to **G**, η to **O**, and ξ to **I**. Note that the form of **G** is no longer a simple vector of weights, but rather a matrix that represents this set of weights as seen by each neural unit.

$$\boxed{\begin{aligned} &\mathbf{O} \leftarrow \sigma[\mathbf{MI} + \mathbf{GO}] \\ &\mathbf{M} \leftarrow (1-L)\mathbf{M} + L\mathbf{O}(\mathbf{I} - \mathbf{I}_b)^T \end{aligned}}$$

Drop σ on the assumption that its value is rolled into **M** and **G**. Drop $\mathbf{I}_b$ in favor of allowing **I** to carry negative activity levels.

$$\boxed{\begin{aligned} &\mathbf{O} \leftarrow \mathbf{MI} + \mathbf{GO} \\ &\mathbf{M} \leftarrow (1-L)\mathbf{M} + L\mathbf{OI}^T \end{aligned}}$$

Implement **G** as shunting inhibition rather than a matrix with some negative weights. The shape of **G** would then need to be annular, that is, its center weights would be at or near zero.

$$\boxed{\begin{array}{l}\mathbf{O} \leftarrow (\mathbf{MI}) \circ (\mathbf{GO}) \\ \mathbf{M} \leftarrow (1-L)\mathbf{M} + L\mathbf{OI}^T\end{array}}$$

C.2. Kalman Filters

Original equations.

$$\boxed{\begin{array}{l}\hat{U}(t) = \overline{U}(t) + \alpha\left[\mathbf{I}(t) - \overline{U}(t)\mathbf{r}(t)\right]\mathbf{r}(t)^T \\ \hat{V}(t-1) = \overline{V}(t-1) + \beta\left[\mathbf{r}(t) - \mathbf{r}'(t)\right]\hat{\mathbf{r}}(t-1)^T \\ \hat{\mathbf{r}}(t) = \mathbf{r}'(t) + \frac{N_0}{\sigma^2}\overline{U}(t)^T\left(\mathbf{I}(t) - \overline{U}(t)\mathbf{r}'(t)\right) \\ \mathbf{r}'(t) = \overline{V}(t-1)\hat{\mathbf{r}}(t-1) + \overline{\mathbf{m}}(t-1) \\ \text{with} \\ \overline{U}(t) = \hat{U}(t-1) \\ \overline{V}(t-1) = \hat{V}(t-2) \\ \mathbf{r}(t) = \hat{\mathbf{r}}(t) \\ \mathbf{r}'(t) \text{ is synonymous with } \overline{\mathbf{r}}(t)\end{array}}$$

Make substitutions implied in [Rao 1999].

$$\boxed{\begin{array}{l}\hat{U}(t) = \hat{U}(t-1) + \alpha\left[\mathbf{I}(t) - \hat{U}(t-1)\hat{\mathbf{r}}(t)\right]\hat{\mathbf{r}}(t)^T \\ \hat{V}(t-1) = \hat{V}(t-2) + \beta\left[\hat{\mathbf{r}}(t) - \overline{\mathbf{r}}(t)\right]\hat{\mathbf{r}}(t-1)^T \\ \hat{\mathbf{r}}(t) = \overline{\mathbf{r}}(t) + \frac{N_0}{\sigma^2}\hat{U}(t-1)^T\left(\mathbf{I}(t) - \hat{U}(t-1)\overline{\mathbf{r}}(t)\right) \\ \overline{\mathbf{r}}(t) = \hat{V}(t-2)\hat{\mathbf{r}}(t-1)\end{array}}$$

Rename $\hat{V}(t-1)$ to $\hat{V}(t)$, since it is effectively functioning as a value in the present.

$$\boxed{\begin{array}{l}\hat{U}(t) = \hat{U}(t-1) + \alpha\left[\mathbf{I}(t) - \hat{U}(t-1)\hat{\mathbf{r}}(t)\right]\hat{\mathbf{r}}(t)^T \\ \hat{V}(t) = \hat{V}(t-1) + \beta\left[\hat{\mathbf{r}}(t) - \overline{\mathbf{r}}(t)\right]\hat{\mathbf{r}}(t-1)^T \\ \hat{\mathbf{r}}(t) = \overline{\mathbf{r}}(t) + \frac{N_0}{\sigma^2}\hat{U}(t-1)^T\left(\mathbf{I}(t) - \hat{U}(t-1)\overline{\mathbf{r}}(t)\right) \\ \overline{\mathbf{r}}(t) = \hat{V}(t-1)\hat{\mathbf{r}}(t-1)\end{array}}$$

Substitue $\hat{\mathbf{r}}(t) - \overline{\mathbf{r}}(t)$.

$$\boxed{\begin{aligned}
&\hat{U}(t) = \hat{U}(t-1) + \alpha\left[\mathbf{I}(t) - \hat{U}(t-1)\hat{\mathbf{r}}(t)\right]\hat{\mathbf{r}}(t)^T\\
&\hat{V}(t) = \hat{V}(t-1) + \beta\left[\frac{N_0}{\sigma^2}\hat{U}(t-1)^T\left(\mathbf{I}(t) - \hat{U}(t-1)\bar{\mathbf{r}}(t)\right)\right]\hat{\mathbf{r}}(t-1)^T\\
&\hat{\mathbf{r}}(t) = \bar{\mathbf{r}}(t) + \frac{N_0}{\sigma^2}\hat{U}(t-1)^T\left(\mathbf{I}(t) - \hat{U}(t-1)\bar{\mathbf{r}}(t)\right)\\
&\bar{\mathbf{r}}(t) = \hat{V}(t-1)\hat{\mathbf{r}}(t-1)
\end{aligned}}$$

Substitute $\bar{\mathbf{r}}(t)$.

$$\boxed{\begin{aligned}
&\hat{U}(t) = \hat{U}(t-1) + \alpha\left[\mathbf{I}(t) - \hat{U}(t-1)\hat{\mathbf{r}}(t)\right]\hat{\mathbf{r}}(t)^T\\
&\hat{V}(t) = \hat{V}(t-1) + \beta\left[\frac{N_0}{\sigma^2}\hat{U}(t-1)^T\left(\mathbf{I}(t) - \hat{U}(t-1)\hat{V}(t-1)\hat{\mathbf{r}}(t-1)\right)\right]\hat{\mathbf{r}}(t-1)^T\\
&\hat{\mathbf{r}}(t) = \hat{V}(t-1)\hat{\mathbf{r}}(t-1) + \frac{N_0}{\sigma^2}\hat{U}(t-1)^T\left(\mathbf{I}(t) - \hat{U}(t-1)\hat{V}(t-1)\hat{\mathbf{r}}(t-1)\right)
\end{aligned}}$$

Use $\hat{\mathbf{r}}$ from previous cycle to compute the update of $\hat{U}$. This is the first transformation that actually changes the mathematical meaning of the equations. This is justified on the assumption that input changes slowly enough that it can be adequately predicted by the previous internal state $\hat{\mathbf{r}}$. Alternately, we assume that the time quantum is small, so that there is little change between cycles.

$$\boxed{\begin{aligned}
&\hat{U}(t) = \hat{U}(t-1) + \alpha\left[\mathbf{I}(t) - \hat{U}(t-1)\hat{\mathbf{r}}(t-1)\right]\hat{\mathbf{r}}(t-1)^T\\
&\hat{V}(t) = \hat{V}(t-1) + \beta\left[\frac{N_0}{\sigma^2}\hat{U}(t-1)^T\left(\mathbf{I}(t) - \hat{U}(t-1)\hat{V}(t-1)\hat{\mathbf{r}}(t-1)\right)\right]\hat{\mathbf{r}}(t-1)^T\\
&\hat{\mathbf{r}}(t) = \hat{V}(t-1)\hat{\mathbf{r}}(t-1) + \frac{N_0}{\sigma^2}\hat{U}(t-1)^T\left(\mathbf{I}(t) - \hat{U}(t-1)\hat{V}(t-1)\hat{\mathbf{r}}(t-1)\right)
\end{aligned}}$$

The difference equations are now in a normal form that can be converted to update equations. Specifically, all variables on the right-hand sides (except input) are at time t-1, and all variables on the left-hand sides are at time t. Also remove "hat" notation on estimated values.

$$\boxed{\begin{aligned}
&U \leftarrow U + \alpha\left[\mathbf{I} - U\mathbf{r}\right]\mathbf{r}^T\\
&V \leftarrow V + \beta\left[\frac{N_0}{\sigma^2}U^T\left(\mathbf{I} - UV\mathbf{r}\right)\right]\mathbf{r}^T\\
&\mathbf{r} \leftarrow V\mathbf{r} + \frac{N_0}{\sigma^2}U^T\left(\mathbf{I} - UV\mathbf{r}\right)
\end{aligned}}$$

Substitute standard variable names.

$$\boxed{\begin{aligned}
&\mathbf{U} \leftarrow \mathbf{U} + \alpha \mathbf{O}_d \mathbf{O}_a^T \\
&\mathbf{V} \leftarrow \mathbf{V} + \beta \left[\frac{N_0}{\sigma^2} \mathbf{U}^T \left(\mathbf{I}_a - \mathbf{UVO}_a \right) \right] \mathbf{O}_a^T \\
&\mathbf{O}_a \leftarrow \mathbf{VO}_a + \frac{N_0}{\sigma^2} \mathbf{U}^T \left(\mathbf{I}_a - \mathbf{UVO}_a \right) \\
&\mathbf{O}_d \leftarrow \mathbf{I}_a - \mathbf{UO}_a
\end{aligned}}$$

Convert constants into balanced update equation form. Use "L" as a generic constant that may have different values for each update equation. This again causes a change in mathematical meaning.

$$\boxed{\begin{aligned}
&\mathbf{U} \leftarrow (1-L)\mathbf{U} + L\mathbf{O}_d \mathbf{O}_a^T \\
&\mathbf{V} \leftarrow (1-L)\mathbf{V} + L\mathbf{U}^T \left(\mathbf{I}_a - \mathbf{UVO}_a \right) \mathbf{O}_a^T \\
&\mathbf{O}_a \leftarrow \mathbf{VO}_a + \frac{N_0}{\sigma^2} \mathbf{U}^T \left(\mathbf{I}_a - \mathbf{UVO}_a \right) \\
&\mathbf{O}_d \leftarrow \mathbf{I}_a - \mathbf{UO}_a
\end{aligned}}$$

For neural plausibility, create a new matrix variable **W** to replace $\frac{N_0}{\sigma^2}\mathbf{U}^T$, and provide a separate learning rule for it.

$$\boxed{\begin{aligned}
&\mathbf{U} \leftarrow (1-L)\mathbf{U} + L\mathbf{O}_d \mathbf{O}_a^T \\
&\mathbf{V} \leftarrow (1-L)\mathbf{V} + L\mathbf{W} \left(\mathbf{I}_a - \mathbf{UVO}_a \right) \mathbf{O}_a^T \\
&\mathbf{W} \leftarrow (1-L)\mathbf{W} + L\mathbf{O}_a \mathbf{O}_d^T \\
&\mathbf{O}_a \leftarrow \mathbf{VO}_a + \mathbf{W} \left(\mathbf{I}_a - \mathbf{UVO}_a \right) \\
&\mathbf{O}_d \leftarrow \mathbf{I}_a - \mathbf{UO}_a
\end{aligned}}$$

Simplify the expression $\mathbf{VO}_a$ to $\mathbf{O}_a$. This effectively modifies the Kalman filter gain to use the current state rather than the predicted state, and again we apply the small time quantum assumption to justify this approximation. However, it is necessary to have a time difference between current state $\mathbf{O}_a$ and the correction $\mathbf{W}\left(\mathbf{I}_a - \mathbf{UO}_a\right)$ in order to learn the state transition matrix **V**. Instead of multiplying by **V** to move the correction ahead of $\mathbf{O}_a$, we can use a delayed copy of $\mathbf{O}_a$.

$$\boxed{\begin{aligned}
&\mathbf{U} \leftarrow (1-L)\mathbf{U} + L\mathbf{O}_d \mathbf{O}_a^T \\
&\mathbf{V} \leftarrow (1-L)\mathbf{V} + L\mathbf{WO}_d \, delay(\mathbf{O}_a^T) \\
&\mathbf{W} \leftarrow (1-L)\mathbf{W} + L\mathbf{O}_a \mathbf{O}_d^T \\
&\mathbf{O}_a \leftarrow \mathbf{VO}_a + \mathbf{WO}_d \\
&\mathbf{O}_d \leftarrow \mathbf{I}_a - \mathbf{UO}_a
\end{aligned}}$$

C.3. Adaptive Resonance Theory (ART)

Original equations. Note that we have chosen the simplest configuration implied in the text, rather than the one preferred by Carpenter and Grossberg. Specifically, we do not follow the Weber Law version of weight normalization.

$$\boxed{\begin{aligned}
&\varepsilon \frac{d}{dt} x_i = -x_i + (1 - A_1 x_i) J_i^+ - (B_1 + C_1 x_i) J_i^- \\
&\varepsilon \frac{d}{dt} x_j = -x_j + (1 - A_2 x_j) J_j^+ - (B_1 + C_1 x_j) J_j^- \\
&V_i = D_1 \sum_j f(x_j) z_{ji} \\
&J_i^+ = I_i + V_i \\
&J_i^- = \sum_j f(x_j) \\
&T_j = D_2 \sum_i h(x_i) z_{ij} \\
&J_j^+ = g(x_j) + T_j \\
&J_j^- = \sum_{k \neq j} g(x_k) \\
&\frac{d}{dt} z_{ij} = K_1 f(x_j) \left[-E_{ij} z_{ij} + h(x_i) \right] \\
&\frac{d}{dt} z_{ji} = K_2 f(x_j) \left[-E_{ji} z_{ji} + h(x_i) \right] \\
&K_2 = E_{ji} = E_{ij} = 1
\end{aligned}}$$

Perform all substitutions. Add the assumption that K_1=1.

$$\boxed{\begin{aligned}
&\varepsilon \frac{d}{dt} x_i = -x_i + (1 - A_1 x_i)(I_i + D_1 \sum_j f(x_j) z_{ji}) - (B_1 + C_1 x_i) \sum_j f(x_j) \\
&\varepsilon \frac{d}{dt} x_j = -x_j + (1 - A_2 x_j)(g(x_j) + D_2 \sum_i h(x_i) z_{ij}) - (B_2 + C_2 x_j) \sum_{k \neq j} g(x_k) \\
&\frac{d}{dt} z_{ij} = f(x_j) \left[-z_{ij} + h(x_i) \right] \\
&\frac{d}{dt} z_{ji} = f(x_j) \left[-z_{ji} + h(x_i) \right]
\end{aligned}}$$

Drop non-linearities f(), g() and h(). In the original paper, these are typically defined as quantizing threshold functions. In the case of f(), the definition involves a combinatorial process over several time-steps that would require some very specialized neural structures to implement. Instead of these, we use an implied non-linearity in the outputs of the neural units, hopefully one that naturally occurs.

$$\boxed{\begin{aligned}
&\varepsilon\frac{d}{dt}x_i = -x_i + (1 - A_1 x_i)(I_i + D_1 \sum_j x_j z_{ji}) - (B_1 + C_1 x_i)\sum_j x_j \\
&\varepsilon\frac{d}{dt}x_j = -x_j + (1 - A_2 x_j)(x_j + D_2 \sum_i x_i z_{ij}) - (B_2 + C_2 x_j)\sum_{k\neq j} x_k \\
&\frac{d}{dt}z_{ij} = x_j\left[-z_{ij} + x_i\right] \\
&\frac{d}{dt}z_{ji} = x_j\left[-z_{ji} + x_i\right]
\end{aligned}}$$

Convert to matrix notation. Combine the scalar activities x_j from the upper set of units (that is, the F_2 units, see [Carpenter & Grossberg 1987] for details on nomenclature) into a vector named **y**. Combine the scalar activities x_i from the lower set of units (F_1) into a vector named **x**. Combine the bottom-up weights z_{ij} into a matrix **V**, and combine the top-down weights z_{ji} into a matrix **U**. Define **G** to be a square matrix of all 1s. Define **H** to be a square matrix of all 1s, but with 0s along its main diagonal.

$$\boxed{\begin{aligned}
&\varepsilon\frac{d}{dt}\mathbf{x} = -\mathbf{x} + (1 - A_1\mathbf{x})\cdot(\mathbf{I} + D_1\mathbf{U}\mathbf{y}) - (B_1 + C_1\mathbf{x})\cdot\mathbf{G}\mathbf{y} \\
&\varepsilon\frac{d}{dt}\mathbf{y} = -\mathbf{y} + (1 - A_2\mathbf{y})\cdot(\mathbf{y} + D_2\mathbf{V}\mathbf{x}) - (B_2 + C_2\mathbf{y})\cdot\mathbf{H}\mathbf{y} \\
&\frac{d}{dt}\mathbf{V} = -\mathbf{y}\circ\mathbf{V} + \mathbf{y}\mathbf{x}^T \\
&\frac{d}{dt}\mathbf{U} = -\mathbf{U}\circ\mathbf{y} + \mathbf{x}\mathbf{y}^T
\end{aligned}}$$

Convert to update equation form.

$$\boxed{\begin{aligned}
&\mathbf{x} \leftarrow \mathbf{x} + \tfrac{1}{\varepsilon}\left[-\mathbf{x} + (1 - A_1\mathbf{x})\cdot(\mathbf{I} + D_1\mathbf{U}\mathbf{y}) - (B_1 + C_1\mathbf{x})\cdot\mathbf{G}\mathbf{y}\right] \\
&\mathbf{y} \leftarrow \mathbf{y} + \tfrac{1}{\varepsilon}\left[-\mathbf{y} + (1 - A_2\mathbf{y})\cdot(\mathbf{y} + D_2\mathbf{V}\mathbf{x}) - (B_2 + C_2\mathbf{y})\cdot\mathbf{H}\mathbf{y}\right] \\
&\mathbf{V} \leftarrow \mathbf{V} - \mathbf{y}\circ\mathbf{V} + \mathbf{y}\mathbf{x}^T \\
&\mathbf{U} \leftarrow \mathbf{U} - \mathbf{U}\circ\mathbf{y} + \mathbf{x}\mathbf{y}^T
\end{aligned}}$$

Treat constants *A* and *C* as 0, *D* as 1, and *B* as slightly larger than 1 (see [Carpenter & Grossberg 1987 – Table 1] for constraints on these values). Roll *B* into **G** and **H**.

$$\boxed{\begin{aligned}
&\mathbf{x} \leftarrow \mathbf{x} + \tfrac{1}{\varepsilon}\left[-\mathbf{x} + \mathbf{I} + \mathbf{U}\mathbf{y} - \mathbf{G}\mathbf{y}\right] \\
&\mathbf{y} \leftarrow \mathbf{y} + \tfrac{1}{\varepsilon}\left[\mathbf{V}\mathbf{x} - \mathbf{H}\mathbf{y}\right] \\
&\mathbf{V} \leftarrow \mathbf{V} - \mathbf{y}\circ\mathbf{V} + \mathbf{y}\mathbf{x}^T \\
&\mathbf{U} \leftarrow \mathbf{U} - \mathbf{U}\circ\mathbf{y} + \mathbf{x}\mathbf{y}^T
\end{aligned}}$$

Recombine terms into balanced update.

$$\boxed{\begin{aligned}&\mathbf{x} \leftarrow (1-\tfrac{1}{\varepsilon})\mathbf{x} + \tfrac{1}{\varepsilon}\left[\mathbf{I} + \mathbf{U}\mathbf{y} - \mathbf{G}\mathbf{y}\right]\\ &\mathbf{y} \leftarrow \mathbf{y} + \tfrac{1}{\varepsilon}\left[\mathbf{V}\mathbf{x} - \mathbf{H}\mathbf{y}\right]\\ &\mathbf{V} \leftarrow (1-\mathbf{y}) \circ \mathbf{V} + \mathbf{y}\mathbf{x}^T\\ &\mathbf{U} \leftarrow \mathbf{U} \circ (1-\mathbf{y}) + \mathbf{x}\mathbf{y}^T\end{aligned}}$$

C.4. Laminar ART (LAMINART)

Original equations. Although the appendix of [Raizada & Grossberg 2000] includes equations for retina and LGN, we omit them because they are not part of the repeated cortical circuit, and because they amount to computing an input value **C**. Also, we do not include the equilibrium form of the equations, as they are redundant with the differential form.

$$\boxed{\begin{aligned}&F(z_{ijk},\Gamma) = \max(z_{ijk} - \Gamma, 0)\\ &\frac{1}{\delta_C}\frac{d}{dt}x_{ijk} = -x_{ijk} + (1-x_{ijk})\left(\alpha C_{ijk} + \phi F(z_{ijk},\Gamma) + V_{21}x_{ijk}^{V2} + att\right)\\ &f(x) = \mu\frac{x^n}{\nu^n + x^n}\\ &\frac{1}{\delta_C}\frac{d}{dt}y_{ijk} = -y_{ijk} + (1-y_{ijk})(C_{ijk} + \eta^+ x_{ijk}) - (y_{ijk}+1) f\left(\sum_{pqr} W^+_{pgrijk} m_{pqr}\right)\\ &\frac{1}{\delta_m}\frac{d}{dt}m_{ijk} = -m_{ijk} + \eta^+ x_{ijk} - m_{ijk} f\left(\sum_{pqr} W^-_{pgrijk} m_{pqr}\right)\\ &\frac{1}{\delta_z}\frac{d}{dt}z_{ijk} = -z_{ijk} + (1-z_{ijk})\left(\lambda[y_{ijk}]^+ + \sum_{pqr} H_{pqrijk} F(z_{pqr},\Gamma) + a^{23}_{excit} att\right) - (z_{ijk}+\psi)\sum_r T^+_{rk} s_{ijr}\\ &\frac{1}{\delta_s}\frac{d}{dt}s_{ijk} = -s_{ijk} + \sum_{pqr} H_{pqrijk} F(z_{pqr},\Gamma) + a^{23}_{inhib} att - s_{ijk} T^-_{rk} s_{ijr}\end{aligned}}$$

Remove squashing functions and instead treat them as implicit.

$$\boxed{\begin{aligned}&\frac{1}{\delta_C}\frac{d}{dt}x_{ijk} = -x_{ijk} + (1-x_{ijk})\left(\alpha C_{ijk} + \phi z_{ijk} + V_{21}x_{ijk}^{V2} + att\right)\\ &\frac{1}{\delta_C}\frac{d}{dt}y_{ijk} = -y_{ijk} + (1-y_{ijk})(C_{ijk} + \eta^+ x_{ijk}) - (y_{ijk}+1)\sum_{pqr} W^+_{pgrijk} m_{pqr}\\ &\frac{1}{\delta_m}\frac{d}{dt}m_{ijk} = -m_{ijk} + \eta^+ x_{ijk} - m_{ijk}\sum_{pqr} W^-_{pgrijk} m_{pqr}\\ &\frac{1}{\delta_z}\frac{d}{dt}z_{ijk} = -z_{ijk} + (1-z_{ijk})\left(\lambda y_{ijk} + \sum_{pqr} H_{pqrijk} z_{pqr} + a^{23}_{excit} att\right) - (z_{ijk}+\psi)\sum_r T^+_{rk} s_{ijr}\\ &\frac{1}{\delta_s}\frac{d}{dt}s_{ijk} = -s_{ijk} + \sum_{pqr} H_{pqrijk} z_{pqr} + a^{23}_{inhib} att - s_{ijk} T^-_{rk} s_{ijr}\end{aligned}}$$

Change to linear algebra notation. Treat all triple-subscripted variables as vectors. The spatial structure implied by these indices is mainly encoded in the connection weight matrices. Since the superscript "+" has a meaning in linear algebra, rename variables that have such superscripts: W^+ to **V**, W^- to **W**, T^+ to **T**, and T^- to **U**.

$$\boxed{\begin{aligned}
&\frac{1}{\delta_C}\frac{d}{dt}\mathbf{x} = -\mathbf{x} + (1-\mathbf{x}) \circ \left(\alpha\mathbf{C} + \phi\mathbf{z} + V_{21}\mathbf{x}^{V2} + att\right)\\
&\frac{1}{\delta_C}\frac{d}{dt}\mathbf{y} = -\mathbf{y} + (1-\mathbf{y}) \circ (\mathbf{C} + \eta^+\mathbf{x}) - (1+\mathbf{y}) \circ \mathbf{Vm}\\
&\frac{1}{\delta_m}\frac{d}{dt}\mathbf{m} = -\mathbf{m} + \eta^+\mathbf{x} - \mathbf{m} \circ \mathbf{Wm}\\
&\frac{1}{\delta_z}\frac{d}{dt}\mathbf{z} = -\mathbf{z} + (1-\mathbf{z}) \circ \left(\lambda\mathbf{y} + \mathbf{Hz} + a_{excit}^{23}att\right) - (\psi + \mathbf{z}) \circ \mathbf{Ts}\\
&\frac{1}{\delta_s}\frac{d}{dt}\mathbf{s} = -\mathbf{s} + \mathbf{Hz} + a_{inhib}^{23}att - \mathbf{s} \circ \mathbf{Us}
\end{aligned}}$$

Change to update equation form. Use L as a generic stand-in for various decay rates.

$$\boxed{\begin{aligned}
&\mathbf{x} \leftarrow (1-L)\mathbf{x} + L(1-\mathbf{x}) \circ \left(\alpha\mathbf{C} + \phi\mathbf{z} + V_{21}\mathbf{x}^{V2} + att\right)\\
&\mathbf{y} \leftarrow (1-L)\mathbf{y} + L\left((1-\mathbf{y}) \circ (\mathbf{C} + \eta^+\mathbf{x}) - (1+\mathbf{y}) \circ \mathbf{Vm}\right)\\
&\mathbf{m} \leftarrow (1-L)\mathbf{m} + L(\eta^+\mathbf{x} - \mathbf{m} \circ \mathbf{Wm})\\
&\mathbf{z} \leftarrow (1-L)\mathbf{z} + L\left((1-\mathbf{z}) \circ (\lambda\mathbf{y} + \mathbf{Hz} + a_{excit}^{23}att) - (\psi + \mathbf{z}) \circ \mathbf{Ts}\right)\\
&\mathbf{s} \leftarrow (1-L)\mathbf{s} + L(\mathbf{Hz} + a_{inhib}^{23}att - \mathbf{s} \circ \mathbf{Us})
\end{aligned}}$$

Use standard variable names. In the update equation for x (layer 6 output), two inputs play the role of descending input $\mathbf{I}_d$. One is $\mathbf{x}^{V2}$, the output of layer 6 from the next higher stage. The other is the attention input *att*. Here we take a cue from the network diagram and treat $\mathbf{x}^{V2}$ as a special case of *att*.

$$\boxed{\begin{aligned}
&\mathbf{O}_d \leftarrow (1-L)\mathbf{O}_d + L(1-\mathbf{O}_d) \circ \left(\alpha\mathbf{I}_a + \phi\mathbf{O}_a + \mathbf{I}_d\right)\\
&\mathbf{y} \leftarrow (1-L)\mathbf{y} + L\left((1-\mathbf{y}) \circ (\mathbf{I}_a + \eta^+\mathbf{O}_d) - (1+\mathbf{y}) \circ \mathbf{Vm}\right)\\
&\mathbf{m} \leftarrow (1-L)\mathbf{m} + L(\eta^+\mathbf{O}_d - \mathbf{m} \circ \mathbf{Wm})\\
&\mathbf{O}_a \leftarrow (1-L)\mathbf{O}_a + L\left((1-\mathbf{O}_a) \circ (\lambda\mathbf{y} + \mathbf{HO}_a + a_{excit}^{23}\mathbf{I}_d) - (\psi + \mathbf{O}_a) \circ \mathbf{Ts}\right)\\
&\mathbf{s} \leftarrow (1-L)\mathbf{s} + L(\mathbf{HO}_a + a_{inhib}^{23}\mathbf{I}_d - \mathbf{s} \circ \mathbf{Us})
\end{aligned}}$$

Remove all factors that are present primarily for gain control on the inputs. Instead, treat these as part of the implied non-linearity associated with each unit.

$$\boxed{\begin{aligned}
&\mathbf{O}_d \leftarrow (1-L)\mathbf{O}_d + L(\alpha\mathbf{I}_a + \phi\mathbf{O}_a + \mathbf{I}_d)\\
&\mathbf{y} \leftarrow (1-L)\mathbf{y} + L(\mathbf{I}_a + \eta^{+}\mathbf{O}_d - \mathbf{Vm})\\
&\mathbf{m} \leftarrow (1-L)\mathbf{m} + L(\eta^{+}\mathbf{O}_d - \mathbf{Wm})\\
&\mathbf{O}_a \leftarrow (1-L)\mathbf{O}_a + L(\lambda\mathbf{y} + \mathbf{HO}_a + a^{23}_{excit}\mathbf{I}_d - \mathbf{Ts})\\
&\mathbf{s} \leftarrow (1-L)\mathbf{s} + L(\mathbf{HO}_a + a^{23}_{inhib}\mathbf{I}_d - \mathbf{Us})
\end{aligned}}$$

Rename constants to lower-case Roman letters, in order starting at “a”.

$$\boxed{\begin{aligned}
&\mathbf{O}_d \leftarrow (1-L)\mathbf{O}_d + L\left(a\mathbf{I}_a + b\mathbf{O}_a + \mathbf{I}_d\right)\\
&\mathbf{y} \leftarrow (1-L)\mathbf{y} + L(\mathbf{I}_a + c\mathbf{O}_d - \mathbf{Vm})\\
&\mathbf{m} \leftarrow (1-L)\mathbf{m} + L(c\mathbf{O}_d - \mathbf{Wm})\\
&\mathbf{O}_a \leftarrow (1-L)\mathbf{O}_a + L(d\mathbf{y} + \mathbf{HO}_a + e\mathbf{I}_d - \mathbf{Ts})\\
&\mathbf{s} \leftarrow (1-L)\mathbf{s} + L(\mathbf{HO}_a + f\mathbf{I}_d - \mathbf{Us})
\end{aligned}}$$

C.5. Hierarchical Temporal Models

Original equations.

$$\boxed{\begin{aligned}
&y_t(i) = P(^{-}e_t \mid c_i(t)) \propto \prod_{j=1}^{M} \lambda_t^{m_j}(r_i^{m_j})\\
&\lambda_t^k(g_r) = P(^{-}e_0^t \mid g_r(t)) \propto \sum_{c_i(t)\in C^k} \alpha_t(c_i, g_r)\\
&\alpha_t(c_i, g_r) = P(^{-}e_t \mid c_i(t)) \sum_{c_j(t-1)\in C^k} P(c_i(t) \mid c_j(t-1), g_r)\alpha_{t-1}(c_j, g_r)\\
&\alpha_0(c_i, g_r) = P(^{-}e_0 \mid c_i(t=0))P(c_i(t=0) \mid g_r)\\
&Bel_t(c_i) \propto \sum_{g_r\in G^k} P(g_r|^{+}e_0)\beta_t(c_i, g_r)\\
&\beta_t(c_i, g_r) = P(^{-}e_t \mid c_i(t)) \sum_{c_j(t-1)\in C^k} P(c_i(t) \mid c_j(t-1), g_r)\beta_{t-1}(c_j, g_r)\\
&\beta_0(c_i, g_r) = P(^{-}e_0 \mid c_i(t=0))P(c_i(t=0) \mid g_r, {}^{+}e_0)\\
&\pi^{child}(g_m) \propto \sum_{i\forall i} I(c_i)Bel(c_i)\\
&I(c_i) = \begin{cases} 1, \text{if } g_m^{child} \text{ is a component of } c_i\\ 0, \text{otherwise}\end{cases}
\end{aligned}}$$

Remove initialization of α and β, and assume random initialization instead. The original model appears to assume episodic operation with a full reset between each problem presentation. Because our focus is on neural plausibility, we change the assumption to continuous operation with time-varying input. In this case, initialization becomes insignificant.

$$y_t(i) = P({}^{-}e_t \mid c_i(t)) \propto \prod_{j=1}^{M} \lambda_t^{m_j}(r_i^{m_j})$$
$$\lambda_t^k(g_r) = P({}^{-}e_{_0}^{t} \mid g_r(t)) \propto \sum_{c_i(t)\in C^k} \alpha_t(c_i, g_r)$$
$$\alpha_t(c_i, g_r) = P({}^{-}e_t \mid c_i(t)) \sum_{c_j(t-1)\in C^k} P(c_i(t) \mid c_j(t-1), g_r)\alpha_{t-1}(c_j, g_r)$$
$$Bel_t(c_i) \propto \sum_{g_r \in G^k} P(g_r \mid {}^{+}e_0)\beta_t(c_i, g_r)$$
$$\beta_t(c_i, g_r) = P({}^{-}e_t \mid c_i(t)) \sum_{c_j(t-1)\in C^k} P(c_i(t) \mid c_j(t-1), g_r)\beta_{t-1}(c_j, g_r)$$
$$\pi^{child}(g_m) \propto \sum_{i \forall i} I(c_i) Bel(c_i)$$
$$I(c_i) = \begin{cases} 1, \text{if } g_m^{child} \text{ is a component of } c_i \\ 0, \text{otherwise} \end{cases}$$

Apart from initialization, α and β are exactly the same. Therefore, only compute α and substitute it for β.

$$y_t(i) = P({}^{-}e_t \mid c_i(t)) \propto \prod_{j=1}^{M} \lambda_t^{m_j}(r_i^{m_j})$$
$$\lambda_t^k(g_r) = P({}^{-}e_{_0}^{t} \mid g_r(t)) \propto \sum_{c_i(t)\in C^k} \alpha_t(c_i, g_r)$$
$$\alpha_t(c_i, g_r) = P({}^{-}e_t \mid c_i(t)) \sum_{c_j(t-1)\in C^k} P(c_i(t) \mid c_j(t-1), g_r)\alpha_{t-1}(c_j, g_r)$$
$$Bel_t(c_i) \propto \sum_{g_r \in G^k} P(g_r \mid {}^{+}e_0)\alpha_t(c_i, g_r)$$
$$\pi^{child}(g_m) \propto \sum_{i \forall i} I(c_i) Bel(c_i)$$
$$I(c_i) = \begin{cases} 1, \text{if } g_m^{child} \text{ is a component of } c_i \\ 0, \text{otherwise} \end{cases}$$

Change to update equation form, but keep time expressions where necessary to distinguish probability expressions.

$$
\boxed{\begin{aligned}
&y(i) \leftarrow P({}^{-}e \mid c_i) \propto \prod_{j=1}^{M} \lambda^{m_j}(r_i^{m_j}) \\
&\lambda^k(g_r) \leftarrow P({}^{-}e_0^t \mid g_r) \propto \sum_{c_i \in C^k} \alpha(c_i, g_r) \\
&\alpha(c_i, g_r) \leftarrow P({}^{-}e \mid c_i) \sum_{c_j \in C^k} P(c_i(t) \mid c_j(t-1), g_r)\alpha(c_j, g_r) \\
&Bel(c_i) \propto \sum_{g_r \in G^k} P(g_r \mid {}^{+}e_0)\alpha(c_i, g_r) \\
&\pi^{child}(g_m) \propto \sum_{i \forall i} I(c_i) Bel(c_i) \\
&I(c_i) = \begin{cases} 1, \text{if } g_m^{child} \text{ is a component of } c_i \\ 0, \text{otherwise} \end{cases}
\end{aligned}}
$$

Convert to linear algebra notation: Define **C** to be a set of basis vectors for the input, where each row is a pattern, and each column is associated with an element in the input vector. Notice that each column of **C** is associated with a node in one or more Markov chains. Substitute matrix multiplication with **C** for the equivalent operations. Substitute vectors for indexed scalars in those same operations. Rename the ascending inputs to $\mathbf{I}_a$ to avoid confusion with the ascending outputs. Change the nature of the input vector from binary to real-valued. In the first equation of the set, change the product into a sum to enables us to express it as a matrix operation. Observe how this changes the mathematical meaning. The original computes the probability that a given input simultaneously matches all the features in each given "coincidence" vector (row of **C**). The new equation computes the cosine distance between the input and each row. With appropriate squashing and normalization, the cosine distance could also be treated as a probability, but with somewhat different meaning than the original.

$$
\boxed{\begin{aligned}
&\mathbf{y} \leftarrow \mathbf{C}\mathbf{I}_a \\
&\lambda^k(g_r) \leftarrow P({}^{-}e_0^t \mid g_r) \propto \sum_{c_i \in C^k} \alpha(c_i, g_r) \\
&\alpha(c_i, g_r) \leftarrow P({}^{-}e \mid c_i) \sum_{c_j \in C^k} P(c_i(t) \mid c_j(t-1), g_r)\alpha(c_j, g_r) \\
&Bel(c_i) \propto \sum_{g_r \in G^k} P(g_r \mid {}^{+}e_0)\alpha(c_i, g_r) \\
&\boldsymbol{\pi} = \mathbf{C}^T Bel
\end{aligned}}
$$

The expression $P(c_i(t) \mid c_j(t-1), g_r)$ gives the Markov transition probabilities. If none of the Markov chains share nodes (as described in the first three chapters of [George 2008]), then this can be expressed as a single block-diagonal matrix **M**. The fourth chapter of [George 2008] generalizes the formulation to allow nodes to be shared between Markov chains. Presumably, the transition probabilities within each chain are normalized separately from the others. In this case, **M** is still block diagonal, but any given node may be associated with several row or column indices, rather than just one. Implicitly, there is also mapping between elements of **y** and sets of units that handle each of the Markov chains. This mapping can be combined with matrix of basis vectors **C** or it can be a separate transform. For the purpose of this analysis, we will assume it is rolled into **C**. Substitute a matrix multiplication with **M** for the equivalent operations. Substitute the vector **y** for the equivalent probability expression.

$$\boxed{\begin{array}{l} \mathbf{y} \leftarrow \mathbf{C}\mathbf{I}_a \\ \lambda^k(g_r) \leftarrow P({}^{-}e_0^t \mid g_r) \propto \sum\limits_{c_i \in C^k} \alpha(c_i, g_r) \\ \boldsymbol{\alpha} \leftarrow \mathbf{y} \circ \mathbf{M}\boldsymbol{\alpha} \\ Bel(c_i) \propto \sum\limits_{g_r \in G^k} P(g_r \mid {}^{+}e_0)\alpha(c_i, g_r) \\ \boldsymbol{\pi} \leftarrow \mathbf{C}^T Bel \end{array}}$$

Use standard variable names for ascending and descending outputs.

$$\boxed{\begin{array}{l} \mathbf{y} \leftarrow \mathbf{C}\mathbf{I}_a \\ \mathbf{O}_a(g_r) \leftarrow P({}^{-}e_0^t \mid g_r) \propto \sum\limits_{c_i \in C^k} \alpha(c_i, g_r) \\ \boldsymbol{\alpha} \leftarrow \mathbf{y} \circ \mathbf{M}\boldsymbol{\alpha} \\ Bel(c_i) \propto \sum\limits_{g_r \in G^k} P(g_r \mid {}^{+}e_0)\alpha(c_i, g_r) \\ \mathbf{O}_d \leftarrow \mathbf{C}^T Bel \end{array}}$$

Define a matrix **N** to represent the assignment of Markov nodes to their respective chains, such that the rows are associated with chains, and each column is associated with a node. If a node is assigned to a chain, the element in the matrix **N** is 1, otherwise it is 0. This matrix does two jobs. First, it sums up activity from all the nodes in a chain and passes the value up the next stage as an element in the next feature vector. Second, it maps descending inputs to Markov chains. Substitute matrix multiplication with N for the equivalent operations. Also, use $P(g_r|^{+}e)$ instead of $P(g_r|^{+}e_0)$, that is, allow descending input to vary over time, as opposed to remaining constant for a given episode. Again, this is due to our continuous operation assumption, which is different from George's work.

$$\boxed{\begin{array}{l} \mathbf{y} \leftarrow \mathbf{C}\mathbf{I}_a \\ \mathbf{O}_a \leftarrow \mathbf{N}\boldsymbol{\alpha} \\ \boldsymbol{\alpha} \leftarrow \mathbf{y} \circ \mathbf{M}\boldsymbol{\alpha} \\ Bel \leftarrow \boldsymbol{\alpha} \circ \mathbf{N}^T\mathbf{I}_d \\ \mathbf{O}_d \leftarrow \mathbf{C}^T Bel \end{array}}$$

Eliminate *Bel*.

$$\boxed{\begin{array}{l} \mathbf{y} \leftarrow \mathbf{C}\mathbf{I}_a \\ \mathbf{O}_a \leftarrow \mathbf{N}\boldsymbol{\alpha} \\ \boldsymbol{\alpha} \leftarrow \mathbf{y} \circ \mathbf{M}\boldsymbol{\alpha} \\ \mathbf{O}_d \leftarrow \mathbf{C}^T(\boldsymbol{\alpha} \circ \mathbf{N}^T\mathbf{I}_d) \end{array}}$$

Rename α to x for better readability. Rearrange order of equations.

$$\boxed{\begin{array}{l}\mathbf{y} \leftarrow \mathbf{C}\mathbf{I}_a \\ \mathbf{x} \leftarrow \mathbf{y} \circ \mathbf{M}\mathbf{x} \\ \mathbf{O}_a \leftarrow \mathbf{N}\mathbf{x} \\ \mathbf{O}_d \leftarrow \mathbf{C}^T(\mathbf{x} \circ \mathbf{N}^T\mathbf{I}_d)\end{array}}$$

References

Thomas J. Anastasio (2001). *Input minimization: a model of cerebellar learning without climbing fiber error signals*. Neuroreport volume 12, number 17, pages 3825-31.

Korbinian Brodmann (1909). *Vergleichende Lokalisationslehre der Grosshirnrinde in ihren Prinzipien dargestellt auf Grund des Zellenbaues*. J.A. Barth Verlag, Leipzig.

Edward M. Callaway (1998). *Local Circuits in Primary Visual Cortex of the Macaque Monkey*. Annual Review of Neuroscience volume 21, pages 47–74.

Gail A. Carpenter and Stephen Grossberg (1987). *A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine*. Computer Vision, Graphics, and Image Processing volume 37, pages 54-115.

Nuno Maçarico da Costa and Kevan A. C. Martin (2010). *Whose cortical column would that be?* Frontiers in Nueroanatomy volume 4, article 16.

R.J. Douglas and Kevin C. Martin (1991). *A functional microcircuit for cat visual cortex*. Journal of Physiology volume 440, pages 735–769.

R.J. Douglas and Kevin C. Martin (2004). *Neuronal Circuits of the Neocortex*. Annual Review of Neuroscience volume 27, pages 419–451.

R.J. Douglas and Kevin C. Martin (2007). *Recurrent neuronal circuits in the neocortex*. Current Biology volume 17, number 13, pages 496-500.

Dileep George (2008). *How the Brain Might Work: A Hierarchical and Temporal Model for Learning and Recognition*. Ph.D. Thesis. Stanford University.

Stephen Grossberg (1999). *How Does the Cerebral Cortex Work? Learning, Attention, and Grouping by the Laminar Circuits of Visual Cortex*. Spatial Vision volume 12, pages 163-186.

Jeff Hawkins and Sandra Blakeslee (2004). *On Intelligence*. Times Books.

Jonathan C. Horton and Daniel L. Adams (2005). *The cortical column: a structure without a function*. Philosophical Transactions of the Royal Society B volume 360, pages 837-862.

David H. Hubel and Torsten N. Wiesel (1959). *Receptive fields of single neurons in the cat's striate cortex*. Journal of Physiology volume 148, pages 574-591.

David H. Hubel and Torsten N. Wiesel (1968). *Receptive Fields and Functional Architecture of Monkey Striate Cortex*. Journal of Physiology volume 195, pages 215-243.

David H. Hubel and Torsten N. Wiesel (1974). *Sequence Regularity and Geometry of Orientation Columns in the Monkey Striate Cortex.* Journal of Comparative Neurology volume 158, pages 267-294.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato and Yann LeCun (2009). *What is the Best Multi-Stage Architecture for Object Recognition?* IEEE International Conference on Computer Vision.

Teuvo Kohonen (1982). *Self-Organized Formation of Topologically Correct Feature Maps.* Bilogical Cybernetics volume 43, pages 59-69.

Yann LeCun, Patrick Haffner, Léon Bottou and Yoshua Bengio (1999). *Object Recognition with Gradient-Based Learning*. In David Forsyth (Editor), *Feature Grouping*, Springer.

Vernon B. Mountcastle (1957). *Modality and topographic properties of single neurons of cat's somatic sensory cortex.* Journal of Neurophysiology volume 20, pages 408-434.

R. Nelson, E.V. Famiglietti, and Helga Kolb (1978). *Intracellular staining reveals different levels of stratification for on-center and off-center ganglion cells in the cat retina*. Journal of Neurophysiology volume 41, pages 472-483.

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

Rajeev D.S. Raizada and Stephen Grossberg (2001). *Context-Sensitive Binding by the Laminar Circuits of V1 and V2: A Unified Model of Perceptual Grouping, Attention, and Orientation Contrast*. Visual Cognition volume 8, pages 431-466.

Rajesh P.N. Rao (1999). *An Optimal Estimation Approach to Visual Perception and Learning*. Vision Research, volume 39, number 11, pages 1963-1989.

Rajesh P.N. Rao and Dana H. Ballard (1997). *Dynamic model of visual recognition predicts neural response properties in the visual cortex.* Neural Computation volume 9, pages 721-763.

Rajesh P. N. Rao and Dana H. Ballard (1999). *Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects.* Nature Neuroscience volume 2, number 1, pages 79-87.

Brandon Rohrer (2010). *kx-trees: An unsupervised learning method for use in developmental agents*. 9th International Conference on Development and Learning, Ann Arbor, Michigan, Aug 18-21.

Brandon Rohrer, Michael Bernard, James D. Morrow, Fred Rothganger and Patrick Xavier (2009). *Model-free learning and control in a mobile robot.* 5th International Conference on Natural Computation, Tianjin, China, Aug 14-16.

Fredrick H. Rothganger and Thomas J. Anastasio (2009). *Using input minimization to train a cerebellar model to simulate regulation of smooth pursuit.* Biological Cybernetics volume 101, pages 339-359.

Gordon Shepherd (Editor) (2004). *The Synaptic Organization of the Brain.* 5th edition. Oxford University Press.

Akiya Watakabe (2009). *Comparative molecular neuroanatomy of mammalian neocortex: What can gene expression tell us about areas and layers?* Development, Growth & Differentiation volume 51, pages 343-354.

Kechen Zhang and Terrence J. Sejnowski (2000). *A universal scaling law between gray matter and white matter of cerebral cortex.* PNAS volume 97, pages 5621–5626.

Distribution

1	MS 1188	John Wagner	1432 (electronic copy)
1	MS 1316	Danny Rintoul	1412 (electronic copy)
1	MS 1010	Brandon Rohrer	6473 (electronic copy)
1	MS 1188	Stephen Verzi	1434 (electronic copy)
1	MS 1004	Patrick Xavier	6385 (electronic copy)
1	MS 0899	Technical Library	9536 (electronic copy)
1	MS 0359	D. Chavez, LDRD Office	1911 (electronic copy)

Sandia National Laboratories