Recurrent processing during object recognition
Supporting Information

Randall C. O’Reilly*1,2, Dean Wyatte*,1, Seth Herd1, Brian Mingus1, David J. Jilk2
*These authors contributed equally
1 Department of Psychology and Neuroscience, University of Colorado Boulder
2 eCortex, Inc.
E-mail: randy.oreilly@colorado.edu, dean.wyatte@colorado.edu

1 Structure of the LVis model

1.1 Early Visual Image Processing

In the mammalian brain, the retina and LGN compress the visual input into an efficient contrast-coded representation using center-surround contrast filters that are radially symmetric (which can be nicely approximated by the difference of two Gaussians, Enroth-Cugell & Robson, 1966; Young, 1987). Then area V1 encodes orientation and other features building upon this basic contrast-enhanced input. (Hubel & Wiesel, 1962) We compress this chain of filters into a single step by using oriented Gabor filters, which are defined as a Gaussian-shaped spatial weighting multiplying a planar sine wave oriented in a given direction:

\[ g(x, y) = e^{-\left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}\right)} \sin\left(\frac{2\pi y}{\lambda}\right) \]  

(1)

where the sine wave moves along the y axis (corresponding to a horizontal orientation tuning), and the Gaussian has differential width terms (\(\sigma_x\), \(\sigma_y\)) for each axis. To obtain different orientations, the coordinates x,y are rotated by a given angle \(\theta\) relative to the original coordinates of the filter (\(x', y'\)):

\[
\begin{align*}
x &= x' \cos(\theta) - y' \sin(\theta) \\
y &= y' \cos(\theta) + x' \sin(\theta)
\end{align*}
\]

(2)

The filter is always normalized to a zero sum in the discrete kernel that is actually used, to ensure that a uniform illumination produces no activation.

In line with other established models of object recognition in cortex (e.g., Wallis & Rolls, 1997; Riesenhuber & Poggio, 1999; Dailey & Cottrell, 1999; Masquelier & Thorpe, 2007), these filtering operations provide a reasonable approximation to the coding properties of V1 simple cells. On top of this simple coding, we implement V1 complex cell responses (described in detail below), computed as aggregations over the V1 simple outputs, such that V1 complex cells have
larger spatial receptive fields than V1 simple ones. Computationally, this contributes somewhat to overall spatial invariance in the model (Fukushima, 1980; Riesenhuber & Poggio, 1999), in addition to reducing the dimensionality of the input considerably, to speed processing. In addition to this simple spatial aggregation over the V1 simple outputs (V1 Simple Max), we further specify V1 complex cells as end-stopping and length summing sub-types, both which have been identified in mammalian V1 (Yazdanbakhsh & Livingstone, 2006). kWTA inhibitory competition dynamics take place at both the V1 simple and V1 complex levels. The complex cell activity then provides the input to the model proper.

The LVis model processes each image at two different spatial frequencies (SF), “high” and “medium”, each of which employs 4 orientations of tuning, times 2 for on vs. off-center polarity. The high-SF pathway uses Gabor filters rendered on a 6x6 pixel kernel, with a wavelength $\lambda = 6$, and Gaussian width terms of $\sigma_x = 1.8$ and $\sigma_y = 1.2$. The medium-SF pathway uses Gabor filters that are twice as large (12x12 kernel, wavelength $\lambda = 12$, and Gaussian width terms of $\sigma_x = 3.6$ and $\sigma_y = 2.4$). These filters are applied in a half-overlapping fashion to the image, such that adjacent V1 simple units process spatial locations that are one half wavelength ($\frac{\lambda}{2}$) away from their neighbors.

The input “retina” resolution is 144x144 pixels, and the high frequency V1 simple filters are computed centered on each pixel (spacing = 1), producing a 144x144x8 (where 8 refers to 4 orientations x 2 polarities) dimensional output, while the medium frequency have a spacing of 2 and produce a 72x72x8 dimensional output.

The output of the V1 simple cells is computed using the kWTA dynamics and point-neuron activation function of Leabra (described below), applied to a net input that is the positive-rectified (values less than 0 are clipped to 0) result of convolving the Gabor kernel with the input image. There are two levels of inhibitory competition – the primary is within the group of different orientation and polarity tunings for the same spatial location (i.e., 8 units = 4 orientations x 2 polarities, as shown in Figure 1). This unit group level competition may reflect competition at the level of the hypercolumn in the brain. This inhibition is computed with a k value of one active unit (the flexibility of the average-based kWTA function allows multiple units to become active in the case of close competition). The secondary level of competition involves a spread of the unit-group level competition across the entire layer of such units, with a discounted gain multiplier and a MAX operation such that the stronger of the unit-group or discounted layer-level competition holds (see kWTA section of Leabra algorithm section for details).

The V1 complex cells perform a spatial integration over a 24 x 24 receptive field of V1 simple cells (with adjacent overlap or spacing of 12 in each dimension). Thus, for the high SF channel, the 144x144 spatial resolution of V1 simple is reduced to just 12x12 spatial resolution for the V1 complex output, and for the medium the 72x72 V1 simple input reduces to 6x6 V1 complex output. This reduction is important for making the resulting network on top of the V1 layer small enough to be computationally tractable. There is a Gaussian weighting as a function of distance away from the center of the complex receptive field ($\sigma$ width of Gaussian = 19.2 – a fairly broad tuning), with the MAX resulting weighted activation across the receptive field driving the output of the complex cell — this MAX operation is known to promote spatially invariant encoding (Riesenhuber & Poggio, 1999). Each subtype of complex cell is computed as follows:

- **V1 Simple Max**: this is the simplest form of complex cell, which simply performs the above-described spatial weighting and MAX operation on the V1 simple outputs. It contributes 8
Figure 1: The LVis model’s V1 filters represent two stages of early visual processing. The first stage, extending from the retina, through the LGN, to V1 layer IV captures the information encoded by V1 simple cells, which are modeled as oriented Gabor filters with ON-OFF and OFF-ON polarities. Two renditions of these filters are shown – the actual 6x6 filter kernels used in the high-resolution pathway (12x12 filters are used in the medium-resolution pathway), and a schematized version thereof which are then used in the V1 Complex portion of the figure. The second stage of captures many of the computations performed in V1 supra- and infragranular layers (referred to here as “V1 complex”). Qualitatively, these computations amount to the encoding of additional visual features (end-stoppings and length-summations) as well as a MAX operation over the V1 simple cells’ spatial receptive field (note: the 4x4 grid depicted in the graphic is a visual simplification; the actual computation is performed over the 24x24 spatial receptive field). The Leabra kWTA dynamic is active during both stages of early visual processing such that the net result of processing is similar to that of a hypercolumn of V1 cells. The End-Stop case is the most complicated, showing that a central (colored) orientation detector is contrasted with the orthogonal detectors surrounding it as shown (in black and white).

- **Length Sum:** these complex units integrate across 3 adjacent simple cells along the direction of orientation, so as to respond to longer edges in the image. They also integrate over differences in polarity, such that it only contributes 4 cells per spatial location (one for each

...
orientation). This type of cell is well-described in electrophysiological recordings (Yazdanbakhsh & Livingstone, 2006).

- **End Stop**: these are the most complex of the complex units, responding to a contrasting orientation located in any of 4 different positions relative to the central location (see Figure 1). They are activated maximally when an orientation changes rapidly, such as at a T junction or a corner. They also integrate over differences in polarity, contributing 4 cells per location. This type of cell is also well-described in electrophysiological recordings (Yazdanbakhsh & Livingstone, 2006).

Figure 1 shows a schematic of the full set of V1 filtering operations, including the actual set of Gabor filters for the high-resolution pathway. The net result from this computation is that the 8 units per spatial location in the V1 simple layer have been expanded to 16 units in the complex layer, but the number of spatial locations encoded has been cut by a factor of 12, resulting in a significant net overall improvement in computational speed. Each of the above cell types is included not only for biological fidelity of the model, but also because it contributes to successful recognition performance.

A kWTA dynamic again operates within the unit groups in the complex layer with aggregation to the entire layer as described above.

### 1.2 Structure of Higher Layers (Extrastriate, Inferotemporal, Output, Semantics)

Proceeding from the V1 complex inputs at the two different spatial frequencies (high and medium), the LVIs model captures the general response properties of extrastriate cortex (V2/V4) and inferotemporal (IT) cortex. As a purely computational convenience in configuring the network, the model’s extrastriate layers remain spatial-frequency specific (in the brain, we would expect these to all be intermixed), which then merge into a unitary IT layer, which then feeds into a naming output layer, and a semantics output layer (Figure 2). All connections are bidirectional, except those from the extrastriate layers back to V1 (in this simplified model, V1 serves strictly as an input layer, but a more complex implementation that contained the full series of backprojections performed similarly at the cost of computational efficiency). The feedforward projections had a relative overall weight strength (relative contribution to postsynaptic net excitatory drive) of 80%, while the feedback projections contributed the remaining 20%. There is some consistent evidence for this differential strength of feedback connections (Self, Kooijmans, Super, Lamme, & Roelfsema, 2012).

All of the units are allowed to learn based on the Leabra learning mechanism (described in the next section), and investigations of the learned weights indicate that sensible intermediate representations develop in the extrastriate and IT layers. For example, the extrastriate neurons become tuned to more complex features over a larger spatial receptive field (compared to V1) during learning. Similarly, IT neurons learn a fully invariant representation of yet more complex features.

Once trained, the single model can discriminate all trained object categories — in contrast, other prevalent feedforward models (e.g., Riesenhuber & Poggio, 1999; Masquelier & Thorpe, 2007) use binary classifiers that would require \( N \) classifiers to differentiate among \( N \) categories. Thus, the overall solution to the invariant object recognition problem that the model develops is
Figure 2: The full LVis model consists of two spatial frequency pathways through V1 and extrastriate cortex that converge at IT. The model’s IT layer feeds into two higher-level layers, a name output layer (Output in the figure) and a Semantics output layer. All layers are connected bidirectionally (except those from the extrastriate layers back to V1), which allows the Leabra learning mechanism to shape representations at all levels over the course of learning.

qualitatively similar with the prevalent feedforward models, yet is also realizable using a homogeneous, biologically plausible set of mechanisms.

Here are the detailed parameters for each layer in the network (note that 15-25% activity levels is the default for Leabra models of the cortex, based on biological estimates; O’Reilly & Munakata, 2000; O’Reilly, Munakata, Frank, Hazy, & Contributors, 2012):

- **Extrastriate**: 64 units per unit group/hypercolumn, arranged into an 6x6 topographical grid (2,304 total units) for the high spatial frequency layer, and a 3x3 grid (576 total units) for the medium spatial frequency layer. Each unit receives from a topographically-corresponding 4x4 grid of V1 unit groups, with 1/2 overlap among neighboring unit groups. kWTA set to 10% activity within each unit group, and shared with a 50% gain factor across the layer.

- **IT**: 200 total units receiving a full projection from all of the extrastriate units (and projecting bidirectionally back to them), with a 15% kWTA activity level (no unit group sub-structure).

- **Naming Output**: 200 units receiving a full projection from the IT (and projecting completely back to it), with a kWTA activity level of 1%. This localist (single active unit)
representation of output names is a computational simplification, standing in for the full phonological production pathways.

- **Semantics**: 200 units receiving fully from IT and projecting completely back to it, with kWTA activation at 25%. Semantics patterns were generated from the LSA pairwise semantics of the object categories.

## 2 The Leabra Learning Algorithm

![Illustration of the primary Leabra mechanisms](image-url)

**Figure 3**: Illustration of the primary Leabra mechanisms, which are described in detail in the text.

The Leabra framework is described in detail in O’Reilly and Munakata (2000; O’Reilly et al., 2012) and O’Reilly (2001), and summarized here. See Table 1 for a listing of parameter values, nearly all of which are at their default settings. This model uses a newer variant of the Leabra equations, but similar performance was obtained from the original standard Leabra equations, which have been used to simulate over 40 different models in O’Reilly and Munakata (2000), and a number of other research models. Thus, the model can be viewed as an instantiation of a systematic modeling framework using standardized mechanisms, instead of constructing new mechanisms for each model. The model can be obtained by emailing the first author at randy.oreilly@colorado.edu.

This version of Leabra contains two primary differences from the original: the activation function is slightly different, in a way that allows units to more accurately reflect their graded excitatory input drive, and the learning rule takes a more continuous form involving contrasts between values integrated over different time frames (i.e., with different time constants), which also produces...
a combination of error-driven and self-organizing learning within the same simple mathematical framework. The fuller explication of this learning rule is currently in preparation (O’Reilly, Herd, & Hazy, in preparation), and goes by the acronym of XCAL (temporally eXtended Contrastive Attractor Learning). It replaces the combination of Contrastive Hebbian Learning (CHL) and standard Hebbian learning used in the original Leabra framework.

2.1 Pseudocode

The pseudocode for Leabra is given here, showing exactly how the pieces of the algorithm described in more detail in the subsequent sections fit together. The individual steps are repeated for each event (trial), which can be broken down into a “minus” and “plus” phase, followed by a synaptic weight updating function. Generally speaking, the minus phase represents the system’s expectation for a given input and the plus phase represents the observation of the outcome. The difference between these two phases is then used to compute the updating function that drives learning. Furthermore, each phase contains a period of “settling” (measured in “cycles”) during which the activation values of each unit are updated taking into account the previous state of the network. Some units are “clamped”, or have fixed activation values and are not subject to this updating rule (e.g., V1 input in the minus phase, V1 input and Output in the plus phase).

Outer loop: For each event (trial) in an epoch:

1. Iterate over minus and plus phases of settling for each event.

   (a) At start of settling, for all units:
   i. Initialize all state variables (activation, $V_m$, etc).
   ii. Clamp external patterns (V1 input in minus phase, V1 input & Output in plus phase).

   (b) During each cycle of settling, for all non-clamped units:
   i. Compute excitatory netinput ($g_e(t)$ or $\eta_j$, eq 5).
   ii. Compute kWTA inhibition for each layer, based on $g_i^\Theta$ (eq 11):
      A. Sort units into two groups based on $g_i^\Theta$: top $k$ and remaining $k + 1$ to $n$.
      B. If basic, find $k$ and $k + 1$th highest; if avg-based, compute avg of $1 \rightarrow k$ & $k + 1 \rightarrow n$.
      C. Set inhibitory conductance $g_i$ from $g_k^\Theta$ and $g_{k+1}^\Theta$ (eq 10).
   iii. Compute point-neuron activation combining excitatory input and inhibition (eq 3).
   iv. Update time-averaged activation values (short, medium, long) for use in learning.

2. After both phases update the weights, for all connections:

   (a) Compute XCAL learning as function of short, medium, and long time averages.
   (b) Increment the weights according to net weight change.
Table 1: Parameters for the simulation (see equations in text for explanations of parameters). All are standard default parameter values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_l$</td>
<td>0.30</td>
</tr>
<tr>
<td>$E_i$</td>
<td>0.25</td>
</tr>
<tr>
<td>$V_{rest}$</td>
<td>0.30</td>
</tr>
<tr>
<td>$\tau$</td>
<td>.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g_l$</td>
<td>0.10</td>
</tr>
<tr>
<td>$g_i$</td>
<td>1.00</td>
</tr>
<tr>
<td>$\Theta$</td>
<td>0.50</td>
</tr>
</tbody>
</table>

2.2 Point Neuron Activation Function

Leabra uses a point neuron activation function that models the electrophysiological properties of real neurons, while simplifying their geometry to a single point. This function is nearly as simple computationally as the standard sigmoidal activation function, but the more biologically-based implementation makes it considerably easier to model inhibitory competition, as described below. Further, using this function enables cognitive models to be more easily related to more physiologically detailed simulations, thereby facilitating bridge-building between biology and cognition.

We use normalized units where the unit of time is 1 msec, the unit of electrical potential is 0.1 V (with an offset of -0.1 for membrane potentials and related terms, such that their normal range stays within the $[0, 1]$ normalized bounds), and the unit of current is $1.0 \times 10^{-8}$.

The membrane potential $V_m$ is updated as a function of ionic conductances $g$ with reversal (driving) potentials $E$ as follows:

$$\Delta V_m(t) = \tau \sum_c g_c(t)g_c(E_c - V_m(t))$$

(3)

with 3 channels ($c$) corresponding to: $e$ excitatory input; $l$ leak current; and $i$ inhibitory input.

Following electrophysiological convention, the overall conductance is decomposed into a time-varying component $g_c(t)$ computed as a function of the dynamic state of the network, and a constant $g_c$ that controls the relative influence of the different conductances. The equilibrium potential can be written in a simplified form by setting the excitatory driving potential ($E_e$) to 1 and the leak and inhibitory driving potentials ($E_l$ and $E_i$) of 0:

$$V^\infty_m = \frac{g_e g_c}{g_e + g_i g_c + g_i g_i}$$

(4)

which shows that the neuron is computing a balance between excitation and the opposing forces of leak and inhibition. This equilibrium form of the equation can be understood in terms of a Bayesian decision making framework (O’Reilly & Munakata, 2000).

The excitatory net input/conductance $g_e(t)$ or $\eta_j$ is computed as the proportion of open excitatory channels as a function of sending activations times the weight values:

$$\eta_j = g_e(t) = \langle x_i w_{ij} \rangle = \frac{1}{n} \sum_i x_i w_{ij}$$

(5)

The inhibitory conductance is computed via the kWTA function described in the next section, and leak is a constant.
In its discrete spiking mode, Leabra implements exactly the AdEx (adaptive exponential) model (Brette & Gerstner, 2005), which has been found through various competitions to provide an excellent fit to the actual firing properties of cortical pyramidal neurons (Wulfram & Richard, 2009), while remaining simple and efficient to implement. However, we typically use a rate-code approximation to discrete firing, which produces smoother more deterministic activation dynamics, while capturing the overall firing rate behavior of the discrete spiking model.

We recently discovered that our previous strategy of computing a rate-code graded activation value directly from the membrane potential is problematic, because the mapping between $V_m$ and mean firing rate is not a one-to-one function in the AdEx model. Instead, we have found that a very accurate approximation to the discrete spiking rate can be obtained by comparing the excitatory net input directly with the effective computed amount of net input required to get the neuron firing over threshold ($g_e^\Theta$), where the threshold is indicated by $\Theta$:

$$g_e^\Theta = \frac{g_l g_i (E_i - V_m^\Theta) + g_l (E_l - V_m^\Theta)}{g_e (V_m^\Theta - E_e)}$$ (6)

$$y_j(t) \propto g_e(t) - g_e^\Theta$$ (7)

where $y_j(t)$ is the firing rate output of the unit.

We continue to use the Noisy X-over-X-plus-1 (NXX1) function, which starts out with a nearly linear function, followed by a saturating nonlinearity:

$$y_j(t) = \frac{1}{\left(1 + \frac{1}{\gamma [g_e(t) - g_e^\Theta]}\right)}$$ (8)

where $\gamma$ is a gain parameter, and $[x]_+$ is a threshold function that returns 0 if $x < 0$ and $x$ if $x > 0$. Note that if it returns 0, we assume $y_j(t) = 0$, to avoid dividing by 0. As it is, this function has a very sharp threshold, which interferes with graded learning learning mechanisms (e.g., gradient descent). To produce a less discontinuous deterministic function with a softer threshold, the function is convolved with a Gaussian noise kernel ($\mu = 0$, $\sigma = .005$), which reflects the intrinsic processing noise of biological neurons:

$$y^*_j(x) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-z^2/(2\sigma^2)} y_j(z - x) dz$$ (9)

where $x$ represents the $[g_e(t) - g_e^\Theta]_+$ value, and $y^*_j(x)$ is the noise-convolved activation for that value. In the simulation, this function is implemented using a numerical lookup table.

### 2.3 k-Winners-Take-All Inhibition

Leabra uses a kWTA (k-Winners-Take-All) function to achieve inhibitory competition among units within a layer (area). The kWTA function computes a uniform level of inhibitory current for all units in the layer, such that the $k + 1$th most excited unit within a layer is generally below its firing threshold, while the $k$th is typically above threshold. Activation dynamics similar to those produced by the kWTA function have been shown to result from simulated inhibitory interneurons that project both feedforward and feedback inhibition (O’Reilly & Munakata, 2000). Thus,
although the kWTA function is somewhat biologically implausible in its implementation (e.g., requiring global information about activation states and using sorting mechanisms), it provides a computationally effective approximation to biologically plausible inhibitory dynamics.

kWTA is computed via a uniform level of inhibitory current for all units in the layer as follows:

\[ g_i = g_{k+1}^\Theta + q(g_k^\Theta - g_{k+1}^\Theta) \]  

(10)

where \( 0 < q < 1 \) (.25 default used here) is a parameter for setting the inhibition between the upper bound of \( g_k^\Theta \) and the lower bound of \( g_{k+1}^\Theta \). These boundary inhibition values are computed as a function of the level of inhibition necessary to keep a unit right at threshold \( \Theta \):

\[ g_i^\Theta = g_e^* E_e - \Theta + g_l^* (E_l - \Theta) \]

(11)

where \( g_e^* \) is the excitatory net input without the bias weight contribution — this allows the bias weights to override the kWTA constraint.

In the basic version of the kWTA function, which is relatively rigid about the kWTA constraint and is therefore used for output layers, \( g_k^\Theta \) and \( g_{k+1}^\Theta \) are set to the threshold inhibition value for the \( k \)th and \( k+1 \)th most excited units, respectively. Thus, the inhibition is placed exactly to allow \( k \) units to be above threshold, and the remainder below threshold. For this version, the \( q \) parameter is almost always .25, allowing the \( k \)th unit to be sufficiently above the inhibitory threshold.

In the average-based kWTA version, \( g_k^\Theta \) is the average \( g_i^\Theta \) value for the top \( k \) most excited units, and \( g_{k+1}^\Theta \) is the average of \( g_i^\Theta \) for the remaining \( n-k \) units. This version allows for more flexibility in the actual number of units active depending on the nature of the activation distribution in the layer and the value of the \( q \) parameter (which is typically .6), and is therefore used for hidden layers.

2.4 XCAL Learning

The full treatment of the new XCAL version of learning in Leabra is presented in an in-preparation paper (O’Reilly et al., in preparation), but the basic equations and a brief motivation for them are presented here.

In the original Leabra algorithm, learning was the sum of two terms: an error-driven component and a Hebbian self-organizing component. In the new XCAL formulation, the error-driven and self-organizing factors emerge out of a single learning rule, which was derived from a biologically detailed model of synaptic plasticity by Urakubo et al. (Urakubo, Honda, Froemke, & Kuroda, 2008), and is closely related to the Bienenstock, Cooper & Munro (BCM) algorithm (Bienenstock, Cooper, & Munro, 1982). In BCM, a Hebbian-like sender-receiver activation product term is modulated by the extent to which the receiving unit is above or below a long-term running average activation value:

\[ \Delta_{\text{bcm}} w_{ij} = x y (y - \langle y^2 \rangle) \]

(12)

(\( x \) = sender activation, \( y \) = receiver activation, and \( \langle y^2 \rangle \) = long-term average of squared receiver activation). The long-term average value acts like a dynamic plasticity threshold, and causes less-active units to increase their weights, while more-active units tend to decrease theirs (i.e., a classic homeostatic function). This form of learning resembles Hebbian learning in several respects, but
can learn higher-order statistics, whereas Hebbian learning is more constrained to extract low-order correlational statistics. Furthermore, the BCM model may provide a better account of various experimental data, such as monocular deprivation experiments (Cooper, Intrator, Blais, & Shouval, 2004).

Figure 4: XCAL dWt function, shown with $\theta_p = 0.5$, which determines the cross-over point between negative and positive weight changes, and $\theta_p \theta_d$ determines the inflection point at the left where the curve goes from a negative slope to a positive slope. This function fits the results of the highly detailed Urakubo et al (Urakubo et al., 2008) model, with a correlation value of $r = 0.89$.

The Leabra XCAL learning rule is based on a contrast between a sender-receiver activation product term (shown initially as just $xy$ – relevant time scales of averaging for this term are elaborated below) and a dynamic plasticity threshold $\theta_p$ (also elaborated below), which are integrated in the XCAL learning function (Figure 4):

$$\Delta_{xcal} w_{ij} = f_{xcal}(xy, \theta_p)$$

where the XCAL learning function was derived by fitting a piecewise-linear function to the Urakubo et al (Urakubo et al., 2008) simulation results based on synaptic drive levels (sender and receiver firing rates; the resulting fit was very good, with a correlation of $r = 0.89$):

$$f_{xcal}(xy, \theta_p) = \begin{cases} (xy - \theta_p) & \text{if } xy > \theta_p \theta_d \\ -xy(1 - \theta_d)/\theta_d & \text{otherwise} \end{cases}$$

($\theta_d = .1$ is a constant that determines the point where the function reverses back toward zero within the weight decrease regime – this reversal point occurs at $\theta_p \theta_d$, so that it adapts according to the dynamic $\theta_p$ value).

The BCM equation produces a curved quadratic function that has the same qualitative shape as the XCAL function (Figure 4). A critical feature of these functions is that they go to 0 as the synaptic activity goes to 0, which is in accord with available data, and that they exhibit a crossover point from LTD to LTP as a function of synaptic drive (which is represented biologically by intracellular Calcium levels). A nice advantage of the linear XCAL function is that, to first approximation, it is just computing the subtraction $xy - \theta_p$.

To achieve full error-driven learning within this XCAL framework, we just need to ensure that the core subtraction represents an error-driven learning term. In the original Leabra, error-driven
learning via the Contrastive Hebbian Learning algorithm (CHL) was computed as:

$$\Delta_{chl} = x^+ y^+ - x^- y^- \quad (15)$$

where the superscripts represent the plus (+) and minus (−) phases. This equation was shown to compute the same error gradient as the backpropagation algorithm, subject to symmetry and a 2nd-order numerical integration technique known as the midpoint method, based the generalized recirculation algorithm (GeneRec; (O’Reilly, 1996)). In XCAL, we replace these values with time-averaged activations computed over different time scales:

- **s** = short time scale, reflecting the most recent state of neural activity (e.g., past 100-200 msec). This is considered the “plus phase” – it represents the outcome information on the current trial, and in general should be more correct than the medium time scale.

- **m** = medium time scale, which integrates over an entire psychological “trial” of roughly a second or so – this value contains a mixture of the “minus phase” and the “plus phase”, but in contrasting it with the short value, it plays the role of the minus phase value, or expectation about what the system thought should have happened on the current trial.

- **l** = long time scale, which integrates over hours to days of processing – this is the BCM-like threshold term.

Thus, the error-driven aspect of XCAL learning is driven essentially by the following term:

$$\Delta_{xcal-err}w_{ij} = f_{xcal}(x_s y_s, x_m y_m) \quad (16)$$

However, consider the case where either of the short term values ($x_s$ or $y_s$) is 0, while both of the medium-term values are > 0 – from an error-driven learning perspective, this should result in a significant weight decrease, but because the XCAL function goes back to 0 when the input drive term is 0, the result is no weight change at all. To remedy this situation, we assume that the short-term value actually retains a small trace of the medium-term value:

$$\Delta_{xcal-err}w_{ij} = f_{xcal}(\kappa x_s y_s + (1 - \kappa) x_m y_m, x_m y_m) \quad (17)$$

(where $\kappa = .9$, such that only .1 of the medium-term averages are incorporated into the effective short-term average).

The self-organizing aspect of XCAL is driven by comparing this same synaptic drive term to a longer-term average, as in the BCM algorithm:

$$\Delta_{xcal-so}w_{ij} = f_{xcal}(\kappa x_s y_s + (1 - \kappa) x_m y_m, \gamma t y_t) \quad (18)$$

where $\gamma_t = 3$ is a constant that scales the long-term average threshold term (due to sparse activation levels, these long-term averages tend to be rather low, so the larger gain multiplier is necessary to make this term relevant whenever the units actually are active and adapting their weights).

Combining both of these forms of learning in the full XCAL learning rule amounts to computing an aggregate $\theta_p$ threshold that reflects a combination of both the self-organizing long-term average, and the medium-term minus-phase like average:

$$\Delta_{xcal}w_{ij} = f_{xcal}(\kappa x_s y_s + (1 - \kappa) x_m y_m, \lambda \gamma t y_t + (1 - \lambda) x_m y_m) \quad (19)$$
where $\lambda = .01$ is a weighting factor determining the mixture of self-organizing and error-driven learning influences (as was the case with standard Leabra, the balance of error-driven and self-organizing is heavily weighted toward error driven, because error-gradients are often quite weak in comparison with local statistical information that the self-organizing system encodes).

The weight changes are subject to a soft-weight bounding to keep within the $0 - 1$ range:

$$\Delta_{sth}w_{ij} = [\Delta_{scalar}]_+(1 - w_{ij}) + [\Delta_{scalar}]_-w_{ij}$$ (20)

where the $[\cdot]_+$ and $[\cdot]_-$ operators extract positive values or negative-values (respectively), otherwise 0.

Finally, as in the original Leabra model, the weights are subject to contrast enhancement, which magnifies the stronger weights and shrinks the smaller ones in a parametric, continuous fashion. This contrast enhancement is achieved by passing the linear weight values computed by the learning rule through a sigmoidal nonlinearity of the following form:

$$\hat{w}_{ij} = \frac{1}{1 + \left(\frac{w_{ij}}{\theta(1-w_{ij})}\right)^-\gamma}$$ (21)

where $\hat{w}_{ij}$ is the contrast-enhanced weight value, and the sigmoidal function is parameterized by an offset $\theta$ and a gain $\gamma$ (standard defaults of 1 and 6, respectively, used here).

## 3 CU3D-100 dataset

The CU3D-100 dataset consists of 3D models from 100 diverse visual categories downloaded from the Google 3D Warehouse (http://sketchup.google.com/3dwarehouse). The majority of models in the dataset are artifacts, with a small number of biological categories. This specific category distribution is largely due to the purpose behind the Google 3D Warehouse community, which was originally established to create 3D models of buildings for use in Google’s Google Earth software. Thus, a disproportionate number of artifacts that occur in and around buildings (e.g., household objects, vehicles, etc.) make up the total set of available models. We established criteria of a minimum of 6 exemplars for a category to be included in the dataset and generally aimed for categories that contained at least 9-10 exemplars. Biological categories often only contained a few exemplars, likely because they are more difficult to translate into 3D models due to being composed of organic shapes and contoured elements. A full breakdown of categories and number of models is listed in the table below.
<table>
<thead>
<tr>
<th>Category</th>
<th># of models</th>
<th>Category</th>
<th># of models</th>
<th>Category</th>
<th># of models</th>
</tr>
</thead>
<tbody>
<tr>
<td>airplane</td>
<td>15</td>
<td>dresser</td>
<td>7</td>
<td>plate</td>
<td>6</td>
</tr>
<tr>
<td>anchor</td>
<td>7</td>
<td>drums</td>
<td>8</td>
<td>pliers</td>
<td>9</td>
</tr>
<tr>
<td>army tank</td>
<td>9</td>
<td>elephant</td>
<td>7</td>
<td>propellor</td>
<td>8</td>
</tr>
<tr>
<td>banana</td>
<td>8</td>
<td>fan</td>
<td>7</td>
<td>record player</td>
<td>10</td>
</tr>
<tr>
<td>bathroom sink</td>
<td>11</td>
<td>fire hydrant</td>
<td>9</td>
<td>remote control</td>
<td>8</td>
</tr>
<tr>
<td>bed</td>
<td>10</td>
<td>fireplace</td>
<td>10</td>
<td>rifle</td>
<td>10</td>
</tr>
<tr>
<td>bicycle</td>
<td>9</td>
<td>fish</td>
<td>9</td>
<td>road sign</td>
<td>10</td>
</tr>
<tr>
<td>blender</td>
<td>7</td>
<td>flashlight</td>
<td>8</td>
<td>rolloff desk</td>
<td>9</td>
</tr>
<tr>
<td>blimp</td>
<td>10</td>
<td>frying pan</td>
<td>8</td>
<td>sailboat</td>
<td>8</td>
</tr>
<tr>
<td>boombox</td>
<td>9</td>
<td>globe</td>
<td>8</td>
<td>scissors</td>
<td>11</td>
</tr>
<tr>
<td>bottle</td>
<td>11</td>
<td>grenade</td>
<td>10</td>
<td>screwdriver</td>
<td>8</td>
</tr>
<tr>
<td>bow and arrow</td>
<td>9</td>
<td>guitar</td>
<td>8</td>
<td>skateboard</td>
<td>10</td>
</tr>
<tr>
<td>cake</td>
<td>10</td>
<td>hammer</td>
<td>7</td>
<td>skull</td>
<td>7</td>
</tr>
<tr>
<td>camera</td>
<td>11</td>
<td>handgun</td>
<td>8</td>
<td>stapler</td>
<td>8</td>
</tr>
<tr>
<td>candle</td>
<td>12</td>
<td>hat</td>
<td>9</td>
<td>stereo speaker</td>
<td>10</td>
</tr>
<tr>
<td>cannon</td>
<td>11</td>
<td>headphones</td>
<td>10</td>
<td>submarine</td>
<td>10</td>
</tr>
<tr>
<td>cannon</td>
<td>8</td>
<td>helicopter</td>
<td>16</td>
<td>synthesizer</td>
<td>9</td>
</tr>
<tr>
<td>car</td>
<td>16</td>
<td>hot air balloon</td>
<td>12</td>
<td>telephone</td>
<td>11</td>
</tr>
<tr>
<td>chair</td>
<td>10</td>
<td>hourglass</td>
<td>9</td>
<td>television</td>
<td>8</td>
</tr>
<tr>
<td>chandelier</td>
<td>10</td>
<td>key</td>
<td>9</td>
<td>toaster</td>
<td>8</td>
</tr>
<tr>
<td>chessboard</td>
<td>9</td>
<td>keyboard</td>
<td>8</td>
<td>toilet</td>
<td>9</td>
</tr>
<tr>
<td>chesspiece</td>
<td>12</td>
<td>kitchen sink</td>
<td>10</td>
<td>traffic cone</td>
<td>8</td>
</tr>
<tr>
<td>clock</td>
<td>10</td>
<td>knife</td>
<td>10</td>
<td>traffic light</td>
<td>9</td>
</tr>
<tr>
<td>computer</td>
<td>8</td>
<td>ladder</td>
<td>9</td>
<td>tree</td>
<td>11</td>
</tr>
<tr>
<td>couch</td>
<td>10</td>
<td>lamp</td>
<td>10</td>
<td>trombone</td>
<td>6</td>
</tr>
<tr>
<td>cross</td>
<td>10</td>
<td>lock</td>
<td>8</td>
<td>trumpet</td>
<td>7</td>
</tr>
<tr>
<td>cup</td>
<td>9</td>
<td>locomotive</td>
<td>9</td>
<td>umbrella</td>
<td>10</td>
</tr>
<tr>
<td>desk</td>
<td>10</td>
<td>machine gun</td>
<td>9</td>
<td>windmill</td>
<td>10</td>
</tr>
<tr>
<td>desk lamp</td>
<td>10</td>
<td>mailbox</td>
<td>8</td>
<td>wrench</td>
<td>15</td>
</tr>
<tr>
<td>dice</td>
<td>9</td>
<td>microwave</td>
<td>8</td>
<td>wrench</td>
<td>15</td>
</tr>
<tr>
<td>digital camera</td>
<td>10</td>
<td>motorcycle</td>
<td>16</td>
<td>yacht</td>
<td>10</td>
</tr>
<tr>
<td>dinosaur</td>
<td>8</td>
<td>palm tree</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>donut</td>
<td>8</td>
<td>person</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>door handle</td>
<td>8</td>
<td>piano</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doorknob</td>
<td>8</td>
<td>plant</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Total:** 942

**Mean per category:** 9.42
4 Blob-based occlusion

4.1 Occluded image examples

Figures 5-8 show examples of applying the blob-based occlusion algorithm for 10-40% occlusion applied to several different CU3D-100 exemplars.

Figure 5: Occluded image examples for a machine gun image.
Figure 6: Occluded image examples for a kitchen sink image.
Figure 7: Occluded image examples for a tree image.
Figure 8: Occluded image examples for an elephant image.
4.2 Occluded object recognition without voting

The results of recognition with the blob-based occlusion algorithm presented in the main text reflect a majority vote across multiple 2D affine transformations of a single image. Recognition under occlusion without this voting procedure (Figure 9) is qualitatively similar to those presented in the main text, but with an impairment in recognition performance across all models.

Figure 9: Results of occluded object recognition without voting.
References


