A continuous time neural model for sequential action: Supplemental Information

George Kachergis[1], Dean Wyatte[2], Randall C. O'Reilly[2],
Roy de Kleijn[1], and Bernhard Hommel[1]
george.kachergis@gmail.com
[1] Institute for Psychological Research &
Leiden Institute for Brain and Cognition, Leiden University
[2] Department of Psychology and Neuroscience,
University of Colorado Boulder

1 The Leabra Learning Algorithm

The Leabra framework is described in detail in ? (?, ?) and ? (?), 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 ? (?), 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.

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 (?, ?), 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.

1.1 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.

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.
   (b) During each cycle of settling, for all non-clamped units:
      i. Compute excitatory netinput ($g_e(t)$ or η_j, eq 1.3).
      ii. Compute kWTA inhibition for each layer, based on g_i^Θ (eq 1.9):
         A. Sort units into two groups based on g_i^Θ: top k and remaining $k+1$ to n.
         B. If basic, find k and $k+1th$ highest; if avg-based, compute avg of $1 \rightarrow k$ & $k+1 \rightarrow n$.
         C. Set inhibitory conductance g_i from g_k^Θ and g_{k+1}^Θ (eq 1.8).
      iii. Compute point-neuron activation combining excitatory input and inhibition (eq 1.1).
      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.

1.2 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

Parameter	Value	Parameter	Value
E_l	0.30	$\overline{g_l}$	0.10
E_i	0.25	$\overline{g_i}$	1.00
E_e	1.00	$\overline{g_e}$	1.00
V_{rest}	0.30	Θ	0.50
τ	.3	γ	80

Table 1: Parameters for the simulation (see equations in text for explanations of parameters). All are standard default parameter values.

(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 $1.0x10^{-8}$.

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

$$\Delta V_m(t) = \tau \sum_c g_c(t)\overline{g_c}(E_c - V_m(t)) \tag{1.1}$$

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 $\overline{g_c}$ 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:

$$V_m^\infty = \frac{g_e\overline{g_e}}{g_e\overline{g_e} + g_l\overline{g_l} + g_i\overline{g_i}} \tag{1.2}$$

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 (?, ?).

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

$$\eta_j = g_e(t) = \langle x_i w_{ij} \rangle = \frac{1}{n}\sum_i x_i w_{ij} \tag{1.3}$$

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 (?, ?), which has been found through various competitions to provide an excellent fit to the actual firing properties of cortical pyramidal neurons (?, ?), 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^Θ), where the threshold is indicated by Θ:

$$g_e^\Theta = \frac{g_i \overline{g}_i (E_i - V_m^\Theta) + \overline{g}_l (E_l - V_m^\Theta)}{\overline{g}_e (V_m^\Theta - E_e)} \tag{1.4}$$

$$y_j(t) \propto g_e(t) - g_e^\Theta \tag{1.5}$$

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:

$$y_j(t) = \frac{1}{\left(1 + \frac{1}{\gamma [g_e(t) - g_e^\Theta]_+}\right)} \tag{1.6}$$

where γ 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:

$$y_j^*(x) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-z^2/(2\sigma^2)} y_j(z - x) dz \tag{1.7}$$

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.

1.3 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 kth 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 (?, ?). 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:

$$g_i = g_{k+1}^\Theta + q(g_k^\Theta - g_{k+1}^\Theta) \tag{1.8}$$

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

$$g_i^\Theta = \frac{g_e^* \bar{g}_e (E_e - \Theta) + g_l \bar{g}_l (E_l - \Theta)}{\Theta - E_i} \tag{1.9}$$

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^Θ and g_{k+1}^Θ are set to the threshold inhibition value for the kth 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 kth unit to be sufficiently above the inhibitory threshold.

In the *average-based* kWTA version, g_k^Θ is the average g_i^Θ value for the top k most excited units, and g_{k+1}^Θ is the average of g_i^Θ 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.

1.4 XCAL Learning

The full treatment of the new XCAL version of learning in Leabra is presented in an in-preparation paper (?, ?), 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. (?, ?), and is closely related to the Bienenstock, Cooper & Munro (BCM) algorithm (?, ?). 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:

$$\Delta_{bcm} w_{ij} = xy(y - \langle y^2 \rangle) \tag{1.10}$$

(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 (?, ?).

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

Figure 1: 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 (?, ?) model, with a correlation value of $r = 0.89$.

this term are elaborated below) and a dynamic plasticity threshold θ_p (also elaborated below), which are integrated in the XCAL learning function (Figure 1):

$$\Delta_{xcal} w_{ij} = f_{xcal}(xy, \theta_p) \tag{1.11}$$

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

$$f_{xcal}(xy, \theta_p) = \begin{cases} (xy - \theta_p) & \text{if } xy > \theta_p \theta_d \\ -xy(1 - \theta_d)/\theta_d & \text{otherwise} \end{cases} \tag{1.12}$$

($\theta_d = .1$ 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 θ_p value).

The BCM equation produces a curved quadratic function that has the same qualitative shape as the XCAL function (Figure 1). 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:

$$\Delta_{chl} = x^+ y^+ - x^- y^- \tag{1.13}$$

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; (?, ?)). 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:

$$\Delta_{xcal-err} w_{ij} = f_{xcal}(x_s y_s, x_m y_m) \tag{1.14}$$

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:

$$\Delta_{xcal-err} w_{ij} = f_{xcal}(\kappa x_s y_s + (1-\kappa) x_m y_m, x_m y_m) \tag{1.15}$$

(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:

$$\Delta_{xcal-so} w_{ij} = f_{xcal}(\kappa x_s y_s + (1-\kappa) x_m y_m, \gamma_l y_l) \tag{1.16}$$

where $\gamma_l = 3$ 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 θ_p threshold that reflects a combination of both the self-organizing long-term average, and the medium-term minus-phase like average:

$$\Delta_{xcal} w_{ij} = f_{xcal}(\kappa x_s y_s + (1-\kappa) x_m y_m, \lambda\gamma y_l + (1-\lambda) x_m y_m) \tag{1.17}$$

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:

$$\Delta_{sb} w_{ij} = [\Delta_{xcal}]_+ (1 - w_{ij}) + [\Delta_{xcal}]_- w_{ij} \tag{1.18}$$

where the $[]_+$ and $[]_-$ 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:

$$\hat{w}_{ij} = \frac{1}{1 + \left(\frac{w_{ij}}{\theta(1-w_{ij})}\right)^{-\gamma}} \tag{1.19}$$

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

1.5 LeabraTI context updating and learning

TO DO!!

References

Last updated: December 4, 2013