APPENDIX I

The Leabra framework used for implementing the model is described in detail in O'Reilly and Munakata (2000) and O'Reilly (2001), and summarized here. This framework has been used in over 40 different models in O'Reilly and Munakata (2000), and a number of other research models. The current model therefore represents an extension to a systematic modeling framework using standardized mechanisms. The model can be obtained by emailing the corresponding author.

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.

For each event:

1. Iterate over minus (−), plus (+), and update (++) phases of settling for each event.

    (a) At start of settling:

        i. For non-PFC/BG units, initialize state variables (activation, v m, etc).

        ii. Apply external patterns (clamp input in minus, input & output, external reward based on minus-phase outputs).

    (b) During each cycle of settling, for all non-clamped units:

        i. Compute excitatory netinput ($g_e(t)$ or η_j, eq 2) (eq 21 for SNr/Thal units).

ii. For Striatum Go/NoGo units in ++ phase, compute additional excitatory and inhibitory currents based on DA inputs from SNc (eq 20).

iii. Compute kWTA inhibition for each layer, based on

A. Sort units into two groups based on g

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

iv. Compute point-neuron activation combining excitatory input and inhibition

(c) After settling, for all units:

i. Record final settling activations by phase

ii. At end of + and ++ phases, toggle PFC maintenance currents for stripes with SNr/Thal act > threshold (.1).

2. After these phases, update the weights (based on linear current weight values):

(a) For all non-BG connections, compute error-driven weight changes (eq 8) with soft weight bounding (eq 9), and hebbian weight changes from plus-phase activations (eq 7), and overall net weight change as weighted sum of error-driven and hebbian (eq 10).

(b) For PV units weight changes are given by delta rule computed as difference between plus phase external reward value and minus phase expected rewards (eq 11).

(c) For LV units, only change weights (using eq 13) if PV expectation $>\theta_{pv}$ or external re-ward/punishment actually delivered.

(d) For Striatum units, weight change is the delta rule on DA-modulated second-plus phase activations minus unmodulated plus phase acts (eq 19).

(e) Increment the weights according to net weight change.

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

$$\Delta V_m(t) = \tau \sum_c g_c(t)\bar{g}_c(E_c - V_m(t)) \quad (1)$$

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 g_c that controls the relative influence of the different conductances.

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:

$$\eta_j = g_e(t) = \langle x_i w_{ij} \rangle = \frac{1}{n}\sum_i x_i w_{ij} \quad (2)$$

The inhibitory conductance is computed via the kWTA function described in the next section, and leak is a constant. Activation communicated to other cells (y_j) is a thresholded (Θ) sigmoidal function of the membrane potential with gain parameter γ:

$$y_j(t) = \frac{1}{(1+\frac{1}{\gamma[V_m(t)-\theta]_+})} \quad (3)$$

where $[x]_+$ 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. 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:

$$y_j^*(x) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{z^2}{2\sigma^2}} y_j(z-x)dz \quad (4)$$

where x represents the $[V_m(t) - \Theta]_+$ 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.

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 g_i for all units in the layer, such that the k +1th most excited unit within a layer is generally below its firing threshold, while the *k*th is typically above threshold:

$$g_i = g^{\theta}_{k+1} + q(g^{\theta}_k - g^{\theta}_{k+1}) \quad (5)$$

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:

$$g^{\theta}_i = \frac{g^*_e \bar{g}_e (E_e - \theta) + g_l \bar{g}_l (E_l - \theta)}{\theta - E_i} \quad (6)$$

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 *k*th and *k* +1th value for the top most excited units, respectively. In the average-based kWTA version used here, 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.

Hebbian and Error-Driven Learning

Leabra uses a combination of error-driven and Hebbian learning. Error-driven learning in Leabra is the symmetric midpoint version of the GeneRec algorithm (O'Reilly, 1996), which is functionally equivalent to contrastive Hebbian learning (CHL). The network settles in two

distinct phases, an expectation (minus) phase where the network's produces an output, and an outcome (plus) phase where the target output is experienced. The network then computes the difference of a pre and postsynaptic activation product between these two phases. For Hebbian learning, Leabra uses essentially the same learning rule used in competitive learning which can be seen as a variant of the Oja normalization (Oja, 1983). The error-driven and Hebbian learning components are combined additively at each connection to produce a net weight change.

The equation for the Hebbian weight change is:

$$\Delta_{hebb} w_{ij} = x_i^+ y_j^+ - y_j^+ w_{ij} = y_j^+ (x_i^+ - w_{ij}) \quad (7)$$

and for error-driven learning using CHL:

$$\Delta_{err} w_{ij} = \left(x_i^+ y_j^+\right) - (x_i^- y_j^-) \quad (8)$$

which is subject to a soft-weight bounding to keep within the $0-1$ range:

$$\Delta_{sberr} w_{ij} = [\Delta_{err}] + \left(1 - w_{ij}\right) + [\Delta_{err}] - w_{ij} \quad (9)$$

The two terms are then combined additively with a normalized mixing constant k_{hebb}:

$$\Delta w_{ij} = \in [k_{hebb}(\Delta_{hebb}) + (1 - k_{hebb})(\Delta_{sberr})] \quad (10)$$

PVLV Equations

See Hazy, Frank & O'Reilly (2010), O'Reilly, Frank, Hazy, & Watz (2007) and O'Reilly & Frank (2006) for further details on the PVLV system. We assume that time is discretized into steps that correspond to environmental events (e.g., the presentation of a CS or US). All of the following equations operate on variables that are a function of the *current* time step t – we omit the t in the notation because it would be redundant. PVLV is composed of two systems, PV (primary value) and LV (learned value), each of which in turn are composed of two subsystems (excitatory and inhibitory). Thus, there are four main value representation layers in PVLV (PVe, PVi, LVe, LVi), which then drive the dopamine (DA) layers (VTA/SNc). There are several changes in the algorithm from this previous work (most notably the inclusion of the PVr and NV systems; see *Learning Rules* section below). These changes are efforts to increase the biological plausibility of the system (e.g. removing synaptic depression), and will be discussed in detail in a future work. The simulations and results described in this paper were only performed using the PVLV system described here; the changes to the algorithms described here were developed completely independently.

Value Representations

The PVLV value layers use standard Leabra activation and kWTA dynamics as described above, with the following modifications. They have a three-unit distributed representation of the scalar values they encode, where the units have preferred values of (0, .5, 1). The overall value represented by the layer is the weighted average of the unit's activation times its preferred value,

and this decoded average is displayed visually in the first unit in the layer. The activation function of these units is a "noisy" linear function (i.e., without the x/(x + 1) nonlinearity, to produce a linear value representation, but still convolved with gaussian noise to soften the threshold, as for the standard units, equation 4), with gain $\gamma = 220$, noise variance $\sigma = .01$, and a lower threshold $\Theta = .17$. The k for kWTA (average based) is 1, and the q value is .9 (instead of the default of .6 in other layers). These values were obtained by optimizing the match for value represented with varying frequencies of 0-1 reinforcement (e.g., the value should be close to .4 when the layer is trained with 40% 1 values and 60% 0 values). Note that having different units for different values, instead of the typical use of a single unit with linear activations, allows much more complex mappings to be learned. For example, units representing high values can have completely different patterns of weights than those encoding low values, whereas a single unit is constrained by virtue of having one set of weights to have a monotonic mapping onto scalar values.

Learning Rules.

The PVe layer does not learn, and is always just clamped to reflect any received reward value (r). By default we use a value of 0 to reflect negative feedback, .50 for no feedback, and 1 for positive feedback (the scale is arbitrary). The PVi layer units (y_j) are trained at every point in time to produce an expectation for the amount of reward that will be received at that time. In the minus phase of a given trial, the units settle to a distributed value representation based on sensory inputs. This results in unit activations y_j^-, and an overall weighted average value across these units denoted PV_i. In the plus phase, the unit activations (y_j^+) are clamped to represent the

actual reward r (a.k.a., PV_e). The weights (w_{ij}) into each PV_i unit from sending units with plus-phase activations x_i^+ , are updated using the delta rule between the two phases of PVi unit activation states:

$$\Delta w_{ij} = \in \left(y_j^+ - y_j^-\right)x_i^+ \tag{11}$$

This is equivalent to saying that the US/reward drives a pattern of activation over the PVi units, which then learn to activate this pattern based on sensory inputs. In addition to the PV_e and PV_i layers there is an additional PV_r layer that is associated with learning about reward detection. This system learns exactly the same way as the PV_i system, but has a slower learning rate for weight decreases relative to increases. The LVe and LVi layers learn in much the same way as the PVi layer (Equation 11), except that the PV system filters the training of the LV values, such that they only learn from actual reward outcomes or when reward is expected by the PVr system, and not when no rewards are present or expected. This condition is as follows:

$$PV_{filter} = min(PV_r, PV_i) < \theta_{min} \vee max(PV_r, PV_i) > \theta_{max} \tag{12}$$

$$\Delta w_i = \begin{cases} \in \left(y_j^+ - y_j^-\right)x_i^+ & \text{if } PV_{filter} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where Θ_{min} is a lower threshold (0.20 by default), below which negative feedback is indicated and Θ_{max} is an upper threshold (0.80), above which positive feedback is indicated (otherwise, no feedback is indicated). Biologically, this filtering requires that the LV systems be driven directly

by primary rewards (which is reasonable, and required by the basic learning rule anyway) and that they learn from DA dips driven by high PVr expectations of reward that are not met. The only difference between the LVe and LVi systems is the learning rate $\in$, which is .05 for LVe and .001 for LVi. Thus, the inhibitory LVi system serves as a slowly integrating inhibitory cancellation mechanism for the rapidly adapting excitatory LVe system.

Finally, the NV layer signals stimulus novelty and produces dopamine bursts for novel stimuli, which slowly decay in magnitude as a stimulus becomes familiar. The habituation for this system is simply:

$$\Delta w_i = -NVx_i \tag{14}$$

The PV, LV, and NV distributed value representations drive the dopamine layer (VTA/SNc) activations in terms of the difference between the excitatory and inhibitory terms for each. Thus, there is a PV delta, an LV delta, and an NV delta:

$$\delta_{pv} = PV_e - PV_i \tag{15}$$

$$\delta_{lv} = LV_e - LV_i \tag{16}$$

$$\delta_{nv} = NV \tag{17}$$

The dopamine (DA) system integrates each of these inputs, using a temporal derivative computation to only produce brief bursts or dips relative to a baseline level of activation (this is

the primary difference from the synaptic depression mechanism used in the earlier published version). The key issue is when to use each of the above values: If primary rewards are present or expected but not present, then the PV system dominates, and otherwise, LV + NV drive it. With the differences in learning rate between LVe (fast) and LVi (slow), the LV delta signal reflects recent deviations from expectations and not the raw expectations themselves, just as the PV delta reflects deviations from expectations about primary reward values. This is essential for learning to converge and stabilize when the network has mastered the task. These two delta signals need to be combined to provide an overall DA delta value, as reflected in the firing of the VTA and SNc units. One sensible way of doing so is to have the PV system dominate at the time of primary rewards, whereas the LV system dominates otherwise, by using the same PV-based filtering as holds in the LV learning rule:

$$\delta = \begin{cases} \left(\delta_{pv}^{t} - \delta_{pv}^{(t-1)}\right) & if\ PV_{filter} \\ \left(\delta_{lv}^{t} - \delta_{lv}^{(t-1)}\right) + \left(\delta_{nv}^{t} - \delta_{nv}^{(t-1)}\right) & otherwise. \end{cases} \quad (18)$$

Special Basal Ganglia Mechanisms

Striatal Learning Function

Each stripe (group of units) in the Striatum layer is divided into Go vs. NoGo in an alternating fashion. The DA input from the SNc modulates these unit activations in the update phase by providing extra excitatory current to Go and extra inhibitory current to the NoGo units in proportion to the positive magnitude of the DA signal, and vice-versa for negative DA

magnitude. This reflects the opposing influences of DA on these neurons (Frank, 2005; Gerfen, 2001). This update phase DA signal reflects the PVLV system's evaluation of the PFC updates produced by gating signals in the plus phase. Learning on weights into the Go/NoGo units is based on the activation delta between the update (++) and plus phases:

$$\Delta w_i = \in x_i(y^{++} - y^{+}) \tag{19}$$

To reflect the finding that DA modulation has a contrast-enhancing function in the striatum (Frank, 2005; Nicola, Surmeier, & Malenka, 2000; Hernandez-Lopez, Bargas, Surmeier, Reyes, & Galarraga, 1997), and to produce more of a credit-assignment effect in learning, the DA modulation is partially a function of the previous plus phase activation state:

$$g_e = \gamma[da]_+ y^+ + (1-\gamma)[da]_+ \tag{20}$$

where $0 < \gamma < 1$ controls the degree of contrast enhancement (.5 is used in all simulations), $[da]_+$ is the positive magnitude of the DA signal (0 if negative), y_+ is the plus-phase unit activation, and g_e is the extra excitatory current produced by the da (for Go units). A similar equation is used for extra inhibition (g_i) from negative da ($[da]_-$) for Go units, and vice-versa for NoGo units.

SNrThal Units

The SNrThal units provide a simplified version of the SNr/GPe/Thalamus layers. They receive a net input that reflects the normalized Go -NoGo activations in the corresponding Striatum stripe:

$$\eta_j = \left[\frac{\sum Go - \sum NoGo}{\sum Go + \sum NoGo}\right]_+ \quad (21)$$

(where []+ indicates that only the positive part is taken; when there is more NoGo than Go, the net input is 0). This net input then drives standard Leabra point neuron activation dynamics, with kWTA inhibitory competition dynamics that cause stripes to compete to update the PFC. This dynamic is consistent with the notion that competition/selection takes place primarily in the smaller GP/SNr areas, and not much in the much larger striatum (e.g., Mink, 1996; Jaeger, Kita, & Wilson, 1995). The resulting SNrThal activation then provides the gating update signal to the PFC: if the corresponding SNrThal unit is active (above a minimum threshold; .1), then active maintenance currents in the PFC are toggled.

This SNrThal activation also multiplies the per-stripe DA signal from the SNc:

$$\delta_j = snr_j \delta \quad (22)$$

where snr_j is the snr unit's activation for stripe j, and δ is the global DA signal.

Random Go Firing

The PBWM system only learns after Go firing, so if it never fires Go, it can never learn to improve performance. One simple solution is to induce Go firing if a Go has not fired after some threshold number of trials. However, this threshold would have to be either task specific or set very high, because it would effectively limit the maximum maintenance duration of the PFC (because by updating PFC, the Go firing results in loss of currently maintained information). Therefore, we have adopted a somewhat more sophisticated mechanism that keeps track of the average DA value present when each stripe fires a Go:

$$\overline{da}_k = \overline{da}_k + \in (da_k - \overline{da}_k) \tag{23}$$

If this value is < 0 and a stripe has not fired Go within 1 or 2 trials (in the 2-back and 3-back respectively), a random Go firing is triggered with some probability (.1). We also compare the relative per-stripe DA averages, if the per-stripe DA average is low but above zero, and one stripe's dak is .05 below the average of that of the other stripes:

$$if\left(\overline{da}_k < .1\right) and \left(\overline{da}_k - \langle\overline{da}\rangle < -.05\right); Go \tag{24}$$

a random Go is triggered, again with some probability (.1). Finally, we also fire random Go in all stripes with some very low baseline probability (.0001) to encourage exploration.

When a random Go fires, we set the SNrThal unit activation to be above Go threshold, and we apply a positive DA signal to the corresponding striatal stripe, so that it has an opportunity to learn to fire for this input pattern on its own in the future.

PFC Maintenance

PFC active maintenance is supported in part by excitatory ionic conductances that are toggled by Go firing from the SNrThal layers. This is implemented with an extra excitatory ion channel in the basic V_m update equation (1). This channel has a conductance value of .5 when active. See Frank, Loughry, & O'Reilly (2001) for further discussion of this kind of maintenance mechanism. The first opportunity to toggle PFC maintenance occurs at the end of the first plus phase, and then again at the end of the second plus phase (third phase of settling). Thus, a complete update can be triggered by two Go's in a row, and it is almost always the case that if a Go fires the first time, it will fire the next, because striatum firing is primarily driven by sensory inputs, which remain constant.

APPENDIX II.

Computations Supporting Manual Output: Match vs. Non-match Decision

As noted in the main text, the network must learn to produce not only the correct verbal output (corresponding to the *n*-back item) but also a manual output (corresponding to whether the current item matches or does not match the *n*-back item). This match vs. non-match decision can be computed by the network simply by comparing the activation patterns in the input with those in the verbal output layer and prefrontal cortex. Indeed, it is precisely this form of "coincidence detection" that is accomplished by the posterior cortical layer.

We confirmed that "coincidence detection" between the verbal output layer and stimulus input layer was the underlying computation performed by the posterior cortical layers as follows. First, we examined those units in the posterior cortical layer that received strong projections from corresponding units in the input and verbal output layers (e.g., large weights from the "A" stimulus in both layers, or from the "B" stimulus in both layers, as indicated by a positive correlation of weights from these two layers). We found that these units projected disproportionately strongly to the target output response than to the nontarget response, relative to those posterior cortical units that do not show this correspondence of weights (e.g., strong weights from the "A" stimulus in the input layer but weak weights from the "A" stimulus in the verbal output layer): ($F(1,98)=4.482$, $p<.05$). Thus, the target manual output is driven largely by those posterior cortical units that are themselves strongly activated by matches between the input and verbal output layers.

As such, the match/nonmatch decision relies not only on coincidence detection mechanisms, but also on mechanisms supporting activation of the correct verbal output – a requirement fulfilled by the connectivity of striatal areas with parietal areas, which trigger the

gating of prefrontal information into the verbal output layer. Thus the match/nonmatch response can be seen as a cumulative result of the network's behavior in total, although it is most directly supported by weight-based computations occurring in the posterior cortical layer.

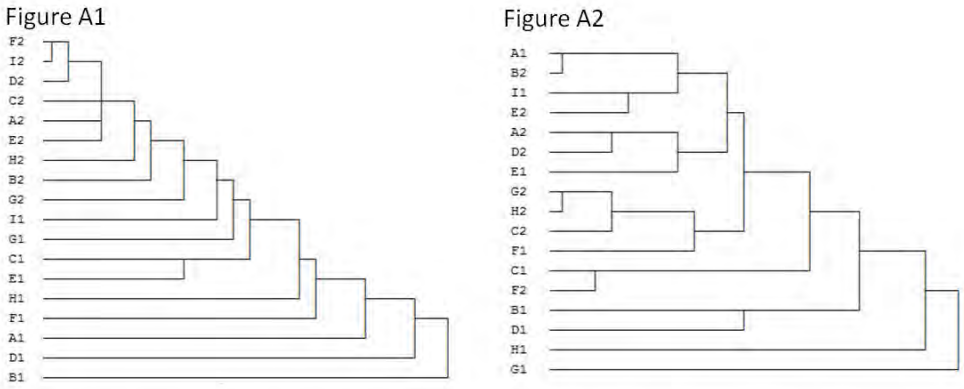
The Underlying Source of Recent Lure Errors

Our approach to identify the source of recent lure errors was to examine in detail the performance of one network performing the 2-back task over the final 7 epochs of training. We first determined that the match/nonmatch decision was typically being performed correctly by the posterior cortical layer (i.e., detecting matches between the recalled verbal output and the stimulus present in the input). Only 25% of recent lure errors reflected a failure to respond to matches/mismatches between the (correct) verbal output and stimulus input layers. That is, the prefrontal layers recalled the correct information to the verbal output layer, but the posterior cortical layer incorrectly responded as though this information matched the information presented in the input.

Nonetheless, approximately 75% of recent lure errors reflected recall of the 1-back instead of the 2-back item in the verbal output layer. To determine whether item confusion within the relevant prefrontal stripe was to blame for this incorrect recall, we examined the representational differentiation among items in prefrontal layers that were gated on a particular trial, using a similar cluster plot analysis as presented in the main text. In particular, we recorded the activations in a prefrontal stripe with a preferred serial order of 1 across the final 7 epochs of training. For each unit in this stripe, we averaged activations across correct trials and incorrect trials separately, conditional on the verbal output for that trial and the current trial's serial order. Finally, we constructed separate cluster plots for correct and incorrect trials to visualize the

differentiation of prefrontal representations of each item x order combination.

This analysis indicated that stripes demonstrated good differentiation among items of the preferred serial order on correct recent lure trials (Figure 1), but a much more haphazard pattern of representational differentiation on incorrect recent lure trials (Figure 2). Thus, recent lure errors are associated with increased item confusion within the prefrontal layers.



Figures 1 & 2. Cluster plots of prefrontal activations on correct (Figure 1) and incorrect (Figure 2) recent lure trials. On correct trials, different items of the preferred serial order for this stripe are well differentiated in prefrontal activations, as indicated by the relatively long paths interconnecting items of serial order one (A1, B1, C1, etc). Items of a dispreferred serial order are not as well differentiated, as indicated by relatively shorter paths interconnecting those items (A2, B2, C2, etc). In contrast, on incorrect recent lure trials (Figure 2), there is a much more haphazard pattern of differentiation among items in terms of prefrontal activations, indicative of item confusion.

In principle, this item confusion within the prefrontal layers could arise from a gating error. That is, this prefrontal stripe may have been gated inappropriately on the current trial, or some other recent trial, and been exposed to items of a dispreferred serial order. In this case, poor differentiation of items would reflect that this stripe had been updated with information that it was poorly suited to represent. However, we found no appreciable differences in the striatal activations between correct and incorrect trials – neither on the trial where the incorrect verbal output was provided, nor on either of the two preceding trials. Thus, prefrontal stripes were gated similarly on incorrect and correct recent lure trials, as well as on the trials immediately preceding them, indicating that gating errors are not a source of the item confusion occurring on

incorrect recent lure trials.

If not due to gating, what could be the source of the item confusion occurring on incorrect recent lure trials? We found that the haphazard pattern of representational differentiation in prefrontal activation states on incorrect recent lure trials – i.e., item confusion – was paralleled by haphazard patterns of *net input* to prefrontal layers on incorrect recent lure trials. Whereas net input to prefrontal layers was substantially different in terms of whether the current trial was of serial order 1 or 2 on correct recent lure trials (Figure A3), incorrect recent lure trials showed much more similar net input to prefrontal layers across trials of serial order 1 and 2 (Figure A4). This result indicates that recent lure errors arise from an instability of prefrontal activation states independent of gating: The clean separation between representations of items of different serial orders is corrupted on incorrect recent lure trials, both in terms of prefrontal activations and net input to prefrontal layers.

We conducted further analyses of representational differentiation on the trial *preceding* incorrect and correct recent lure trials, but found no appreciable differences in the prefrontal representations on the trials preceding recent lure errors relative to the representations on the trials preceding correct rejections of recent lures. This similarity indicates that the corruption of prefrontal representations on incorrect recent lure trials is due to the recent lure itself, and not to a corruption of the representation of the 2-back stimulus occurring prior to the recent lure. Our model thus indicates that recent lure errors occur due to a lack of stability of prefrontal representations to interference arising from the recent lure itself.

In summary, these analyses suggested that recent lure errors did not arise because of gating problems, but rather because of non-robust representations in prefrontal cortex that were susceptible to interference from incoming stimuli. These particular representations may have

been susceptible to interference from lures to the extent that they were similar to the 1-back stimulus, perhaps as a result of Hebbian learning in the sequences leading up to recent lure errors.

Figure A3

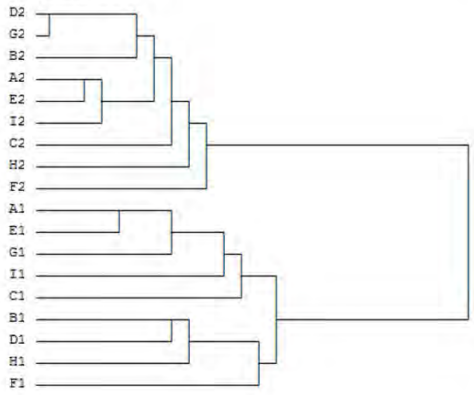
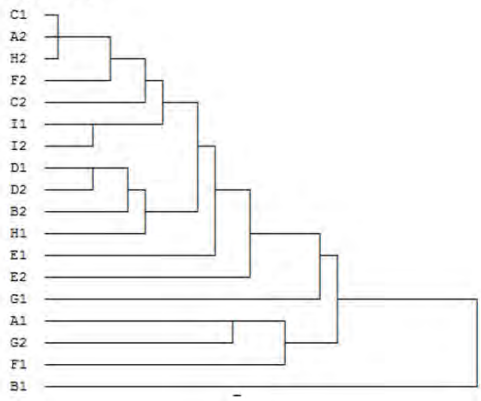


Figure A4



Figures 3 & 4. Cluster plots of prefrontal net input on correct (Figure 1) and incorrect (Figure 2) lure trials. Although net input to prefrontal layers on correct recent lure trials (Figure 1) was reliably differentiated in terms of the serial order of the current trial (as indicated by the large cluster separating items of serial order 2 from those of serial order 1), net input to prefrontal layers on incorrect recent lure trials (Figure 2) was not reliably differentiated according to serial order. Thus, the representational differentiation among items of different serial orders in terms of activation states (Figure 1 & 2) was paralleled by differentiation among items of different serial orders in terms of net input.

Hebbian and Error-driven Computations Contributing to Item Differentiation in the Prefrontal Layers

The binding of items to context in our n-back model relies on two principle developments: the development of an order-based striatal gating signal as a result of reinforcement learning, and the increasing prefrontal differentiation of items occurring with a preferred serial order as a result of Hebbian and error-driven learning. As discussed in the main text, the order-based gating policy develops as a result of reinforcement learning because it is supported by strong connectivity between the parietal and striatal layers, but also because it maximizes reinforcement relative to alternative gating policies.

In contrast, increasing representational differentiation in the prefrontal layers develops via Hebbian and error-driven learning processes over repeated training experiences. To see why Hebbian and error-driven learning lead naturally to this kind of representational differentiation, consider an incorrect trial on the 2-back task, where the serial-order based gating policy had correctly updated a prefrontal stripe with the "A" stimulus presented 2 trials previously, but the prefrontal representation of this "A" stimulus is not yet sufficiently distinct from its representation of other stimuli. This indistinct prefrontal representation may bias the posterior cortical and verbal output layers such that the "B" unit in the verbal output layer is ultimately activated instead of the correct "A" unit. Thus, there will be a resulting difference in activation states between the incorrect answer (produced during Leabra's minus phase, as described in Appendix I) and the correct answer (produced during Leabra's plus phase, also described in Appendix I). This difference will lead to an error-driven learning signal that changes specifically those weights – from the prefrontal layer that was gated on this trial to the verbal output and posterior cortical layer with which the prefrontal layers are connected – that served to conflate the "B" and "A" stimuli. In addition, Hebbian learning will further strengthen connections among those (correct) units that are simultaneously activated in Leabra's plus phase. Iterative learning of this type eventually converges to yield prefrontal representations that maximally distinguish the stimuli that any given stripe must represent, so that such errors are not produced. Thus, because each stripe eventually contains representations of items occurring with only one particular serial order (due to the order-based gating policy learned by the striatum), error-driven and Hebbian learning only ever train stripes to maximally distinguish those stimuli of that preferred serial order.