

Running Head: How Inhibitory Oscillations Can Train Neural Networks and Punish Competitors

How Inhibitory Oscillations Can Train Neural Networks and Punish Competitors

Kenneth A. Norman, Ehren Newman, Greg Detre, & Sean Polyn

Department of Psychology
Princeton University
Green Hall
Princeton, NJ 08544
{knorman, enewman, gdetre}@princeton.edu
polyn@psych.upenn.edu

The first two authors contributed equally to this research

In press at *Neural Computation*
3rd January 2006

Abstract

We present a new learning algorithm that leverages oscillations in the strength of neural inhibition to train neural networks. Raising inhibition can be used to identify weak parts of target memories, which are then strengthened. Conversely, lowering inhibition can be used to identify competitors, which are then weakened. To update weights, we apply the Contrastive Hebbian Learning equation to successive time steps of the network. The sign of the weight change equation varies as a function of the phase of the inhibitory oscillation. We show that the learning algorithm can memorize large numbers of correlated input patterns without collapsing, and that it shows good generalization to test patterns that do not exactly match studied patterns.

Contents

Introduction	3
Data Indicating Competitor-Punishment	3
The Learning Algorithm	4
The Role of Inhibition	4
Precis of the Learning Algorithm	4
Implementing the Algorithm Using Inhibitory Oscillations	5
Network Architecture and Biological Relevance	6
General Simulation Methods	9
Simulations	9
Simulation 1: Omnidirectional Pattern Com- pletion as a Function of Input-Pattern Overlap and Test-Pattern Noise	10
Methods	10
Results	11
Simulation 2: 3-Layer Autoencoder	14
Autoencoder methods	15
Results of Autoencoder Simulations	16
Discussion	18
Functional Properties of the Learning Algorithm	18
Relating the Oscillating Algorithm to Neural Theta Oscillations	18
Applications to Hippocampal Architectures	19
Relating the Oscillating Algorithm to BCM	19
Applying the Oscillating Algorithm to Psycho- logical Data	19
Conclusions	20
Appendix A: Model Parameters	20
Basic Network Parameters	20
Oscillation Parameters	20
Appendix B: Effects of Hidden-Layer Size	20
Appendix C: Effects of Hidden-Layer Oscillations	20
Acknowledgements	21
References	24

Introduction

The idea that memories compete to be retrieved is one of the most fundamental axioms of neural processing. According to this view, retrieval success is a function of the amount of input that the target memory receives, relative to other, competing memories. If the target memory receives substantially more input than competing memories, it will be retrieved quickly and accurately; but if support for the target memory is low, relative to competing memories, the target memory will be retrieved slowly or not at all.

This view implies that, to maximize subsequent retrieval success, the memory system can enact two distinct kinds of changes. The most obvious way to improve retrieval is to strengthen the target memory. However, it should also be possible to improve retrieval by *weakening competing memories*. Over the past decade, the idea that competitors are punished has received extensive empirical support. Studies by Michael Anderson and others have demonstrated the following regularity: If a memory receives input from the retrieval cue, but the memory is not ultimately retrieved, then the memory is weakened. Furthermore, this weakening appears to be proportional to the amount of input that a competitor receives (e.g., Anderson, 2003). Put simply: The more that a memory competes (so long as it is not ultimately retrieved), the more it is punished. We review some illustrative findings in the *Data Indicating Competitor Punishment* section below.

However, despite the obvious functional utility of punishing competitors, and the large body of psychological research indicating that competitor punishment occurs, extant computational models of memory have not directly addressed the issue of competitor punishment. In the *Learning Algorithm* section below, we present a theory of how the brain can exploit regular oscillations in neural inhibition to punish competing memories and strengthen weak parts of target memories. In the *Simulations* section, we explore the functional properties of our oscillation-based learning algorithm: How well can it store patterns, relative to other learning algorithms that do not explicitly incorporate competitor-punishment; and, how does increasing overlap between patterns affect the learning algorithm's ability to store item-specific features of individual patterns?

The ultimate goal of this work is to show how explicitly incorporating competitor-punishment mechanisms into neural learning algorithms can:

- improve the algorithms' ability to memorize overlapping patterns
- improve our understanding of how inhibitory oscil-

lations (e.g., theta oscillations) contribute to learning

Data Indicating Competitor-Punishment

The phenomenon of competitor punishment is nicely illustrated by data from Michael Anderson's Retrieval Induced Forgetting (RIF) paradigm (for a comprehensive overview of RIF results, see Anderson, 2003). In the RIF paradigm, participants are given a list of category-exemplar pairs (e.g. Fruit-Apple, Fruit-Kiwi, and Fruit-Pear) one at a time and are told to memorize the pairs. Immediately after viewing the pairs, participants are given a practice phase where they practice retrieving a subset of the items on the list (e.g. they are given Fruit-Pe___ and must say Pear). After a delay (e.g., 20 minutes), participants' memory for all of the items on the study list is tested. Practicing Fruit-Pe___ improves recall of Pear but hurts recall of competing items (e.g., Apple). Importantly, Anderson and Spellman (1995) found that reduced recall of Apple is evident even when subjects are given *independent cues* that were not presented at study (e.g., Red-A___). This finding indicates that the Apple representation itself has been weakened, and not just the Fruit-Apple connection. Anderson, Bjork, and Bjork (1994) also found that practicing Fruit-Pe___ results in more punishment for strong associates of Fruit (Apple) than weak associates of Fruit (Kiwi). Intuitively, strong associates compete more than weak associates, so they suffer more competitor-punishment. The fact that impaired recall of the competitor (relative to baseline) lasts on the order of tens of minutes, and possibly longer, suggests that impaired recall is due to changes in synaptic weights, as opposed to carryover of activation states from the retrieval practice phase.

Anderson's retrieval-induced forgetting experiments are a particularly well-characterized example of competitor-punishment. Importantly, though, they are not the only example of this dynamic. Below, to illustrate the generalized nature of this phenomenon, we briefly review findings from other domains that can be understood in terms of competitor-punishment:

- Metaphor comprehension: Glucksberg, Newsome, and Goldvarg (2001) showed that word meanings that are not applicable to the current sentence suffer lasting inhibition. For example, after reading "My lawyer is a shark", participants were slower to evaluate sentences that reference the concept "swim" (e.g., "Geese are good swimmers"). Glucksberg et al. argue that, when participants read "My lawyer is a shark", less appropriate meanings of shark (e.g., "swim") compete with more appropriate meanings (e.g., "vicious"). "Swim" receives input but loses the competition so it is punished.

- Task switching: Mayr and Keele (2000) found that, after switching from Task A to Task B, it is more difficult to switch back to Task A than to switch to a new task (Task C). This can be explained in terms of Task A competing with Task B during the initial switch. Because Task A competes but loses (i.e., participants eventually succeed in switching to Task B), the neural representation of Task A is punished, thereby making it more difficult to re-activate later.
- Negative priming: In negative priming tasks, participants have to process one object and ignore other objects on each trial (e.g., participants might be instructed to name the green object and ignore objects that are not green). Objects that were ignored sometimes re-appear as target objects on subsequent trials. Studies have found that participants are slower to process objects that were ignored on previous trials, compared to objects that were not ignored (see Fox, 1995 for a review). This can be explained in terms of the idea that non-target objects compete with target objects. By attending to the target color, participants bias the competition such that the target object wins the competition. Because the representations of non-target objects receive strong input from the stimulus array but lose the competition, these representations are punished.
- Cognitive dissonance reduction: Freedman (1965) showed that if a child is given a mild threat not to play with a toy (and they don't play with the toy), they end up liking the toy less. In contrast, if a child is given a strong threat not to play with a toy, there is no attitude change. This paradigm can be understood in terms of a competition between wanting to play with the toy ("play") and not wanting to play with the toy ("don't play"). In the mild threat condition, "don't play" just barely wins over "play"; "play" is a losing competitor so it is punished, resulting in reduced liking. In the strong threat condition, "don't play" wins easily over "play"; there is not strong competition, so "play" is not punished.

The ubiquitous presence of competitor-punishment in psychology spurred us to develop a learning mechanism that can account for this dynamic. Our goal is to come up with a neural network learning algorithm that punishes representations if and only if they compete (and lose) during retrieval. Also, the learning algorithm should be able to efficiently train new patterns into the network, in a manner that supports subsequent recall of these patterns. Ultimately we believe that these goals are synergistic: Pushing away competitors during training should improve the accuracy of recall at test.

The Learning Algorithm

In this section, we present a learning algorithm for rate-coded neural networks that can punish competitors as well as train new patterns into a network. The algorithm depends critically on changes in the strength of neural inhibition. By way of background, we first review how inhibition works in our model. Then, we provide an overview of how the learning algorithm works. Finally, we provide a more detailed account of how the learning algorithm exploits changes in neural inhibition to punish competitors and strengthen weak memories.

The Role of Inhibition

Neural systems need some way of controlling excitatory neural activity so this activity does not spread across the entire system, causing a seizure. In keeping with O'Reilly and Munakata (2000), we argue that inhibitory interneurons act to control excitatory activity. Inhibitory interneurons accomplish this goal by sampling the overall amount of excitatory activity within a particular region via diffuse input projections, and sending back a commensurate amount of inhibition via diffuse output projections. In this manner, inhibitory interneurons act to enforce a *set point* on the amount of excitatory activity. Increasing the strength of inhibition leads to a decrease in the overall amount of excitatory activity and reducing the strength of inhibition leads to an increase in the overall amount of excitatory activity. In terms of this framework, an excitatory neuron will be active if the amount of excitation that it receives is sufficient to counteract the amount of inhibition that it receives. These active units make up the representation of the input pattern. Units that receive a substantial amount of excitatory input, but not enough to counteract the effects of inhibition, can be thought of as competitors. Given processing noise, it is possible that these competing units could be recalled in place of target units.

Precis of the Learning Algorithm

The learning algorithm utilizes the Contrastive Hebbian Learning (CHL) weight change equation (Ackley, Hinton, & Sejnowski, 1985; Hinton & Sejnowski, 1986; Hinton, 1989; Movellan, 1990). CHL learning involves contrasting a more desirable state of network activity (sometimes called the *plus* state) with a less desirable state of network activity (sometimes called the *minus* state). The CHL equation adjusts network weights to strengthen the more desirable state of network activity (so it is more likely to occur in the future) and weaken the less desirable state of network activity (so it is less likely to occur in the future).

$$dW_{ij} = \epsilon ((X_i^+ Y_j^+) - (X_i^- Y_j^-)) \quad (1)$$

In the above equation, X_i is the activation of the presynaptic (sending) unit, Y_j is the activation of the postsynaptic (receiving) unit. The $+$ and $-$ superscripts refer to plus-state and minus-state activity, respectively. dW_{ij} is the change in weight between the sending and receiving units, and ϵ is the learning rate parameter.

Our algorithm uses changes in the strength of neural inhibition to generate plus and minus patterns to feed into the CHL equation. To memorize a pattern of activity, we start by soft-clamping the target pattern onto the network. Clamp strength was tuned such that, given a normal level of inhibition, all of the target features (and only those features) are active. This pattern serves as the plus state for learning. We then create two distinct kinds of minus patterns, by raising and lowering inhibition, respectively.

- Raising inhibition distorts the target pattern by making it harder for target units to stay on.
- Lowering inhibition distorts the target pattern by making it easier for non-target units to be active.

Next, the learning algorithm updates weights via two separate CHL-based comparisons. First, it applies CHL to the difference in network activity given normal vs. high inhibition. Second, it applies CHL to the difference in network activity given normal vs. low inhibition.

Comparing Normal vs. High Inhibition At a functional level, the normal vs. high inhibition comparison *strengthens weak parts of the target pattern*, by increasing their connectivity with other parts of the target pattern. Raising inhibition acts as a kind of “stress test” on the target pattern: If a target unit is receiving relatively little collateral support from other target units, such that its net input is just above threshold, raising inhibition will trigger a decrease in the activation of that unit. However, if a target unit is receiving strong collateral support, such that its net input is far above threshold, it will be relatively unaffected by this manipulation. The CHL equation (applied to normal vs. high inhibition) strengthens units that turn off when inhibition is raised, by increasing weights from other active units. These weight changes ensure that a target unit that drops out on a given trial will receive more input the next time that cue is presented. If the same pattern is presented repeatedly, eventually the input to that unit will increase to the point where it no longer drops out in the high inhibition condition. At this point, the unit should be well-connected to the rest of the target representation (making it possible for the network to pattern complete that unit) and no further strengthening will occur.

Comparing Normal vs. Low Inhibition The normal vs. low inhibition comparison *punishes competing units*, by reducing their connectivity with target units. As discussed earlier, competing units can be defined as non-target units that (given normal levels of inhibition) receive almost enough net input to come on, but not enough input to be active in the final, settled state of the network. If a non-target unit is located just below threshold, then lowering inhibition will cause that unit to become active. However, if a non-target unit is far below threshold (i.e., it is not receiving strong input), it will be relatively unaffected by this manipulation. The CHL equation (applied to normal vs. low inhibition) weakens units that turn on when inhibition is lowered, by reducing weights from other active units. These weight changes ensure that a unit that competes on one trial will receive less input the next time that cue is presented. If the same cue is presented repeatedly, eventually the input to that unit will diminish to the point where it no longer activates in the low inhibition condition. At this point, the unit is no longer a competitor, so no further punishment occurs.

Implementing the Algorithm Using Inhibitory Oscillations

The fact that the learning algorithm involves changes in the strength of inhibition led us to consider how the algorithm relates to neural theta oscillations (rhythmic changes in local field potential at a frequency of approximately 4 to 8 Hz in humans). Theta oscillations depend critically on changes in the firing of inhibitory interneurons (Buzsaki, 2002; Toth, Freund, & Miles, 1997), and there are several data points indicating that theta oscillations might play a role in learning (e.g., Seager, Johnson, Chabot, Asaka, & Berry, 2002; Huerta & Lisman, 1996). In the *Discussion* section at the end of the paper, we assess the correspondence between our algorithm and theta in more detail. At this point, the key insights are:

- Continuous inhibitory oscillations are widespread in the brain.
- These oscillations might serve as a neural substrate for our learning algorithm.

The version of the learning algorithm described in the previous section (where inhibition is set to “normal”, “higher-than-normal”, or “lower-than-normal”) is useful for expository purposes, but the discrete nature of the inhibitory states conflicts with the continuous nature of theta oscillations. To remedy this, we devised an implementation of the learning algorithm that oscillates inhibition in a continuous sinusoidal fashion (from “higher than normal” to “lower than normal”). Also, instead of

changing weights by comparing normal vs. high inhibition and normal vs. low inhibition, we change weights by comparing network activity on successive time steps. With regard to the Contrastive Hebbian Learning algorithm, the key intuition is that — at each point in the inhibitory oscillation — the network is either moving *toward* the target state (i.e., the pattern of network activity when inhibition is at its normal level) or it is moving *away* from its target state, toward a less desirable state where there is either too little activity (in the case of high inhibition) or too much activity (in the case of low inhibition). Consider the pattern of activity at time t and the pattern of activity at time $t + 1$. If inhibition is moving toward its normal level, then the activity pattern at time $t + 1$ will be more correct than the activity pattern at time t . In this situation, we will use the CHL equation to adapt weights to make the pattern of activity at time t more like the pattern at time $t + 1$. However, if inhibition is moving away from its normal level, then the activity pattern at time $t + 1$ will be less correct than the activity pattern at time t (it will either contain too much or too little activity, relative to the target pattern). In this situation, we will use the CHL equation to adapt weights to make the pattern of activity at time $t + 1$ more like the pattern at time t . These rules are formalized in Equations 2 and 3.

If inhibition is returning to its normal value:

$$dW_{ij} = \text{rate}((X_i(t+1)Y_j(t+1)) - (X_i(t)Y_j(t))) \quad (2)$$

If inhibition is moving away from its normal value:

$$dW_{ij} = \text{rate}((X_i(t)Y_j(t)) - (X_i(t+1)Y_j(t+1))) \quad (3)$$

Note that the two equations are identical, except for a change in sign. These equations, collectively, serve the same functions as normal vs. high inhibition and normal vs. low inhibition comparisons described earlier: Competitors are punished when the network moves between normal and low inhibition and back again; and weak parts of the target representation are strengthened when the network moves between normal and high inhibition and back again. However, instead of changing weights by comparing snapshots taken at disparate points in time, Equations 2 and 3 achieve the same goal by comparing temporally adjacent network states. Figure 1 summarizes the learning algorithm.

Network Architecture and Biological Relevance

Although we think the oscillating learning algorithm may be applicable to multiple brain structures, the work described here focuses on applying the algorithm to

a neocortical network architecture. McClelland, McNaughton, and O'Reilly (1995) and many others (e.g., Hinton & Ghahramani, 1997; Grossberg, 1999) have argued that the goal of neocortical processing is to gradually develop an internal model of the structure of the environment that allows the network to generate predictions about unseen input features. According to the Complementary Learning Systems model developed by McClelland et al. (1995), one of the defining features of cortical learning is that cortex assigns similar representations to similar inputs. This property allows the network to generalize to new patterns, based on their similarity to previously encountered patterns. McClelland et al. (1995) contrast this with hippocampal learning, which (according to their model) involves assigning distinct representations to input patterns, regardless of their similarity; this property allows hippocampus to do one-trial memorization but hurts its ability to generalize to new patterns based on similarity (see also Marr, 1971; McNaughton & Morris, 1987; Rolls, 1989; Hasselmo, 1995; Norman & O'Reilly, 2003).

To implement a model of cortical learning, our initial simulations used a simple two-layer network, comprised of an *input/output layer* and a *hidden layer*. The network is shown in Figure 2. The input/output layer was used to present patterns to the network. The hidden layer was allowed to self-organize. Every input/output unit was connected to every input/output unit (including itself) and to every hidden unit. All of the synaptic connections were bidirectional and modifiable according to the dictates of the learning algorithm.

This architecture is capable of completing missing pieces of input patterns, via input-layer recurrences and backprojections from the hidden layer. More importantly, the hidden layer gives it the ability to adaptively re-represent the inputs, to facilitate this process of pattern completion. An important goal of the simulations below is to assess whether the oscillating algorithm, applied to this simple cortical architecture, can simultaneously meet the following desiderata for cortical learning (McClelland et al., 1995; O'Reilly & Norman, 2002):

- The network should assign similar hidden-layer representations to similar inputs.
- After repeated exposure to a set of patterns, the network should be able to fill in missing pieces of those patterns, even if the patterns are highly correlated (insofar as real-world input patterns show a high degree of correlation; Simoncelli & Olshausen, 2001).
- The network should be able to generalize to input patterns that resemble (but do not exactly match) trained patterns.

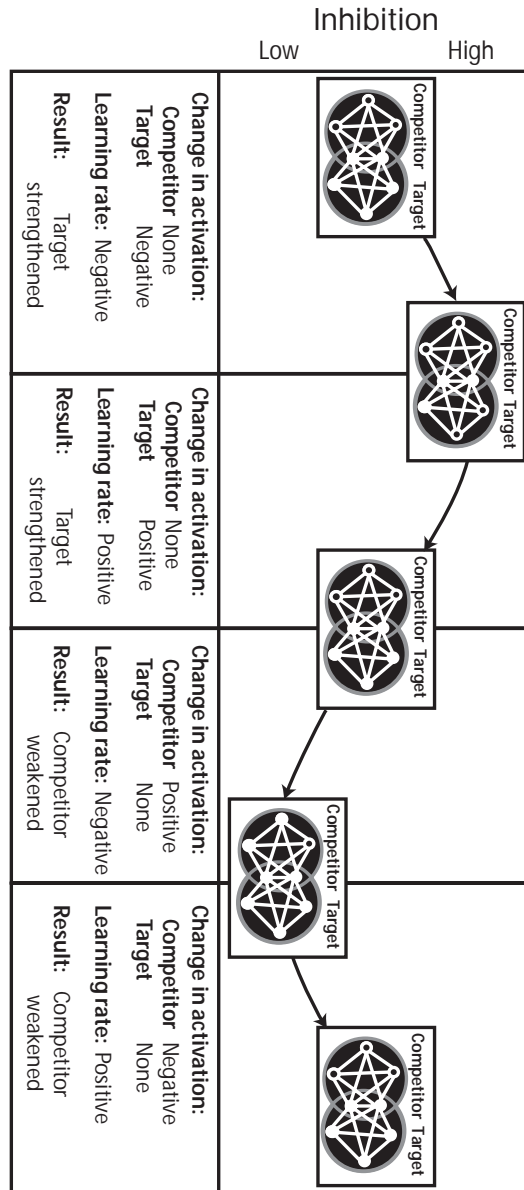


Figure 1: Summary of the combined learning algorithm, showing how target and competitor activity change during different phases of the inhibitory oscillation, and how these changes in activity affect learning. Moving from normal to high back to normal inhibition serves to identify and strengthen weak parts of the target pattern. Moving from normal to low back to normal inhibition serves to identify and punish competitors.

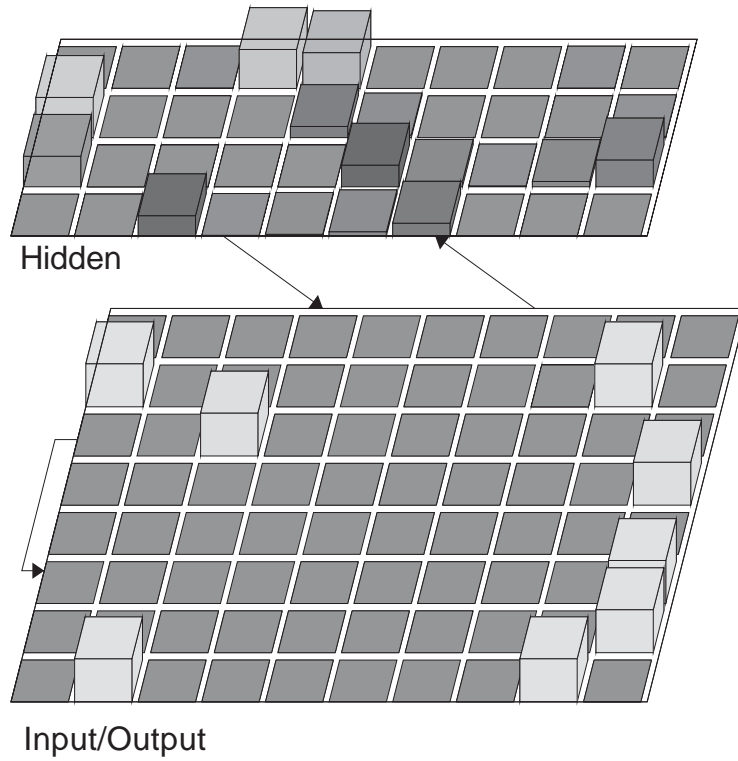


Figure 2: Diagram of the network used in our simulations. Patterns were presented to the lower part of the network (the *input/output* layer). The upper part of the network (the *hidden* layer) was allowed to self-organize. Every unit in the input/output layer was connected to every input/output unit (including itself) and to every hidden unit via modifiable, bidirectional weights. All of the simulations described in the paper used an 80 unit input/output layer. The hidden layer contained 40 units, except when specifically noted otherwise.

General Simulation Methods

The simulation was implemented using a modified version of O’Reilly’s Leabra algorithm (O’Reilly & Munakata, 2000). Apart from a small number of changes listed below (most importantly, relating to the weight update algorithm, and how we added an oscillating component to inhibition) all other aspects of the algorithm used here were identical to Leabra. For a more detailed description of the Leabra algorithm, see O’Reilly and Munakata (2000).

As per the Leabra algorithm, we only explicitly simulated excitatory units and excitatory connections between these units; we did not explicitly simulate inhibitory interneurons. Excitatory activity was controlled by means of a k -winner-take-all (k WTA) inhibitory mechanism (O’Reilly & Munakata, 2000; Minai & Levy, 1994). The k WTA algorithm sets the amount of inhibition for each layer to a value such that at most k units in that layer show activation values $> .25$ (fewer than k units will be active if excitatory input does not exceed the leak current, which exerts a constant drag on unit activation). According to this algorithm, all of the units in a layer receive the same amount of inhibitory input on a given time step, but the amount of inhibition can vary across layers. The k WTA algorithm can be viewed as a shortcut that captures the “set point” role of inhibitory interneurons while reducing computational overhead (relative to explicitly simulating the neurons). The k WTA algorithm also makes it easy to specify the desired amount of activity in a layer by changing the k model parameter. The k parameter was set to $k = 8$ in both the input/output and hidden layers, except when specified otherwise.

To implement the inhibitory oscillation required for the learning algorithm, we used the following procedure: First, at each time step, we used the k WTA algorithm to compute a baseline (normal) level of inhibition. Then, we added an oscillating component to the baseline inhibition value. The oscillating component was only added to the input/output layer, not the hidden layer. We limited the oscillation to the input/output layer because we wanted to build the simplest possible architecture that exhibits the desired learning dynamic. We found that adding oscillations to the hidden layer increases the complexity of the model’s behavior, but it does not substantially affect learning performance in either a positive or negative fashion (see *Appendix C* for a concrete demonstration of this point). The magnitude of the oscillating component was varied in a sinusoidal fashion from min to max (where min and max are negative and positive numbers, respectively).¹

¹We chose values for min and max according to the following criteria: min has to be low enough to allow competitors to activate during the low inhibition phase, but not so low that the network becomes

At the start of each training trial, the target pattern was soft-clamped onto the input/output layer. Over the course of a given trial, inhibition was oscillated once from its normal value to the high inhibition value, then back to normal, then down to the low inhibition value, then back to normal. The onset of the inhibitory oscillation was delayed 20 time steps from the onset of the stimulus. This delay ensures that activity will reach its equilibrium state (corresponding to the retrieved memory) prior to the start of the oscillation. The period of the inhibitory oscillation was set to 80 time steps. This number was chosen because it gives the network enough time for changes in inhibition to lead to changes in activation, but no more time than was necessary. In principle, we could oscillate inhibition multiple times per stimulus. However — given the way that we calculated weight updates (see below) — the effects of multiple inhibitory cycles could be simulated perfectly by staying with one oscillation per stimulus and increasing the learning rate. For a summary of key model parameters relating to the inhibitory oscillation (and other aspects of the model as well), see *Appendix A*.

At each time step (starting at the beginning of the inhibitory oscillation) weight updates were calculated using Equations 2 and 3. However, these weight updates were not applied until the end of the trial. This policy makes it easier to analyze network behavior, because weight changes can not feed back and influence patterns of activation within a trial.

Simulations

In the following simulations, we explore the oscillating algorithm’s ability to meet the desiderata outlined in the *Network Architecture and Biological Relevance* section above. In particular, we are interested in the algorithm’s ability to support *omnidirectional pattern completion*: i.e., its ability to recall any piece of a pattern, when given the rest of the pattern as a cue. The use of the term “omnidirectional” sets this kind of pattern completion apart from asymmetric forms of pattern completion where, e.g., the first half of the pattern can cue recall of the second half, but not vice versa.

To illustrate the strengths and weaknesses of the oscillating algorithm, we compare it to O’Reilly’s Leabra algorithm (O’Reilly, 1996; O’Reilly & Munakata, 2000). Leabra consists of two parts. The core of Leabra is a CHL-based error-driven learning rule, which we will refer to as *Leabra-Error*. In contrast to the oscillating algorithm, which uses changes in the strength of inhibition

epileptic; max has to be high enough such that poorly-supported target units turn off during the high inhibition phase, but not so high that well-supported target units turn off also.

to generate patterns for CHL, the Leabra-Error algorithm learns by comparing the following two phases:

- a *minus* phase, where some features of the to-be-learned pattern are omitted, and the network has to fill in missing features
- a *plus* phase, where the entire to-be-learned pattern is clamped on

The level of inhibition is kept constant across both phases. By comparing minus and plus patterns using CHL, the network learns to minimize the discrepancy between its “guess” about missing features, and the actual pattern. The full version of Leabra complements Leabra-Error with a simple Hebbian learning rule that (during the plus phase) strengthens weights between sending and receiving units when they are both active, and weakens weights when the receiving unit is active but the sending unit is not. This Hebbian rule was developed by Grossberg (1976), who called it *instar learning*; O’Reilly and Munakata (2000) describe the same algorithm, using the name *CPCA Hebbian Learning*.² Simulations conducted by O’Reilly (see, e.g., O’Reilly, 2001; O’Reilly & Munakata, 2000) have demonstrated that adding small amounts of CPCA Hebbian learning to Leabra-Error boosts the learning performance of Leabra-Error by forcing it to represent meaningful input features in the hidden layer.

As recommended by O’Reilly and Munakata (2000), our Leabra comparison simulations used a small proportion of CPCA Hebbian learning (such that weight changes were 99% driven by Leabra-Error, and 1% by CPCA Hebb).³ Finally, to more directly compare the form of CHL inherent in Leabra-Error to the form of CHL inherent in the oscillating algorithm, we also ran simulations using the Leabra-Error rule on its own (without any CPCA Hebbian learning). Bias weight learning was turned off in the Leabra and Leabra-Error simulations in order to better match the oscillating-algorithm simulations (which did not include bias weight learning).

In graphs of simulation results, error bars indicate the standard error of the mean, computed across simulated participants. When error bars are not visible, this is because they are too small relative to the size of the symbols on the graph (and thus are covered by the symbols).

²It is important to emphasize that CPCA Hebbian Learning and Contrastive Hebbian Learning are completely different algorithms: The latter algorithm operates based on differences between two activation states, whereas the former algorithm operates based on single activity snapshots.

³O’Reilly and Munakata (2000) found that higher proportions of Hebbian learning hurt performance, by causing the network to over-focus on prototypical features and to under-focus on lower-frequency features. Pilot simulation work, not published here, confirms that this was true in our Leabra simulations as well.

Simulation 1: Omnidirectional Pattern Completion as a Function of Input-Pattern Overlap and Test-Pattern Noise

In this simulation, we explore the oscillating algorithm’s ability to memorize both correlated and uncorrelated patterns. When given a large number of correlated input patterns, some self-organizing learning algorithms have a tendency to over-represent shared features and under-represent item-specific features, leading to poor recall of item-specific features (e.g., Norman & O’Reilly, 2003 discuss this problem as it applies to CPCA Hebbian learning). In this section, we show that the oscillating algorithm is not subject to this problem. To the contrary, we show that the oscillating algorithm meets all three of the desiderata outlined in the *Biological Relevance* section above:

- The oscillating algorithm outperforms both Leabra and Leabra-Error at recalling individuating features of highly correlated input patterns, both in terms of asymptotic capacity and in terms of training speed.
- The oscillating algorithm shows good generalization to test cues that do not exactly match stored patterns.
- The oscillating algorithm learns representations that reflect the similarity structure of the input space.

Methods

Input Pattern Creation We gave the network 200 binary input patterns to learn. Each pattern had 8 (out of 80) units active. To generate the patterns, we started with a single prototype pattern, and then distorted the prototype by randomly turning off some number of (prototype) units and turning on an equivalent number of (non-prototypical) units. By varying the number of “flipped bits” we were able to vary the average overlap between input patterns. There were three overlap conditions: 57% average overlap (achieved by flipping 2 bits), 28% average overlap (achieved by flipping 4 bits), and 11% average overlap (achieved by flipping all 8 bits). We call the latter condition the *unrelated pattern* condition because the patterns do not possess any central tendency. In creating the patterns (for all of the levels of bit-flipping mentioned above), we implemented a minimum pairwise distance constraint, such that every input pattern differed from every other input pattern by at least two (out of eight) active bits.

Training and Testing All three algorithms were repeatedly presented with the 200-pattern training set until learning reached asymptote. After each epoch of training, we tested pattern completion by measuring the network’s ability to recall a single, non-prototypical feature

from each pattern, given all of the other features of that pattern as a retrieval cue. In the simulations reported here, recall was marked as correct if the activation of the correct unit was larger than the activation of all of the other (non-cued) input/output units.⁴

To assess the model's ability to generalize to test cues that do not exactly match studied patterns, we distorted retrieval cues by adding Gaussian noise to the pattern that was clamped onto the network. Specifically: Each unit's external input value was adjusted by a value sampled from a zero-mean Gaussian distribution. These input values, once adjusted by noise, remained fixed throughout the trial. We manipulated the amount of noise at test by adjusting the variance of the noise distribution.

Applying Leabra to Omnidirectional Pattern Completion For our Leabra and Leabra-Error simulations, we constructed minus phase patterns by randomly blanking out 4 of the 8 units in the input pattern, thereby forcing the network to guess the correct values of these patterns. In the plus phase, we clamped the full 8-unit pattern onto the input layer. Every time that a pattern was presented, a different (randomly selected) set of 4 units was blanked. Otherwise, if the same 4 units were blanked each time, the learning algorithm would learn to recall those 4 units but not any of the other units.⁵

Learning Rates Based on pilot simulations, we selected .0005 as our default learning rate for Leabra and Leabra-Error. Simulations using this learning rate yielded asymptotic capacity that was almost identical to the capacity achieved with lower learning rates, and training time was within acceptable bounds. For the oscillating algorithm, we were able to achieve near-peak performance with much higher learning rates. We found that a learning rate of .05 for the oscillating-algorithm yielded the best combination of high final capacity and (relatively) short training time. The number of training epochs for each algorithm/learning-rate combination was adjusted to ensure that training lasted long enough to reach asymptote. Large differences in learning rates were mirrored by commensurately large differences in training duration (e.g., the oscillating-algorithm simulations with learning rate .05 took 250 epochs to reach asymptote;

⁴We have also run pattern-completion simulations using a more restrictive recall criterion, whereby recall was marked "correct" if the activation of the correct unit was $> .5$, and none of the incorrect units had activation $> .5$. All of the advantages of the oscillating algorithm (relative to other algorithms) that are shown in *Simulation 1* and *Simulation 2* (below) are also present when we use this more restrictive recall criterion.

⁵This is not the only way to train a Leabra network so it supports pattern completion. However, it is the most effective method that we were able to find. Any conclusions that we reach about Leabra and Leabra-Error are restricted to the particular variants that we used in our simulations, and may not apply to simulations where other methods are used to generate partial patterns for the minus phase.

in contrast, Leabra simulations with learning rate .0005 took 10,000 epochs to reach asymptote).

Results

Capacity Figure 3 shows the asymptotic number of patterns learned for the oscillating algorithm, Leabra, and Leabra-Error. For unrelated input patterns and low levels of test-pattern noise, the oscillating algorithm learns approximately 150 out of 200 patterns, but it does less well than both Leabra and Leabra-Error. However, for higher levels of test-pattern noise and higher levels of input-pattern overlap, the relative position of the algorithms reverses, and the oscillating algorithm performs substantially better than Leabra and Leabra-Error. *Appendix C* shows that the oscillating algorithm's advantage for highly-overlapping inputs is still obtained when inhibition is oscillated in the hidden layer (in addition to the input/output layer).

Hidden Representations The oscillating algorithm's superior performance for high levels of input pattern overlap and test pattern noise stems from its ability to maintain reasonable levels of pattern separation on the hidden layer, even when inputs are very similar. Figure 4 plots the average pairwise overlap between patterns in the hidden layer (at the end of training), as a function of input overlap.⁶ The figure shows that all three algorithms maintain good pattern separation in the hidden layer given low input overlap, but as input overlap increases, hidden overlap increases much more sharply in the Leabra and Leabra-Error simulations vs. in the simulations using the oscillating algorithm. The high level of hidden-layer overlap in the Leabra and Leabra-Error simulations facilitates recall of shared features but makes it difficult for the network to recall the unique features of individual patterns. This problem is especially severe given high levels of test-pattern noise: When hidden representations are located close together, this increases the odds that (given a noisy input pattern) the network will slip out of the correct attractor into a neighboring attractor.

The oscillating algorithm's good pattern separation in the high-overlap condition is due, in large part, to its ability to punish competitors. If the representations of two patterns (call them pattern A and pattern B) get too close to each other, then pattern A will start appearing as a competitor (during the low inhibition phase) during study of pattern B, and vice versa. Assuming that A and B are both presented a large number of times at training, the ensuing competitor-punishment will have the effect of pushing apart the hidden-layer representations of A and B so they no longer compete with one another.

⁶Both input overlap and hidden overlap were operationalized using a cosine distance measure; this measure ranges from zero (no overlap) to one (maximal overlap).

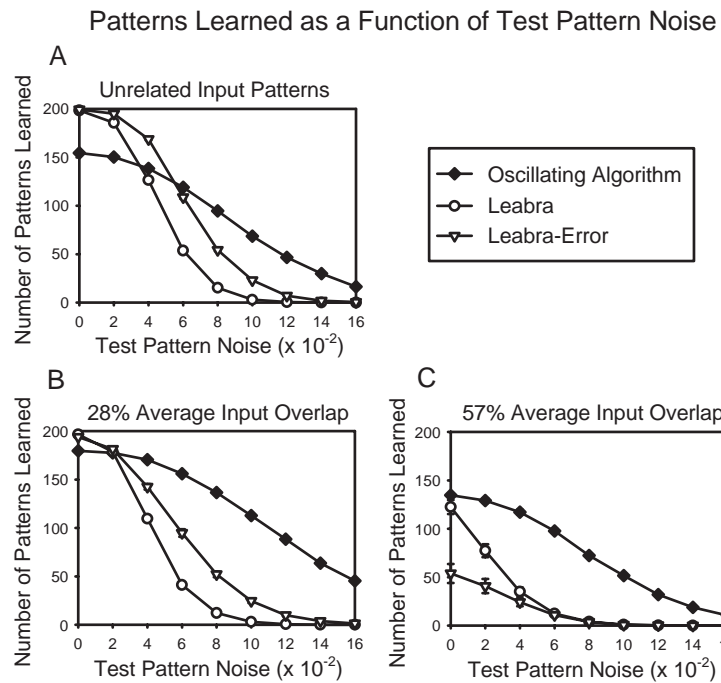


Figure 3: This figure shows the number of patterns (out of 200) successfully recalled at the end of training by each algorithm, as a function of the amount of overlap between input patterns (part A: unrelated patterns; part B: correlated patterns, 28% overlap; part C: correlated patterns, 57% overlap) and the amount of noise applied to retrieval cues at test. Leabra and Leabra-Error outperform the oscillating algorithm given low input-pattern overlap and low levels of test-pattern noise. However, for higher levels of input-pattern overlap and test-pattern noise, the oscillating algorithm outperforms Leabra and Leabra-Error.

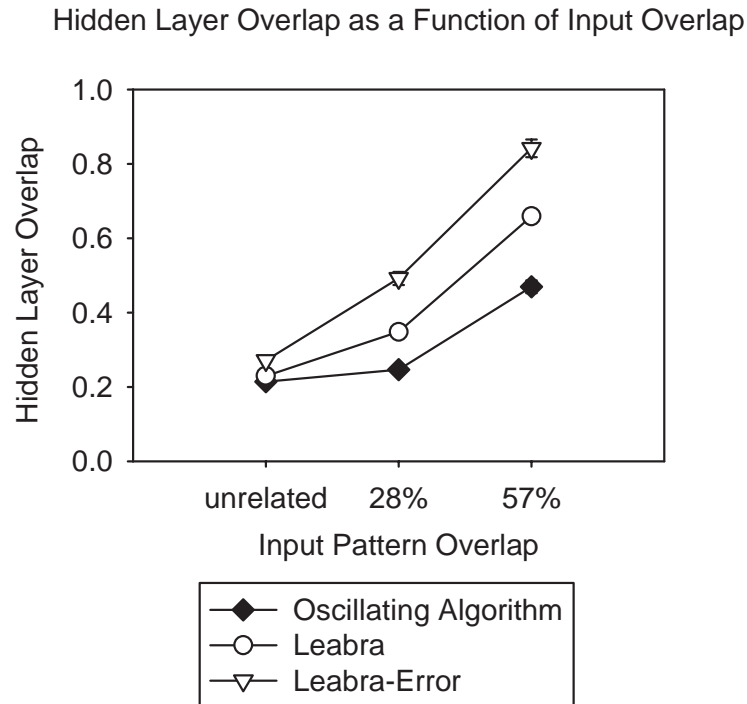


Figure 4: This figure plots, for the oscillating algorithm, Leabra, and Leabra-Error, the average pairwise overlap between patterns in the hidden layer (at the end of training), as a function of input-pattern overlap. Hidden-layer overlap is lower for the oscillating algorithm than for Leabra and Leabra-Error.

Another factor that contributes to the oscillating algorithm's good performance is its ability to focus learning on features that are not already well-learned. Given a large number of correlated patterns, the oscillating algorithm stops learning about prototypical features relatively early in training (once their representation is strong enough to resist increased inhibition), and focuses instead on learning idiosyncratic features of individual items (which are less able to resist increased inhibition). Reducing learning of prototypical features, relative to item-specific features, improves pattern separation and (through this) pattern completion performance.

While the oscillating algorithm shows more pattern separation than Leabra and Leabra-Error, it still possesses the key property that it assigns similar hidden representations to similar stimuli. In this respect, the oscillating algorithm (applied to this two-layer cortical network) differs strongly from hippocampal architectures that automatically assign distinct representations to stimuli (e.g., Norman & O'Reilly, 2003). To quantify the oscillating algorithm's tendency to use similarity-based representations, we computed the correlation (across all pairs of patterns) between input-layer overlap and hidden-layer overlap. Figure 5 plots this *input-hidden similarity score* for the oscillating algorithm, Leabra, and

Leabra-Error, as a function of input-pattern overlap. The average similarity score for the oscillating algorithm is approximately .5. For the values of input pattern overlap plotted here, the similarity scores for the oscillating algorithm are higher than the scores for Leabra-Error, but lower than the scores for Leabra.

The observed difference between Leabra and the oscillating algorithm can be viewed in terms of a simple tradeoff: Leabra learns representations that are true to the structure of the input space, but (given similar input patterns) this fidelity leads to high hidden-layer overlap that hurts recall. The oscillating algorithm gives up a small amount of this fidelity, but — as a result of this sacrifice — it is much better able to recall given high levels of input-pattern overlap and noisy test cues. Furthermore, we should emphasize that the tradeoff observed here is a direct consequence of the limited size of the hidden layer: When hidden layer size is increased, the oscillating algorithm is able to utilize the extra hidden units to simultaneously boost similarity scores and capacity; we demonstrate this point in *Appendix B*.

Training Speed Finally, in addition to measuring capacity, we can also evaluate how quickly the various algorithms reach their asymptotic capacity. Across all of the conditions described above, the oscillating al-

Input-Hidden Similarity Score as a Function of Input Overlap

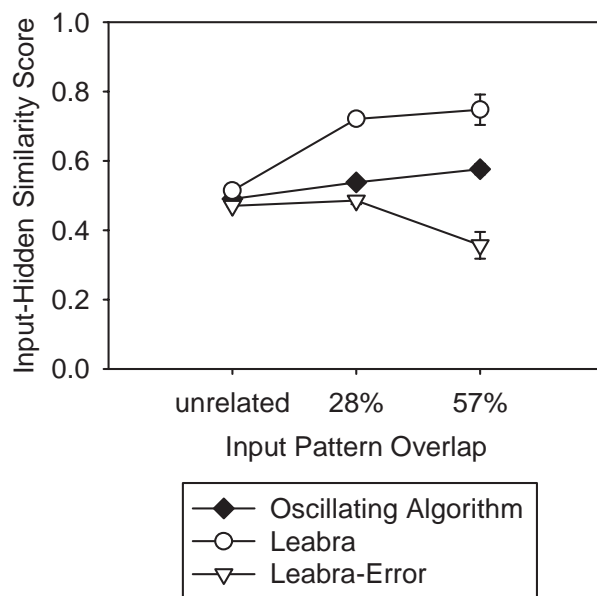


Figure 5: This figure plots, for the oscillating algorithm, Leabra, and Leabra-Error, the models’ tendency to assign similar hidden representations to similar input patterns. See text for more details on how this similarity score was computed. Similarity scores for the oscillating algorithm are higher than similarity scores associated with Leabra-Error, and lower than similarity scores associated with Leabra.

gorithm learns more quickly than Leabra and Leabra-Error. To illustrate this point, we selected two conditions where asymptotic capacity was approximately matched between the oscillating algorithm and either Leabra or Leabra-Error: Specifically, we compared the oscillating algorithm and Leabra-Error for unrelated input patterns with .06 test-pattern noise; we also compared the oscillating algorithm and Leabra for input patterns with 57% overlap and zero test-pattern noise. For each of these conditions, we plotted the time course of learning across epochs, for a variety of Leabra and Leabra-Error learning rate values, in Figure 6.

In the figure, the oscillating-algorithm training curves lie to the left of the Leabra and Leabra-Error training curves, across the full range of Leabra and Leabra-Error learning rates that we tested (ranging from .0005 to .03). While it is possible to increase the initial speed of learning in Leabra and Leabra-Error by increasing the learning rate parameter, this also has the effect of lowering the asymptotic capacity of the Leabra and Leabra-Error networks to below the asymptotic capacity of the oscillating algorithm.

The Leabra and Leabra-Error variants used here learn more slowly than the oscillating algorithm because they only learn about a subset of intra-item associations on

each trial. For example, if the top 4 units are blanked during the minus phase in Leabra-Error, the network will learn how to complete from the bottom 4 units to the top 4 units, but it will not learn how to complete from the top 4 to the bottom 4. Thus, it takes multiple passes through the study set (blanking a different set of units each time) for Leabra-Error to strengthen all of the different connections that are required to support omnidirectional pattern completion. In contrast, the oscillating algorithm has the ability to learn about the whole pattern on each trial.

Simulation 2: 3-Layer Autoencoder

In this simulation, we set out to replicate and extend the results of *Simulation 1* using a different network architecture: the *3-layer autoencoder* (Ackley et al., 1985). These networks consist of an input layer, which is connected to a hidden layer, which in turn is connected to an output layer. During training, the to-be-learned pattern is presented (in its entirety) to the input layer, and activity is allowed to propagate through the network. The goal of autoencoder learning is to adjust network weights so the network is able to reconstruct a copy of the input pattern on the output layer. The main difference between the autoencoder architecture and the 2-layer architecture used in *Simulation 1* is that, in the autoencoder architec-

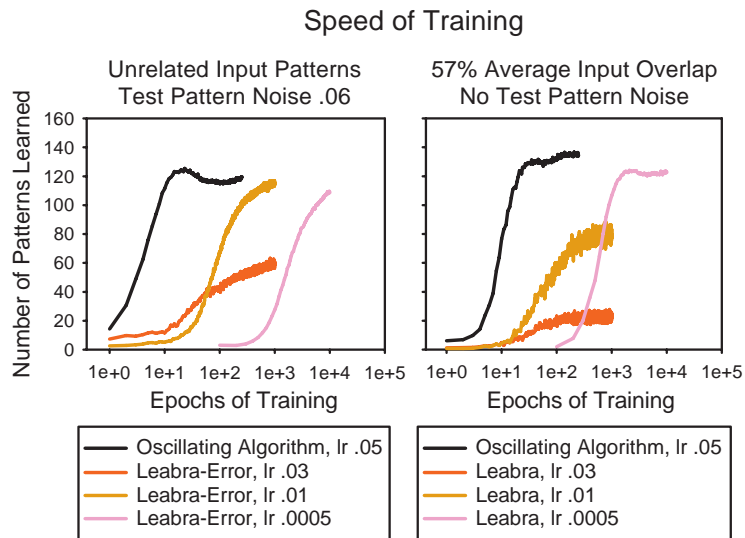


Figure 6: Training speed for the oscillating algorithm, Leabra, and Leabra-Error. The left-hand figure plots the time course of training for the oscillating algorithm and Leabra-Error for unrelated input patterns and .06 test-pattern noise; the right-hand figure plots the time course of training for the oscillating algorithm and Leabra for 57% input overlap and zero test-pattern noise. In both figures, the oscillating algorithm training curves are located to the left of the Leabra and Leabra-Error training curves, across a wide variety of Leabra and Leabra-Error learning rate values. Error bars were omitted from the graph for visual clarity; for all of the points shown here, the standard error was less than 3.5.

ture, there are no direct connections between the input units that receive external input (from the retrieval cue) at test, and the to-be-retrieved output unit — everything has to go through the hidden layer. Thus, the autoencoder architecture constitutes a more stringent test of whether a learning algorithm can develop information-preserving hidden representations that support pattern completion.

We compared the oscillating algorithm’s ability to learn patterns in the autoencoder architecture to Leabra and Leabra-Error. Also, the way the network was structured (with distinct input and output layers) made it possible for us to explore two additional comparison algorithms: Almeida-Pineda recurrent backpropagation, and standard (non-recurrent) backpropagation.

The results of this simulation replicate the key finding from *Simulation 1*: The oscillating algorithm outperforms the comparison algorithms at omnidirectional pattern completion when both input-pattern overlap and test-pattern noise are high. However, unlike in *Simulation 1*, the oscillating algorithm also outperforms the comparison algorithms in tests with unrelated input patterns and low levels of test-pattern noise. We attribute this latter finding to the fact that the oscillating algorithm automatically strengthens weak connections between target units. In contrast, error-driven algorithms like backpropagation and Leabra-Error only strengthen connections between target units if this is needed to reduce error

at training.

Autoencoder methods

Network Architecture To implement an autoencoder architecture, we added an output layer to the network, so the network was comprised of an 80-unit input layer, a 40-unit hidden layer, and an 80-unit output layer. The input layer had a full, bidirectional projection to the hidden layer, and the hidden layer had a full, bidirectional projection to the output layer. To maximize comparability with our initial simulations, every input unit was directly connected to every other input unit, and every output unit was directly connected to every other output unit (the one crucial difference was that, in the autoencoder simulations, the input units were not directly connected to the output units). We used the same connection parameters as in *Simulation 1*; all connections were modifiable. The one exception to the scheme outlined here was the backpropagation autoencoder, which only had feedforward connections (from the input layer to the hidden layer, and from the hidden layer to the output layer).

Training and Testing All of the algorithms were given 150 patterns to memorize. The training set was repeatedly presented (in a different order each time) until learning reached asymptote. The details of training for each algorithm are presented below. After each training

epoch, we tested pattern completion: On each test trial, we left out one feature from the input pattern, and measured how well the network was able to recall the missing feature on the output layer. As with all of the simulations described earlier, we only tested recall of item-specific features (i.e., features that were not part of the prototype pattern), and we scored recall as being correct based on whether the to-be-recalled output unit was more active than all of the other (non-cued) units in the output layer. Finally, as in *Simulation 1*, we manipulated the level of input pattern overlap, and we also explored how distorting test cues (by adding Gaussian noise to the test patterns) affected pattern completion performance.⁷

Methods for Oscillating-Algorithm Simulations We trained the oscillating-algorithm version of the autoencoder by simultaneously presenting the same pattern to both the input and output layers. During training, inhibition was oscillated on the input and output layers, but not on the hidden layer. The input-layer and output-layer oscillation parameters were identical to each other, and identical to the parameters that we used in *Simulation 1*.

Methods for Leabra and Leabra-Error Simulations The Leabra and Leabra-Error autoencoder simulations used a two-phase design. In the minus phase, the complete target pattern was clamped onto the input layer (but not the output layer) and the network was allowed to settle. In the plus phase, the complete target pattern was clamped onto both the input and output layers and the network was allowed to settle. Otherwise, the details of the Leabra and Leabra-Error simulations were the same as in *Simulation 1*.

Methods for Recurrent and Non-Recurrent Backpropagation Simulations For our simulations using the Almeida-Pineda (A-P) recurrent backpropagation algorithm (Almeida, 1989; Pineda, 1987), we used the rbp++ program contained in the PDP++ neural network software package.⁸ For our simulations using the (non-recurrent) backpropagation algorithm (Rumelhart, Hinton, & Williams, 1986), we used the bp++ program contained in the PDP++ software package. For both sets of simulations (rbp++ and bp++), we used the default learning parameters built in to the software package (e.g., momentum = .9), except we changed the learning rate (as described below), and — as per all of the simulations in

⁷A given amount of test-pattern distortion had less of an effect in *Simulation 2* than in *Simulation 1* because, in *Simulation 2*, we were only distorting the input-layer pattern, whereas in *Simulation 1*, we were applying the distortion to a shared input/output layer (so the distortion had a direct effect on output activity, in addition to an indirect effect via the distorted input pattern). To compensate for this difference, we used a wider range of test-pattern noise values in *Simulation 2* than in *Simulation 1*.

⁸This software can be downloaded from Randy O'Reilly's PDP++ website at the University of Colorado: <http://psych.colorado.edu/~oreilly/PDP++/PDP++.html>

this paper — we turned off bias weight learning.

Learning Rates We used a learning rate of .0005 for all four comparison algorithms. The oscillating-algorithm simulations used a learning rate of .05. We allowed each algorithm to run until learning reached asymptote (10,000-20,000 epochs for Leabra, Leabra-Error, and A-P recurrent backpropagation; 100,000 epochs for feedforward backpropagation; 250 epochs for the oscillating algorithm).⁹

Results of Autoencoder Simulations

The results of our autoencoder capacity simulations are shown in Figure 7. The oscillating algorithm outperforms the comparison algorithms in every condition, with the sole exception of high input overlap, low test pattern noise (where the oscillating algorithm's performance is comparable to backpropagation). As in *Simulation 1*, the oscillating algorithm's advantage over other algorithms is larger for high test-pattern-noise than for low-test-pattern noise. The most notable difference between *Simulation 1* and *Simulation 2* is that — in this simulation — the oscillating algorithm outperforms the comparison algorithms given low input pattern overlap and no test-pattern noise (whereas the opposite is true in *Simulation 1*).

The fact that the oscillating algorithm shows better pattern completion than the four comparison algorithms can be explained as follows:

- Pattern completion performance is a direct function of the strength of links between features within a pattern.
- Standard autoencoder training (as embodied by the four comparison algorithms) does not force the network to learn strong associations between input features.

In order to minimize reconstruction error, autoencoders need to represent all of the input features somewhere in the hidden layer, but they do not have to link these features. In many situations, the autoencoder can minimize reconstruction error by representing features individually, and then reconstructing the input on a feature-by-feature basis. This strategy leads to poor pattern completion performance. In contrast, the oscillating learning algorithm places strong pressure on the network to form direct associations between the features of to-be-memorized patterns (because units need collateral support from other units in order to withstand the "stress test" of increased inhibition). These strong links result in good pattern completion performance.

⁹We also ran backpropagation simulations with larger learning rates and the results were qualitatively similar to the results obtained here

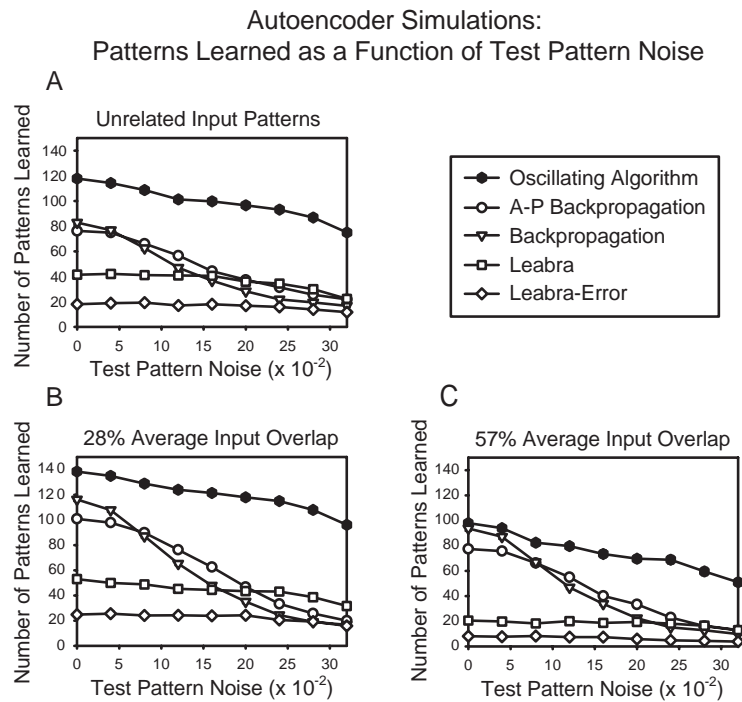


Figure 7: Results of three-layer autoencoder simulations where we manipulated input-pattern overlap and test-pattern noise. The oscillating algorithm performs better than the other algorithms in all conditions except for high input overlap, low test-pattern noise (where the oscillating algorithm’s performance is comparable to backpropagation). In general, the oscillating algorithm’s advantage is accentuated for high levels of test-pattern noise.

Discussion

The research presented here shows how oscillations in the strength of neural inhibition can facilitate learning. Specifically:

- Lowering inhibition can be used to identify competing memories so they can be punished
- Raising inhibition can be used to identify weak parts of memories so they can be strengthened

As mentioned earlier, the specific weight change equation (CHL) that we use in the oscillating algorithm is not novel. Rather, the novel claim is that changes in the strength of inhibition can be used to generate “minus” (i.e., less desirable) patterns to feed into the CHL equation. In the discussion section, we provide a brief overview of the primary virtues of the oscillating learning algorithm, relative to the other algorithms considered here. Then, we discuss how the oscillating algorithm relates to neural data on oscillations and learning, and how the oscillating algorithm relates to the BCM rule (Bienenstock, Cooper, & Munro, 1982).

Functional Properties of the Learning Algorithm

In the *Simulations* section, we showed that the oscillating algorithm (applied to a cortical network architecture) meets the desiderata for cortical learning outlined earlier: good completion of overlapping patterns (after repeated exposure to those patterns); good generalization to retrieval cues that do not exactly match studied patterns; and good correspondence between the structure of the input patterns and the hidden representations (i.e., similar input patterns tend to get assigned similar hidden representations).

We attributed the oscillating algorithm’s good performance for overlapping inputs and noisy test cues to its ability to *punish competing memories*. Whenever memories start to blend together, they start to compete with one another at retrieval, and the competitor-punishment mechanism pushes them apart. In this manner, the oscillating algorithm retains good pattern separation in the hidden layer (see Figure 3) even when inputs overlap strongly. As discussed earlier, this extra separation is not without costs (e.g., it incrementally degrades the hidden layer’s ability to represent the structure of the input space, compared to Leabra), but the costs are small relative to the following benefit: Maintaining good separation between representations helps to ensure that memories can be accurately stored and accessed even in difficult situations (e.g, when there are many similar memo-

ries stored in the system, and the cue only slightly favors one memory over the other).

The fact that the oscillating algorithm outperforms all of the comparison algorithms in *Simulation 2* (pattern completion with an autoencoder architecture), even for unrelated input patterns, points to another key property of the algorithm: It *automatically probes* for weak parts of the attractor (by raising inhibition) and strengthens these weak parts. This automatic probing and strengthening ensures that the network will be able to pattern-complete from one arbitrary sub-part of the pattern to another, regardless of whether that particular partial pattern has been encountered before. In contrast, the other algorithms that we examined (e.g., Leabra-Error) show a large performance hit when the partial patterns used to cue retrieval at test do not exactly match the patterns used to cue retrieval (during the minus phase) at training.

Relating the Oscillating Algorithm to Neural Theta Oscillations

As mentioned earlier, we think that neural theta oscillations (and theta-dependent learning processes) may serve as the neural substrate of the oscillating learning algorithm. Theta oscillations have been observed in humans in both neocortex and the hippocampus. Raghavachari, Kahana, Rizzuto, Caplan, Kirschen, Bourgeois, Madsen, and Lisman (2001) found that cortical theta was gated by stimulus presentation during a memory experiment, and Rizzuto, Madsen, Bromfield, Schulze-Bonhage, Seelig, Aschenbrenner-Scheibe, and Kahana (2003) found that theta phase is reset by stimulus onset. These findings both indicate that theta oscillations are present at the right time to support stimulus memorization. Other findings point to a more direct link between theta and synaptic plasticity. In a recent study, Seager et al. (2002) found that eyeblink conditioning occurred more quickly when animals were trained during periods of high vs. low hippocampal theta power (see Berry & Seager, 2001 for a review of similar studies). Also, Huerta and Lisman (1996) induced theta oscillations in a hippocampal slice preparation and found that the direction of plasticity (LTP vs. LTD) depends on the phase of theta (see also Holscher, Anwyl, & Rowan, 1997 for a similar result in anesthetized animals, and Hyman, Wyble, Goyal, Rossi, & Hasselmo, 2003 for a similar result in behaving animals).

The finding that LTP is obtained during one phase of theta and LTD is obtained during another phase fits very well with the oscillating algorithm’s postulate that one part of the inhibitory oscillation (going from normal to high to normal inhibition) is primarily concerned with strengthening target memories, and the other part of the oscillation (going from normal to low to normal inhi-

bition) is primarily concerned with weakening competitors. Although this result is very encouraging, more work is needed to explore the mapping between the LTP/LTD findings cited above, and our model. The mapping is not straightforward, because the aforementioned studies used local field potential to index theta, and it is unclear how much local field potential is driven by excitatory vs. inhibitory neurons.

One could reasonably ask why we think the oscillation in our algorithm relates to theta oscillations as opposed, e.g., to alpha or gamma oscillations. Functionally, the frequency of the oscillation in our algorithm is bounded by two constraints: The oscillation has to be fast enough such that the oscillation completes least one full cycle (and ideally more) when a stimulus is presented. This rules out slow oscillations (< 1 Hz). Also, if the oscillation is too fast relative to the speed of spreading activation in the network, then competitors will not have a chance to activate during the low inhibition phase. This constraint rules out very fast oscillations. Thus, although we are not certain that theta is the correct frequency, the functional constraints outlined here and the findings relating theta to learning (outlined above) make this a possibility worth pursuing.

Applications to Hippocampal Architectures

Although this paper has focused on cortical network architectures, we also think that our ideas about theta (i.e., that theta can help to selectively strengthen weak target units and punish competitors) may be applicable to hippocampal architectures. At this time, there are several theories (other than ours) regarding how theta oscillations might contribute to hippocampal processing. For example, Hasselmo, Bodelon, and Wyble (2002) argue that theta oscillations help tune hippocampal dynamics for encoding vs. retrieval, such that dynamics are optimized for encoding during one phase of theta, and dynamics are optimized for retrieval during another phase of theta. Hasselmo's model varies the relative strengths of different excitatory projections as a function of theta (to foster encoding vs. retrieval), but does not vary inhibition. In contrast, our model varies the strength of inhibition, but does not vary the strength of excitatory inputs. At this point in time, it is unclear how our model relates to Hasselmo's model. We do not see any direct contradictions between our model and Hasselmo's model (insofar as they manipulate different model parameters as a function of theta), so it seems possible that the two models could be merged, but further simulation work is needed to address this question.

Relating the Oscillating Algorithm to BCM

In this section, we briefly review another algorithm (the BCM algorithm: Bienenstock et al., 1982) that can be viewed as implementing competitor punishment. Like the CPCA Hebbian learning rule mentioned earlier, the BCM algorithm is set up to learn about clusters of correlated features. The main difference between BCM and CPCA Hebbian learning relates to the circumstances under which synaptic weakening (LTD) occurs. CPCA Hebbian learning reduces synaptic weights when the receiving unit is active but the sending unit is not. In contrast, BCM reduces synaptic weights from *active sending units* when the receiving unit's activation is above zero but below its average level of activation. Thus, BCM actively pushes away input patterns from weakly activated hidden units. This form of synaptic weakening can be construed as a form of competitor punishment: If a memory receives enough input to activate its hidden representation, but not enough to fully activate that representation, that memory is weakened. In contrast, if a memory does not receive enough input to activate its hidden representation, the memory is not affected. The main functional difference between competitor-punishment in BCM vs. the oscillating algorithm is that BCM can only punish competitors if their representations show above-zero (but below-average) activation. In contrast, the oscillating algorithm actively probes for competitors (by lowering inhibition) and is therefore capable of punishing competitors even if they are not active given normal levels of inhibition. This "active probing" mechanism should result in much more robust competitor punishment. Importantly, BCM's form of competitor punishment and the oscillating algorithm's form of competitor punishment are not mutually exclusive: It is possible that combining the algorithms would result in better performance than either algorithm taken in isolation. We will explore ways of integrating BCM with the oscillating learning algorithm in future research.

Applying the Oscillating Algorithm to Psychological Data

In this paper, we have focused on functional properties of the learning algorithm (e.g., its capacity for learning patterns, given different levels of overlap). Another way to constrain the model is to explore its ability to simulate detailed patterns of psychological data. In one line of research, we have used the model to account for several key findings relating to Retrieval-Induced Forgetting (Anderson, 2003; see the *Data Indicating Competitor Punishment* section at the beginning of the paper for more discussion of this phenomenon). For example, the model can explain the finding that forgetting of competing items is *cue-independent* (Anderson & Spell-

man, 1995), the finding that competitor-punishment effects are larger when subjects are asked to retrieve the target vs. when they are just shown the target (Anderson, Bjork, & Bjork, 2000), and the finding that the amount of competitor-punishment is proportional to the strength of the competitor (Anderson et al., 1994). This modeling work is described in detail in Norman, Newman, and Detre (submitted).

Conclusions

The research presented here started with a psychological puzzle: How can we account for data showing that competitors are punished? In the course of addressing this issue, we found that competitor-punishment mechanisms can boost networks’ ability to learn highly overlapping patterns (by ensuring that hidden representations do not collapse together). We also observed that the changing-inhibition aspect of our algorithm bears a strong resemblance to neural theta oscillations. As such, this research may end up speaking to the longstanding puzzle of how theta oscillations contribute to learning. The challenge now is to follow up the admittedly preliminary results presented here, with a more detailed assessment of how the basic principles of the oscillating algorithm (competitor punishment via decreased inhibition, and selective strengthening via increased inhibition) can shed light on psychological, neural, and functional issues.

Appendix A: Model Parameters

Basic Network Parameters

At the beginning of each simulation, all of the weights were initialized to random values from the uniform distribution centered on .5 with range = .4. The initial weight values were symmetric, such that the initial weight from unit i to unit j was equivalent to the initial weight from unit j to unit i . This symmetry was maintained through learning because the weight update equations are symmetric.

Other model parameters:

Parameter	Value
stm_gain	0.4
input/output layer dt_{vm}	0.2
hidden layer dt_{vm}	0.15
i_kwt_pt	0.325

Apart from the parameters mentioned above, all other

parameters shared by the oscillating learning algorithm and Leabra were set to their Leabra default values.

Oscillation Parameters

The oscillating component of inhibition was varied from $min = -1.21$ to $max = 1.96$. As per Equations 2 and 3, the sign of the learning rate was shifted from positive to negative depending on whether inhibition was moving toward its normal (midpoint) value, or away from its normal value. The network was given 20 time steps to settle into a stable state before the onset of the inhibitory oscillation. Figure 8 shows how inhibition was oscillated on each trial, and how the sign of the learning rate was changed as a function of the phase of the inhibitory oscillation.

Appendix B: Effects of Hidden-Layer Size

In this simulation, we show that the oscillating algorithm can take advantage of additional hidden-layer resources to store more patterns, and to more accurately represent the structure of the input space. Specifically, we explored the effect of increased hidden layer size (120 vs. 40 units) on the oscillating algorithm, Leabra, and Leabra-Error. The hidden-layer k value was adjusted as a function of hidden-layer size to ensure that (on average) 20% of the hidden units would be active for both the 120-hidden-unit simulations and the 40-hidden-unit simulations. Input patterns had 57% average overlap, and we used a test-pattern noise value of .04.

Figure 9 shows the effects of hidden-layer size on capacity and on the fidelity of the network’s representations (as indexed by our “similarity score” measure). There are two important results: First, increasing hidden-layer size boosts the number of patterns learned by the oscillating algorithm. The effect of increasing hidden-layer size is numerically larger for the oscillating algorithm than for Leabra and Leabra-Error, so the capacity advantage for the oscillating algorithm is preserved in the 120-hidden-unit condition. The second important result is that, for all three algorithms, our input-hidden similarity metric (as described in the *Hidden Representations* section above) is substantially larger for the large-network simulations: For the oscillating algorithm, the similarity score is .576 for the 40-unit simulation and .788 for the 120-unit simulation.

Appendix C: Effects of Hidden-Layer Oscillations

For simplicity’s sake, the oscillating-algorithm simulations described in the body of the paper oscillated inhi-

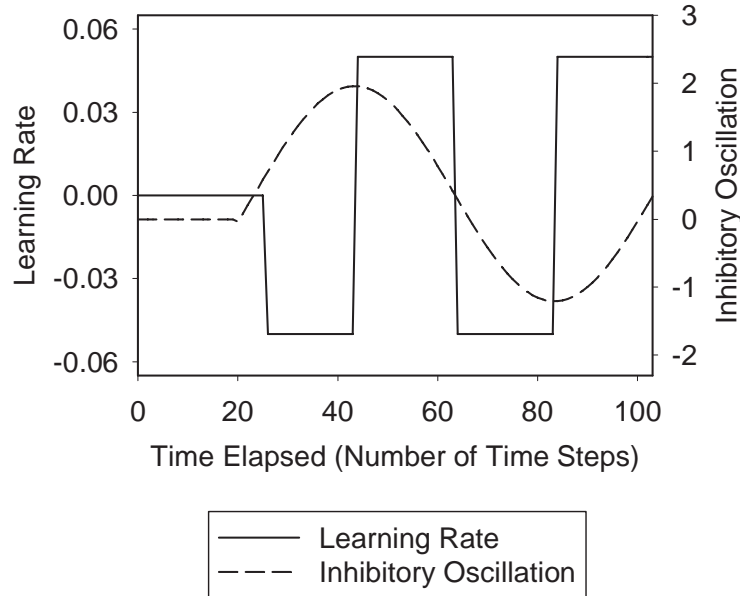


Figure 8: Illustration of how inhibition was oscillated on each trial. At each time step, the “inhibitory oscillation” component depicted on this graph was added to the value of inhibition computed by the kWTA algorithm. The graph also shows how the sign of the learning rate was set to a positive value when the inhibitory oscillation was moving toward its midpoint, and it was set to a negative value when the inhibitory oscillation was moving away from its midpoint.

bition in the input/output layer, but not the hidden layer. In this simulation, we show that the same qualitative pattern of results is obtained when inhibition is simultaneously oscillated in both the input/output layer and in the hidden layer.

We selected hidden-layer oscillation parameters such that, over the course of training, the effect of the inhibitory oscillation on network activity (operationalized as the difference in average network activation from the peak of the inhibitory oscillation to the trough of the oscillation) was approximately equated for the input/output layer and the hidden layer.¹⁰ The simulation used input patterns with 57% overlap.

The results of this simulation are shown in Figure 10: The oscillating algorithm continues to outperform Leabra and Leabra-Error at learning highly overlapping patterns, even with the addition of oscillations in the hidden layer. We did not attempt to fine-tune the

performance of the model once we added hidden oscillations, so the detailed pattern of results obtained here (e.g., the fact that the network performed slightly better without hidden oscillations) should not be viewed as reflecting parameter-independent properties of the model. Rather, these results constitute an existence proof that the oscillating-algorithm advantages that we find in our “default parameter” simulations (without hidden-layer oscillations) can also be observed in simulations with comparably sized hidden-layer and input/output-layer oscillations.

Acknowledgements

This research was supported by NIH grant R01MH069456, awarded to KAN.

¹⁰To equate the average effect of the inhibitory oscillation on activity in the input/output layer vs. the hidden layer, we ended up using a much smaller-sized oscillation in the hidden layer than in the input/output layer: hidden oscillation $min = -0.18$ and $max = 0.10$; for the input oscillation we used our standard $max = 1.96$ and a slight smaller-than-usual $min = -1.11$; we set the learning rate to .03. The input-layer inhibitory oscillation needs to be large in order to offset the strong (excitatory) external input coming into the target units. Hidden units do not receive this strong external input, so less inhibition is required to deactivate these units during the high-inhibition phase.

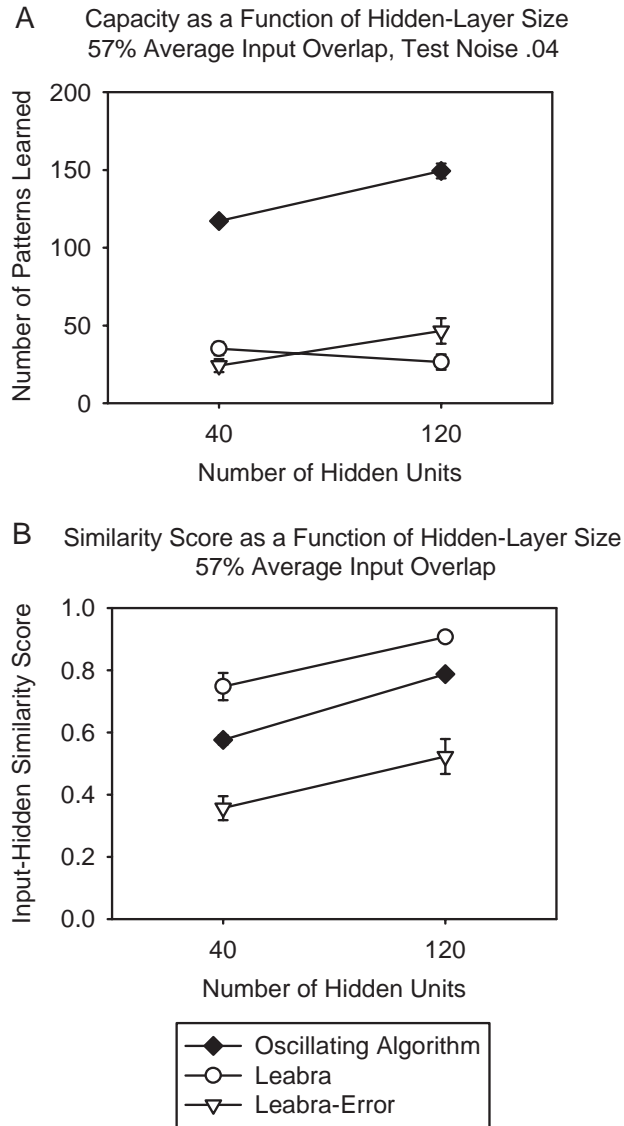


Figure 9: Part A: Capacity scores (number of patterns learned out of 200) for the oscillating algorithm, Leabra, and Leabra-Error, given 57% input pattern overlap and .04 test-pattern noise. Increasing hidden-layer size increases the number of patterns learned by the oscillating algorithm, and the oscillating algorithm continues to perform well relative to Leabra and Leabra-Error. Part B: Input-hidden similarity scores (see *Simulation 1* for how these were calculated) given 57% input pattern overlap. All three algorithms show better similarity scores for the larger network.

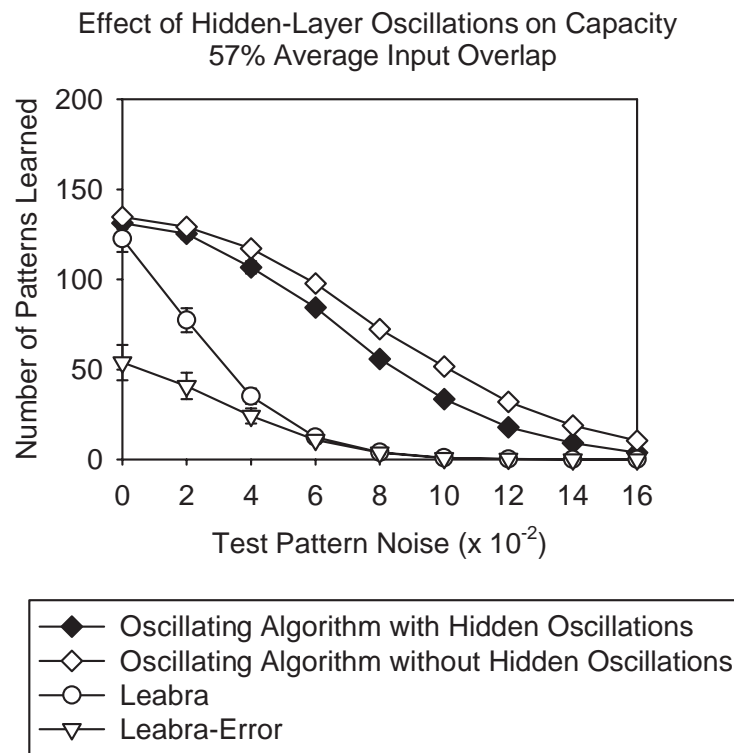


Figure 10: Capacity scores (number of patterns learned out of 200) for the oscillating algorithm with and without hidden-layer oscillations, given 57% input pattern overlap. Results for Leabra and Leabra-Error are included for comparison purposes. The same qualitative pattern of results is present both with and without hidden-layer oscillations.

References

- Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, 9, 147–169.
- Almeida, L. B. (1989). Backpropagation in nonfeedforward networks. In I. Aleksander (Ed.), *Neural computing*. Kogan Page.
- Anderson, M. C. (2003). Rethinking interference theory: Executive control and the mechanisms of forgetting. *Journal of Memory and Language*, 49, 415–445.
- Anderson, M. C., Bjork, E. L., & Bjork, R. A. (2000). Retrieval-induced forgetting: Evidence for a recall-specific mechanism. *Memory & Cognition*, 28, 522.
- Anderson, M. C., Bjork, R. A., & Bjork, E. L. (1994). Remembering can cause forgetting: Retrieval dynamics in long-term memory. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 5, 1063–1087.
- Anderson, M. C., & Spellman, B. A. (1995). On the status of inhibitory mechanisms in cognition: Memory retrieval as a model case. *Psychological Review*, 102, 68.
- Berry, S. D., & Seager, M. A. (2001). Hippocampal theta oscillations and classical conditioning. *Neurobiology of Learning and Memory*, 76, 298–313.
- Bienenstock, E. L., Cooper, L. N., & Munro, P. W. (1982). Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2(2), 32–48.
- Buzsáki, G. (2002). Theta oscillations in the hippocampus. *Neuron*, 33, 325–340.
- Fox, E. (1995). Negative priming from ignored distractors in visual selection. *Psychonomic Bulletin and Review*, 2, 145–173.
- Freedman, J. L. (1965). Long-term behavioral effects of cognitive dissonance. *Journal of Experimental Social Psychology*, 1, 145–155.
- Glucksberg, S., Newsome, M. R., & Goldvarg, G. (2001). Inhibition of the literal: Filtering metaphor-irrelevant information during metaphor comprehension. *Metaphor and Symbolic Activity*, 16, 277–293.
- Grossberg, S. (1976). Adaptive pattern classification and universal recoding I: Parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23, 121–134.
- Grossberg, S. (1999). How does the cerebral cortex work? learning, attention, and grouping by the laminar circuits of visual cortex. *Spatial Vision*, 12, 163–186.
- Hasselmo, M. E. (1995). Neuromodulation and cortical function: Modeling the physiological basis of behavior. *Behavioural Brain Research*, 67, 1–27.
- Hasselmo, M. E., Bodelon, C., & Wyble, B. P. (2002). A proposed function for hippocampal theta rhythm: Separate phases of encoding and retrieval enhance reversal of prior learning. *Neural Computation*, 14, 793–818.
- Hinton, G. E. (1989). Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural Computation*, 1, 143–150.
- Hinton, G. E., & Ghahramani, Z. (1997). Generative models for discovering sparse distributed representations. *Philosophical Transactions of the Royal Society (London) B*, 352, 1177–1190.
- Hinton, G. E., & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In D. E. Rumelhart, J. L. McClelland, & PDP Research Group (Eds.), *Parallel distributed processing. Volume 1: Foundations* (Chap. 7, pp. 282–317). Cambridge, MA: MIT Press.
- Holscher, C., Anwyl, R., & Rowan, M. J. (1997). Stimulation on the positive phase of hippocampal theta rhythm induces long-term potentiation that can be depotentiated by stimulation on the negative phase in area CA1 in vivo. *Journal of Neuroscience*, 17, 6470.
- Huerta, P. T., & Lisman, J. E. (1996). Synaptic plasticity during the cholinergic theta-frequency oscillation in vitro. *Hippocampus*, 49, 58–61.
- Hyman, J. M., Wyble, B. P., Goyal, V., Rossi, C. A., & Hasselmo, M. E. (2003). Stimulation in hippocampal region CA1 in behaving rats yields long-term potentiation when delivered to the peak of theta and long-term depression when delivered to the trough. *Journal of Neuroscience*, 23, 11725–11731.
- Marr, D. (1971). Simple memory: A theory for archicortex. *Philosophical Transactions of the Royal Society (London) B*, 262, 23–81.
- Mayr, U., & Keele, S. (2000). Changing internal constraints on action: the role of backward inhibition. *Journal of Experimental Psychology: General*, 1, 4–26.
- McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102, 419–457.
- McNaughton, B. L., & Morris, R. G. M. (1987). Hippocampal synaptic enhancement and information

- storage within a distributed memory system. *Trends in Neurosciences*, 10(10), 408–415.
- Minai, A. A., & Levy, W. B. (1994). Setting the activity level in sparse random networks. *Neural Computation*, 6, 85–99.
- Movellan, J. R. (1990). Contrastive Hebbian learning in the continuous Hopfield model. In D. S. Touretzky, G. E. Hinton, & T. J. Sejnowski (Eds.), *Proceedings of the 1989 Connectionist Models Summer School* (pp. 10–17).
- Norman, K. A., Newman, E. L., & Detre, G. J. (submitted). A neural network model of retrieval-induced forgetting. *Psychological Review*.
- Norman, K. A., & O'Reilly, R. C. (2003). Modeling hippocampal and neocortical contributions to recognition memory: A complementary-learning-systems approach. *Psychological Review*, 4, 611–646.
- O'Reilly, R. C. (1996). *The Leabra model of neural interactions and learning in the neocortex*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA.
- O'Reilly, R. C. (2001). Generalization in interactive networks: The benefits of inhibitory competition and Hebbian learning. *Neural Computation*, 13, 1199–1242.
- O'Reilly, R. C., & Munakata, Y. (2000). *Computational explorations in cognitive neuroscience: Understanding the mind by simulating the brain*. Cambridge, MA: MIT Press.
- O'Reilly, R. C., & Norman, K. A. (2002). Hippocampal and neocortical contributions to memory: Advances in the complementary learning systems framework. *Trends in Cognitive Sciences*, 12, 505–510.
- Pineda, F. J. (1987). Generalization of backpropagation to recurrent neural networks. *Physical Review Letters*, 18, 2229–2232.
- Raghavachari, S., Kahana, M. J., Rizzuto, D. S., Caplan, J. B., Kirschen, M. P., Bourgeois, B., Madsen, J. R., & Lisman, J. E. (2001). Gating of human theta oscillations by a working memory task. *Journal of Neuroscience*, 9, 3175–3183.
- Rizzuto, D. S., Madsen, J. R., Bromfield, E. B., Schulze-Bonhage, A., Seelig, D., Aschenbrenner-Scheibe, R., & Kahana, M. J. (2003). Reset of human neocortical oscillations during a working memory task. *Proceedings of the National Academy of Sciences*, 13, 7931–7936.
- Rolls, E. T. (1989). Functions of neuronal networks in the hippocampus and neocortex in memory. In J. H. Byrne, & W. O. Berry (Eds.), *Neural models of plasticity: Experimental and theoretical approaches* (pp. 240–265). San Diego, CA: Academic Press.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, & PDP Research Group (Eds.), *Parallel distributed processing. Volume 1: Foundations* (Chap. 8, pp. 318–362). Cambridge, MA: MIT Press.
- Seager, M. A., Johnson, L. D., Chabot, E. S., Asaka, Y., & Berry, S. D. (2002). Oscillatory brain states and learning: Impact of hippocampal theta-contingent training. *Proceedings of the National Academy of Sciences*, 99, 1616–1620.
- Simoncelli, E. P., & Olshausen, B. A. (2001). Natural image statistics and neural representation. *Annual Review of Neuroscience*, 24, 193–216.
- Toth, K., Freund, T. F., & Miles, R. (1997). Disinhibition of rat hippocampal pyramidal cells by GABAergic afferents from the septum. *Journal of Physiology*, 500, 463–474.