BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

COGNITIVE COMPUTING: ALGORITHM DESIGN IN THE INTERSECTION OF COGNITIVE SCIENCE AND EMERGING COMPUTER ARCHITECTURES

by

BENJAMIN CHANDLER

B.S., Carnegie Mellon University, 2007

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2014

Approved by

First Reader

Ennio Mingolla, PhD
Professor and Chair, Department of Speech-Language Pathology and Audiology
Northeastern University

Second Reader

Gregory Snider, MS
Researcher, HP Labs
Hewlett-Packard Company

Third Reader

Arash Yazdanbakhsh, MD, PhD
Research Assistant Professor, Center for Computational Neuroscience and Neural Technology

COGNITIVE COMPUTING: ALGORITHM DESIGN IN THE INTERSECTION OF COGNITIVE SCIENCE AND EMERGING COMPUTER ARCHITECTURES

(Order No.)

BENJAMIN CHANDLER

Boston University, Graduate School of Arts and Sciences, 2014

Major Professor: Ennio Mingolla, PhD, Professor and Chair
Department of Speech-Language Pathology and
Audiology, Northeastern University

ABSTRACT

For the first time in decades computers are evolving into a fundamentally new class of machine. Transistors are still getting smaller, more economical, and more power-efficient, but operating frequencies leveled off in the mid-2000's. Today, improving performance requires placing a larger number of slower processing cores on each of many chips. Software written for such machines must *scale out* over many cores rather than *scaling up* with a faster single core. Biological computation is an extreme manifestation of such a many-slow-core architecture and therefore offers a potential source of ideas for leveraging new hardware. This dissertation addresses several problems in the intersection of emerging computer architectures and biological computation, termed *Cognitive Computing:* What mechanisms are necessary to maintain stable representations in a large distributed learning system? How should complex biologically-inspired algorithms be tested? How do visual sensing limitations like occlusion influence performance of classification algorithms?

Neurons have a limited dynamic output range, but must process real-world signals over a wide dynamic range without saturating or succumbing to endogenous noise. Many existing neural network models leverage spatial competition to address this issue, but require hand-tuning of several parameters for a specific, fixed distribution of inputs. Integrating spatial competition with a stabilizing learning process produces a neural network model capable of autonomously adapting to a non-stationary distribution of inputs.

Human-engineered complex systems typically include a number of architectural features to curtail complexity and simplify testing. Biological systems do not obey these constraints. Biologically-inspired algorithms are thus dramatically more difficult to engineer. Augmenting standard tools from the software engineering community with features targeted towards biologically-inspired systems is an effective mitigation.

Natural visual environments contain objects that are occluded by other objects. Such occlusions are under-represented in the standard benchmark datasets for testing classification algorithms. This bias masks the negative effect of occlusion on performance. Correcting the bias with a new dataset demonstrates that occlusion is a dominant variable in classification performance. Modifying a state-of-the-art algorithm with mechanisms for occlusion resistance doubles classification performance in high-occlusion cases without penalty for unoccluded objects.

Contents

List of Tables

List of Figures

List of Abbreviations

Cog	....	Cog ex Machina
ConvNet	....	Convolutional Neural Network
CPU	....	Central Processing Unit
CSV	....	Comma-Separated Values
GPU	....	Graphics Processing Unit
hRCF	....	Homeostatic Recurrent Competitive Field
ItEM	....	Iterative Evolution of Models
NORB	....	NYU Object Recognition Benchmark
PASCAL VOC	....	PASCAL Visual Object Classes
RCF	....	Recurrent Competitive Field
SORBO	....	Synthetic Object Recognition Benchmark with Occlusion
VCS	....	Version Control System

Chapter 1

Introduction

For the first time in decades computers are evolving into a fundamentally new class of machine. Transistors are still getting smaller, more economical, and more power-efficient, but operating frequencies leveled off in the mid-2000's and have been steadily falling ever since. The key to faster computers used to be an exponential growth in clock frequency. Today, improving performance requires placing a larger number of slower processing cores on each of many chips (Kogge et al., 2008). Software written for such machines must *scale out* over many cores rather than *scaling up* with a faster single core. Biological computation is an extreme manifestation of such a many-slow-core architecture and therefore offers a potential source of ideas for leveraging new hardware. This dissertation addresses several problems in the intersection of emerging computer architectures and biological computation, termed *Cognitive Computing:* What mechanisms are necessary to maintain stable representations in a large distributed learning system? How should complex biologically-inspired algorithms be tested? How do visual sensing limitations like occlusion influence performance of classification algorithms?

Two competing forces are driving the shift to machines with a larger number of slower cores. Physics is pushing the semiconductor industry inexorably towards massively parallel computation and tight co-location of processing and memory – architectural principles expressed in biological systems (Kogge et al., 2008; Mckenzie et al., 2010; Snider et al., 2011). Decades of software work, however, relied on fast single-core processor performance. Legacy code and the cost of re-training programmers to work effectively with new architectures are moderating the rate of the transition. This growing gap between the needs of software and the constraints of performing computation in the physical world offers considerable opportunity for new approaches to computing (Mckenzie et al., 2010; Snider, 2007).

The Cog ex Machina (or Cog) project at HP Labs is an attempt to leverage certain similarities between emerging computer architectures and biological computation to tackle problems at the cutting edge of artificial intelligence and machine learning (Snider et al., 2011). Cog provides a programming model, optimizing compiler, and runtime system, all of which are designed to scale to hardware with millions of processor cores. Consistent with biological brain architectures, Cog asserts that the most practical route to high power efficiency is co-locating memory and computation in a system with millions of relatively slow processor elements. This strategy is pragmatic. Clusters of commodity graphics processors (GPUs) are already a reasonably good match for the Cog programming model. As hardware evolves, Cog applications can continue riding the commodity curve. The high-performance computing community expects a thousand-fold increase in hardware capability over the next decade. The majority of this increase will come from greater parallelism, which Cog is designed to leverage (Dongarra et al., 2011; Kogge et al., 2008).

A good programming model, while necessary, is insufficient for scaling towards practical problems. Brains are massively parallel and employ co-located processing and memory, but are not architected like conventional software systems. Complex software systems are typically designed with minimally-interacting modules of functionality with as few circular dependencies as possible. Modules are arranged in a hierarchy of layers, such that each layer leverages the abstractions of the layer below and provides higher orders of abstraction to the layer above. Brains, in contrast, do not exhibit modularity and abstraction to nearly the same degree and embed *learning* everywhere (Felleman and Van Essen, 1991; Modha and Singh, 2010). Standard software engineering techniques for managing complexity in large systems are thus hard to apply directly. This means that principles from neuroscience and cognitive science offer useful hints, but do not map directly to emerging machine architectures.

Conversely, very large clusters of computers cannot be programmed like small machines. New frameworks are necessary to abstract away parallelism and hardware faults while preserving programmer productivity (Dean and Ghemawat, 2008). Machine learning and artificial intelligence algorithms designed for fast single-core processors leverage decades of mathematics and engineering to run efficiently on digital computers, but frequently cannot translate directly to emerging massively parallel architectures. A synthesis of biological and computer science insights is necessary to tackle next-generation problems.

An approach informed by both biology and computer science is particularly well-suited to vision problems. Cog performs best on problems with strong *data locality* and high *implicit parallelism*. A simple example of an algorithm that exhibits these properties is convolution. Each output point in a convolution is independent from the other output points and depends only on a spatially localized region of the input. This means that a highly parallel computer can distribute the computation across many cores with very little inter-core communication. Processing bank transactions, in contrast, is a problem with poor locality and poor implicit parallelism because each transaction can touch any account in the system. This is an implicit *serial bottleneck*. Such bottlenecks can pop up in unexpected places. For example, anisotropic diffusion is a computer vision technique for smoothing out noise while preserving edge information (Perona and Malik, 1990). While each iteration of this algorithm is highly parallel and depends only on local information, running anisotropic diffusion to convergence requires information from the entire field (Fischl and Schwartz, 1997).

Many vision problems fit well into parallel architectures due to the locality and implicit parallelism of visual information. Information flowing from the retina is organized in a *retinotopic map* that preserves (with some local spreading) the adjacency relationship of information from neighboring photoreceptors. This retinotopic organization is a common feature in many parts of the visual system. Mammalian visual cortex contains tens of regions that process information in retinotopic maps (Felleman and Van Essen, 1991). This property implies high locality and implicit parallelism because long-range signal transmission is slow and vision must operate under tight time constraints. Given these physical limitations, the visual system can only produce quick results if most computation relies on local information and most signaling happens over short distances.

A sophisticated visual system offers the additional benefit of adaptable behavior. Organisms as simple as the horseshoe crab have behaviors that are visually driven. In the case of the horseshoe crab, however, the information flowing from the retina is highly specialized to a single behavioral task, namely finding a mate (Barlow et al., 2001).

Mammals lack this degree of specialization, instead relying on a large number of interacting cortical maps to handle general visual tasks. These cortical maps interact not just with each other, but *guide behavior in real time.* This closed loop between perception and action allows the organism to resolve perceptual ambiguity and thus operate successfully in a challenging environment. Closed-loop operation both internally and in concert with the environment is the core of cognitive computing. This dissertation addresses a cluster of problems aimed at stepping towards such a system:

1. *Homeostatic plasticity:* necessary to maintain stability in a large-scale learning

system

2. *Testing and tuning cognitive computing algorithms:* tools to simplify the development of complex biologically-inspired algorithms

3. *Robust classification of occluded objects:* algorithms that degrade gracefully in response to visual occlusions

   (a) *Measuring object classification performance with occlusion:* a new dataset designed to diagnose the performance of classification algorithms in occluded environments

   (b) *Building an occlusion-resistant object classifier:* a new object classification algorithm designed for state-of-the-art performance in non-occluded environments but graceful degradation as the percentage of occluded pixels increases

Chapter 2

Homeostatic plasticity

2.1 Introduction

The noise/saturation dilemma is a classical problem for neural network models of static spatial pattern processing (Grossberg, 1973). Neurons have a small, fixed output range of approximately two orders of magnitude in spiking rate. If the input signal is too large, the output will saturate. If the signal is too small, endogenous noise will dominate the output. Neither of these behaviors are desirable in a neural system. Communicating behaviorally worthless information squanders valuable metabolic energy. A well-constructed biological model should adjust its sensitivity to properly account for the range of inputs.

Some competitive neural network models handle the noise-saturation dilemma by responding to spatial ratios in the input rather than absolute values. This strategy works because it encodes the input in an approximately magnitude-invariant manner. A high-intensity input pattern will produce a similar output as a low-intensity input pattern if the ratios among inputs to nodes in the network are the same. This is essentially equivalent to *divisive normalization*, where each output is divided by the sum of all outputs.

While a ratio response is much less susceptible to noise and saturation problems, it discards information about the absolute intensity of the input signal and requires careful tuning of the network parameters. If the network parameters do not match the input, the network will lose information to saturation or noise even though it responds primarily to ratios (Grossberg, 1973). This is particularly relevant in changing environments. If a recurrent competitive field (RCF) is presented with nothing but noise, the ratio response *should* be noisy. Similarly, if the input is exceptionally large compared to past inputs, complete saturation of the output might be a good thing. An unmodified RCF will attempt to extract ratios in both of these cases, rather than allowing the entire output to *quench*, meaning suppress its outputs to zero, or saturate.

The purpose of the present chapter is to present a network that responds with some sense of a temporal "running average" to a *series* of spatial input patterns. The network should produce uniform zero output in response to a noisy input pattern that is very weak and saturate uniformly in response to a noisy pattern that is much stronger than all recent ones. The input levels at which the network should completely quench or saturate are a function of the *past inputs* to the network. As such, this is a *temporal* noise-saturation dilemma. The network must respond to spatial ratios for each input, as in the standard noise-saturation dilemma. It must also *learn the distribution of the inputs over time* such that it can suppress input patterns that are exceptionally small and saturate in response to input patterns that are exceptionally large.

Note that temporal transients present a unique and orthogonal set of challenges that this work does not address (Gaudiano, 1993; Francis et al., 1994). Temporal transients require a system that can adapt over a timescale of milliseconds. The temporal noise-saturation dilemma as defined here applies over timescales of minutes to days.

2.1.1 Synaptic scaling

The biological mechanism, synaptic scaling, described by Turrigiano et al. (1998) and modeled in van Rossum et al. (2000), offers a potential solution for the temporal noise-saturation dilemma. Synaptic scaling is an example of a *homeostatic* learning process. Homeostatic learning acts as a stabilizing counter-force for associative learning processes. Associative processes modify the strength of the connection between two neurons as a function of the correlation in activity of those two neurons. Generally, correlated activity increases the connection strength and uncorrelated activity decreases the connection strength. Pure associative learning is fundamentally unstable, however. Increasing the connection strength between two neurons increases the correlation, triggering a further increase in the connection strength. Homeostatic learning mechanisms pull the system back towards a stable regime that maximizes information transmission and minimizes information loss to saturation and noise (Turrigiano and Nelson, 2004). This intrinsic tuning property may allow the system to automatically find the correct gain and signal transmission properties while minimizing information loss.

Synaptic scaling serves a similar computational purpose as Oja's rule and generalized Hebbian learning (Oja, 1982; Hyvärinen and Oja, 1998). The scaling process uses only information available locally at a connection, but maintains approximately normalized connection strengths with *non-associative* learning. A non-associative learning process adjusts the strength of a connection without regard to the correlation in activity levels between the pre-connection and post-connection nodes.

2.2 Homeostatic recurrent competitive field model

The homeostatic recurrent competitive field (hRCF) model extends the Grossberg (1973) recurrent competitive field (RCF) model with the homeostatic plasticity model from van Rossum et al. (2000). The RCF model is defined by a single ordinary differential equation:

$$\dot{x_i} = -Ax_i + (B - x_i)f(x_i) - x_i \sum_{k \neq i} f(x_k) + I_i \tag{2.1}$$

where x_i is the activity level of node i, A is a decay constant, B is a maximum activity level constant, $f(x)$ is a signal function, and I_i is the external input to node i. Such a network contains a single layer of nodes as shown in Figure 2·1. Each node has an activity level that decays to zero at a rate proportional to the network parameter A. Each node has a self-excitation connection that increases activity up to a maximum level, B, modulated by the signal function $f()$. Each node inhibits all other nodes, modulated by the signal function $f()$. Each node receives an external bottom-up input defined by I_i.

The analysis in Grossberg (1973) focuses primarily on the behavior of the network after all the input terms are set to zero. This behavior is governed primarily by the signal function, $f()$. There are three primary classes of signal function: linear, faster-than-linear, and slower-than-linear.

If f is a linear function, such as $f(x) = x$, the network *normalizes* the input and *preserves the pattern*. Normalization means the network output is approximately scaled to the range $[0, B)$, no matter the range of the input. Pattern preservation means the node-to-node ratios in the output match those in the input.

A faster-than-linear signal function, such as $f(x) = x^2$, produces *winner-take-all* behavior. In this case, the network allows the largest input node to survive while all other nodes are quenched to zero. This is also called $0 - 1$ behavior, as the network sharpens the input pattern to a single high-activity node.

A slower-than-linear signal function, such as $f(x) = \frac{x}{1+x}$ acts to *homogenize* the

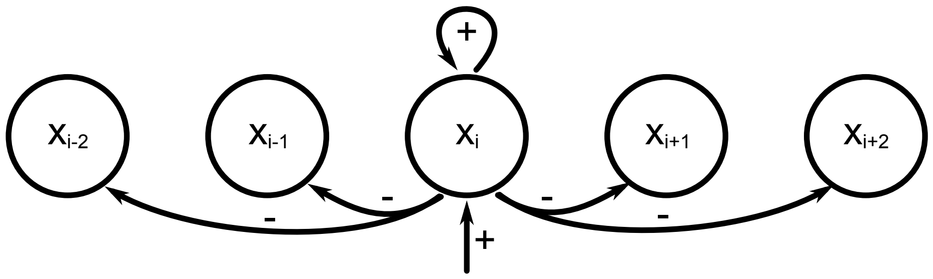


Figure 2·1: RCF connectivity. Node x_i has a bottom-up excitatory input and a recurrent self-excitation connection. It inhibits all other nodes in the network.

input pattern. All nodes converge to the same non-zero value.

More complex behaviors are possible by building composite signal functions. A sigmoid function, such as $f(x) = \frac{x^2}{\alpha^2+x^2}$ with shape parameter α, is an example. This function joins a faster-than-linear region to a slower-than-linear region at an inflection point defined by α. The effect is a network that *quenches* small signals to zero, while simultaneously preserving and homogenizing large signals. This behavior is *k-winner-take-all*. Like the regular winner-take-all case, the output is bi-state. A given node either wins and maintains a high activity level, or drops to zero. A standard winner-take-all network allows exactly one winner. A k-winner-take-all network allows k winners. The precise value of k depends on the input pattern and the network parameters and has no known analytical expression. Figure 2·2 illustrates network behavior for each of the four major type of signal functions.

The hRCF model adds a mechanism inspired by the van Rossum et al. (2000) synaptic scaling model. The core of this model is a single differential equation per tuned weight:

$$\dot{w} = \beta w[a_{goal} - a] \tag{2.2}$$

where w is the weight intensity, β is a rate parameter, a_{goal} is a target average activity

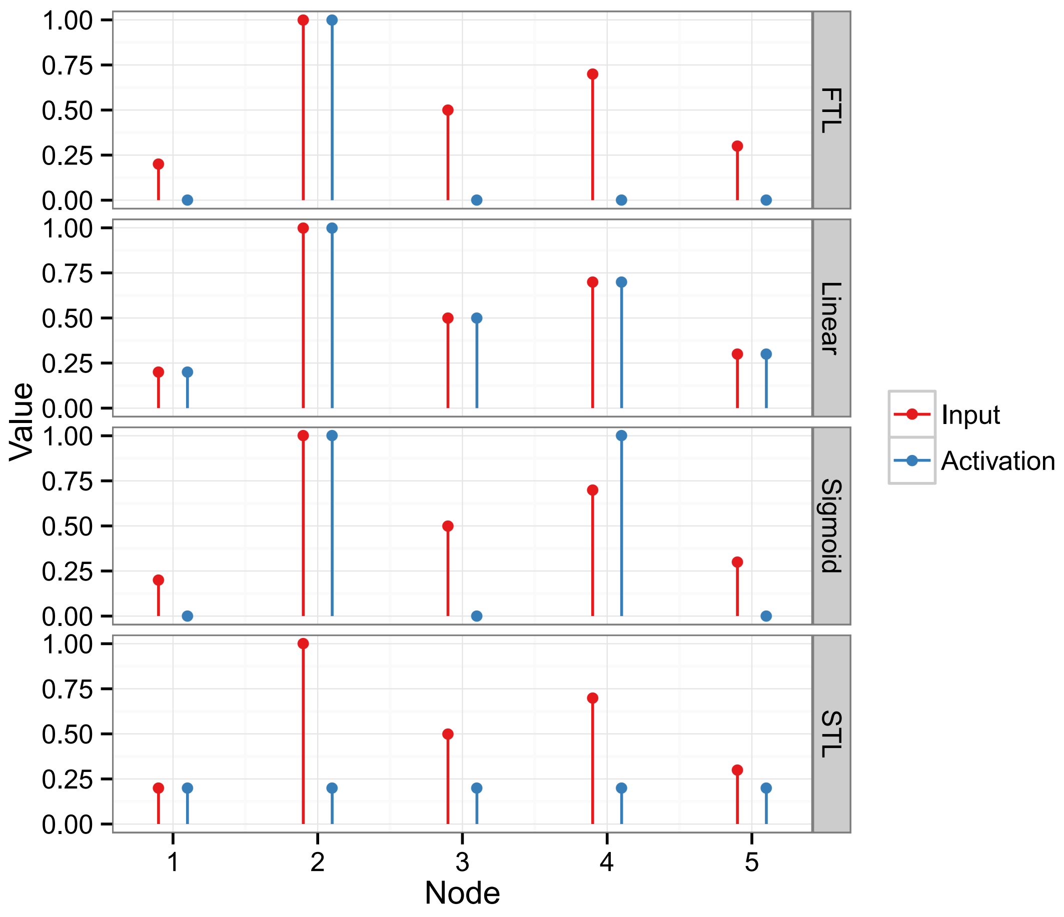


Figure 2·2: RCF behavior depends on the network signal function. Every recurrent connection in the network passes through a signal function, $f(x)$. The form of the signal function determines the type of pattern processing exhibited by the network. With a *linear* signal function of the form $f(x) = x$, the network preserves the input pattern. With a *faster-than-linear* (FTL) signal function of the form $f(x) = x^2$, the network sharpens the input pattern to a single winner. With a *slower-than-linear* (STL) signal function of the form $f(x) = \frac{x}{1+x}$, the network homogenizes the input. A *sigmoid* signal function of the form $f(x) = \frac{x^2}{\alpha^2+x^2}$ combines a slower-than-linear function and faster-than-linear function at an inflection point defined by parameter α. This produces k-winner-take-all behavior. The top k inputs are amplified, where the remaining nodes are suppressed to zero.

level parameter, and a is the current average activity level. This scaling rule is *multiplicative* in that the change in the weight is proportional to current value of the weight, not just the error.

The van Rossum et al. (2000) model only addresses excitatory synapses. The data in Rutherford et al. (1998), however, suggests that synaptic scaling may have a corresponding effect in inhibitory synapses. The hRCF model incorporates both studies by scaling excitatory and inhibitory synapses in a multiplicative manner, but in opposite directions.

2.2.1 First-generation hRCF model

There are two versions of the hRCF models, differing primarily in how the average activity level is defined. The first-generation model sums the instantaneous activities of the nodes in the network and averages the resulting value over time. The second-generation system defines the average activity level as the average *count* of active nodes. This more complex definition of average activity level has desirable behavior for networks with a sigmoid signal function.

The first-generation hRCF model is defined by the following set of differential equations:

$$\dot{x}_i = -Ax_i + (B - x_i)(f(x_i)w + I_i) - x_i \sum_{k \neq i} (f(x_k)W + I_k) \tag{2.3}$$

$$\dot{a} = \frac{1}{\tau}(-a + \sum_i x_i) \tag{2.4}$$

$$\dot{w} = \beta w(G - a) \tag{2.5}$$

$$\dot{W} = \beta W(a - G) \tag{2.6}$$

The network structure and dynamics are identical to a standard Grossberg (1973)

RCF, save for the addition of w and W scaling terms and a different connection pattern between the input layer and network layer. The scaling terms act to tune the balance of excitation and inhibition in the network. w determines the intensity of excitation. It increases when the average activity level is too small and decreases when the average activity level is too large. W determines the intensity of inhibition. It decreases when the average activity level is too small and increases when the average activity level is too large. These two processes are a direct application of synaptic scaling, and are identical save for a sign change. β is the rate parameter and G is the goal activity level, replacing a_{goal} from van Rossum et al. (2000).

The input layer to network layer connectivity is different from a standard Grossberg (1973) RCF in that it is all-to-all instead of one-to-one. Each node receives excitatory input from the corresponding input node and inhibitory input from the remaining input nodes. As the interesting segment of the network behavior is that which takes place after input is switched off, this change only serves to decrease the convergence time.

The average activity level a is similar to the formulation in van Rossum et al. (2000). It uses an identical time constant parameter, τ. As the underlying system is a continuous network instead of a single spiking neuron, the instantaneous activity the average activity level tracks is the sum across x_i.

The network parameters assume the values in Table 2.1. The network size is set just large enough that the pattern processing is behavior is readily apparent. Network connectivity is all-to-all, so larger networks are dramatically slower to simulate. The decay rate is set to a reasonable value given the activation bound, but the network is not particularly sensitive to this value. The activation bound is similarly unconstrained. The chosen value is numerically stable and consistent with prior work. The sigmoid inflection point is set given the activation bound. Networks with a sigmoid

Parameter	Description	Value
N	Number of nodes	5
A	Decay rate	1
B	Upper activation bound	3
α	Inflection point of sigmoid signal function	0.5
τ	Time scale for slow averaging process	400
β	Homeostasis rate constant	0.005
G	Target activity level	3

Table 2.1: Parameters for first-generation hRCF model

Name	Function
Linear	$f(x) = x$
Slower than linear	$f(x) = \frac{x}{1+x}$
Faster than linear: $n = 2$	$f(x) = x^2$
Faster than linear: $n = 4$	$f(x) = x^4$
Sigmoid: $n = 2$	$f(x) = \frac{x^2}{\alpha^2+x^2}$
Sigmoid: $n = 4$	$f(x) = \frac{x^4}{\alpha^4+x^4}$

Table 2.2: Signal functions for first-generation hRCF model

signal function will tend to quench units with an activation below this value and homogenize units with an activation above it. The synaptic scaling parameters τ and β are set such that synaptic scaling operates on a much slower time scale than the fast network dynamics. The target activity level is set such that synaptic scaling process will try to keep an average aggregate network activity equivalent to one fully-activated node. This is a safe choice for every signal function.

The signal functions, $f(x)$, are chosen from Table 2.2. This set of signal functions includes at least one example of each major class of pattern processing behaviors. The faster-than-linear and sigmoid cases include two degrees of nonlinearity, described by the exponent n. A higher exponent indicates a sharper non-linearity.

The initial conditions are set as follows:

$$x_i(0) = 0 \tag{2.7}$$

$$x(0) = G \tag{2.8}$$

$$w(0) = 1 \tag{2.9}$$

$$W(0) = 1 \tag{2.10}$$

The network is not sensitive to its initial conditions, so long as $w(0)$ and $W(0)$ are positive. Negative values are outside the normal range, and values of zero would disable tuning. $a(0)$ is set as if the network were already at an equilibrium average activity level of G to keep down the simulation time, but non-equilibrium initial values produce identical steady-state behavior.

Each simulation consists of 500 presentation intervals (or *epochs*), each running for ten time units. In the first five time units of each interval, a random input pattern is presented to the network. Each element in the input pattern is independently chosen from a uniform random distribution in the range $[0, 1]$. For the remaining five time units, input is switched off and the network is allowed to reverberate. Figure 2·3 illustrates a typical epoch with a k-winner-take-all network. In both the input and no-input cases, five time units is typically sufficient for the fast network dynamics to equilibrate. At the end of each epoch, each x_i is set back to the initial value of zero. The values of a, w, and W carry over from epoch to epoch.

The input pattern $[0.2, 1.0, 0.4, 0.8, 0.2]$ is used to probe pattern processing behavior. This pattern is consistent with that used to illustrate pattern processing in Grossberg (1973). To visualize pattern processing behavior as of a given presentation interval, the network is copied and re-simulated using the diagnostic input pattern instead of a random pattern.

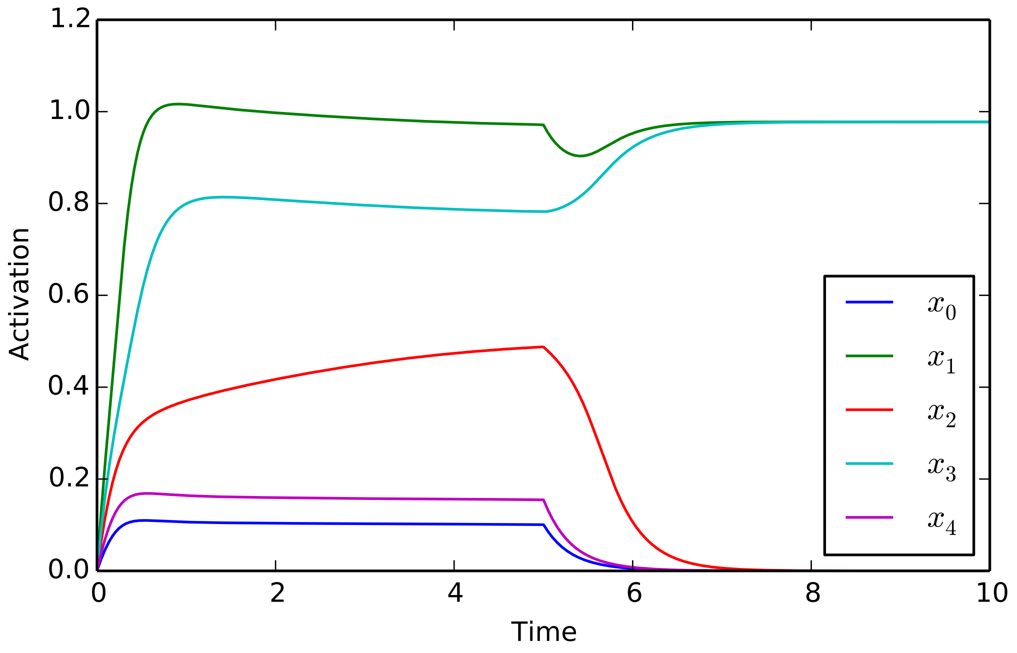


Figure 2·3: Example trace of a five-node k-winner-take-all RCF network over a single presentation interval. The instantaneous activations of the five nodes are labeled x_1 through x_5. External input is supplied for the first five time units of the simulation. At $t = 5$, the input switches off and the network is free to reverberate. In this case the network chooses the top two nodes as winners and suppresses the remaining three nodes to zero. Node two has not yet reached a steady-state value by $t = 5$, illustrating a subtlety of the simulation protocol. Five time units of input are typically sufficient for the network to reach equilibrium. This is not always the case, however. In some instances the simulation protocol will force the network to respond before the node activities have settled.

2.2.2 Second-generation hRCF model

The second-generation hRCF model replaces the definition of average activity level in the first-generation model with an average count, k, of active nodes. Tuning k directly allows the network to exhibit k-winner-take-all behavior reliably with a sigmoid signal function.

The second-generation hRCF model is defined by the following set of differential equations:

$$\dot{x}_i = -Ax_i + (B - x_i)(I_i + f(x_i))w - x_i \sum_{k \neq i} (f(x_k) + I_k)W \tag{2.11}$$

$$k = \sum_i \tanh(10x_i) \tag{2.12}$$

$$g(t) = \frac{1}{1 + e^{25(-t+7.5)}} \tag{2.13}$$

$$\dot{a} = \frac{1}{\tau}(-a + k/N)g(t) \tag{2.14}$$

$$\dot{w} = \beta w(G - a) \tag{2.15}$$

$$\dot{W} = \beta W(a - G) \tag{2.16}$$

The revised equations differ from the first-generation system in three major ways. First, the excitatory and inhibitory tuning coefficients (w and W) scale both the bottom-up and recurrent network connections. The first-generation system only applied scaling to the recurrent connections. This change allows the network to converge more quickly.

The second and third changes are both in the average activity level, a. The first-generation system tracks the long-term average of the sum of the x_i terms. The second-generation system integrates a strongly non-linear function that estimates the instantaneous number of active nodes, or k. Network nodes with low or zero instantaneous activity (x_i) contribute nothing to this term. Nodes with a moderate to high instantaneous activity contribute a value of one. The constant 10 in the expression for k defines "low" at an appropriate level. Instead of tracking the sum of x_i terms, the average activity a instead tracks k directly. This allows the homeostatic plasticity process to directly tune the *number* of active nodes.

Averaging the k term over the entire presentation interval over-estimates the steady-state value of k for a given epoch. All non-zero x_i terms immediately increase the value of k by one. This is *all* the x_i terms during the input presentation phase of each epoch. The third and final modification is a $g(t)$ function aimed at preventing a from changing until the final 2.5 time units of each epoch. By this point, the system is typically close to convergence and the instantaneous value for k is an accurate reflection of the steady-state value. The constants in the expression for $g(t)$ define an extremely sharp sigmoid function that transitions from zero to one at $t = 7.5$.

The parameters match those of the first-generation system with three exceptions. First, τ has decreased from 400 to 100. This change compensates for the fact that a is only permitted to change for the final 2.5 time units of each epoch. The value of a changes for a quarter of the time it would have in the first-generation system, but it changes four times as quickly during that period. The second change, a β value of 0.01, causes a similar increase in the rate of tuning. This allows the system to reach a tuned state in fewer epochs.

The final change is a goal activity level, G, of 0.5. This new goal activity level corresponds to a minor change in the system of differential equations. The first-generation system treated G as the *aggregate* goal activity level across the entire network. The second-generation system uses G as a goal activity level for *each* node. The aggregate goal value with a five-node network and $G = 0.5$ is therefore 2.5. This indicates a target mean-k value of 2.5, or half the network active at the end of each epoch.

All simulations with the second-generation hRCF model use a sigmoid signal function with $n = 4$. This signal function exhibited the strongest k-winner-take-all behavior in the simulations with the first-generation hRCF model.

The simulation protocol for the second-generation hRCF system is structured in the same manner as with the first-generation system. It includes three times as many epochs, however, and varies the magnitude of the input as a function of the epoch number. Figure 2·4 shows the mean input intensity as a function of the epoch number. For the first 500 epochs, inputs are chosen from the range $[0, 10]$. The second 500 epochs lower the input range to $[0, 0.1]$. The final 500 epochs increase the range to $[0, 100]$. For each block of epochs, the system must adjust its sensitivity to compensate for the intensity of the input.

2.3 Results

Simulation results for the first-generation model with a linear signal function are shown in Figure 2·5. Part (a) shows the network state at the end of each of the 500 epochs for each of the model equations. The plot for x includes traces for each of the five x_i equations. The rapid mixing in x indicates that the network is successfully reverberating with each new random input pattern while the synaptic scaling process tunes the network. In this case, the tuning process up-regulates excitation and down-regulates inhibition to reach the goal activity level.

Part (b) shows the pattern processing behavior sampled at four intervals over the simulation. These results were computed using the probe input pattern. As the simulation proceeds, the tuning process shifts the network balance towards excitation. While the pattern remains roughly constant, the magnitude increases accordingly.

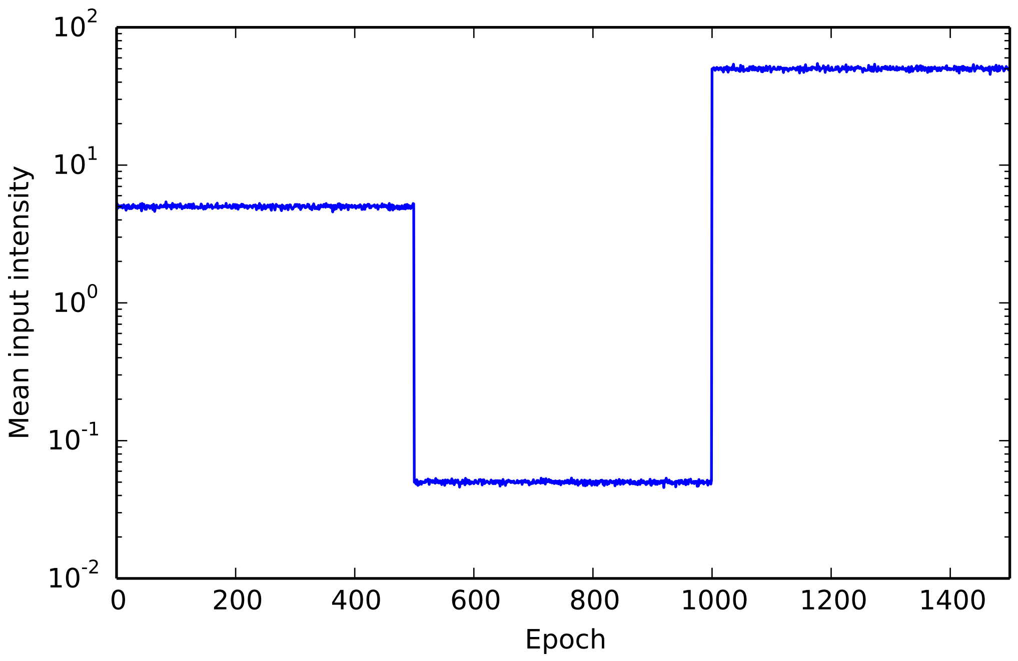


Figure 2·4: Mean input intensity by epoch. For each epoch (presentation interval), each network receives a random input chosen from a uniform random distribution. The lower bound of this distribution is always 0. The upper bound is a function of the epoch. For the first phase of 500 epochs, the upper bound is 10. The second phase lowers the upper bound to 0.1. The third phase increases the upper bound to 100. This protocol contains a sharp high-to-low intensity transition at epoch 501 and a sharp low-to-high intensity transition at epoch 1001.

Figures 2·6 through 2·10 use a format identical to Figure 2·5. In Figure 2·6, tuning with a slower-than-linear signal function produces similar results to the linear case. The output pattern remains roughly constant, but the magnitude shifts to bring the network to the goal activity level.

Figure 2·7 shows a faster-than-linear signal function with $n = 2$. In this case, the tuning process simply acts to sharpen the output slightly. The fast state dynamics are also important in this case, as the rapid changes indicate that the network is successfully picking a new winner on each presentation interval.

Figure 2·8 shows a faster-than-linear signal function with $n = 4$. This case is effectively identical to that in Figure 2·7.

Figures 2·9 and 2·10 show sigmoid signal functions with $n = 2$ and $n = 4$, respectively. The results are similar to the other signal functions. The $n = 4$ case is slightly more interesting as the tuning process shifts the quenching threshold. Before tuning, an input of 0.4 falls below this threshold. This is why the output of node three is quiet on the first interval. After tuning, the quenching threshold shifts lower, such that 0.4 is above the threshold.

Figures 2·11 and 2·12 use the second-generation hRCF model. Figure 2·11 illustrates successful tuning of k with the second-generation model. As shown in Figure 2·4, the input distribution changes sharply at epochs 501 and 1001. Epoch 501 dramatically *decreases* the input range. Epoch 1001 dramatically *increases* the input range. A standard k-winner-take-all RCF with no homeostatic tuning responds by lowering the value of k for epochs 501 through 1000. The hRCF system, in contrast, is able to adjust its sensitivity such that the mean value of k returns to the target level of 2.5. Both systems are generally insensitive to the dramatically larger input range in epochs 1001 through 1500.

Figure 2·12 expands on the initial result in Figure 2·11 to more directly address

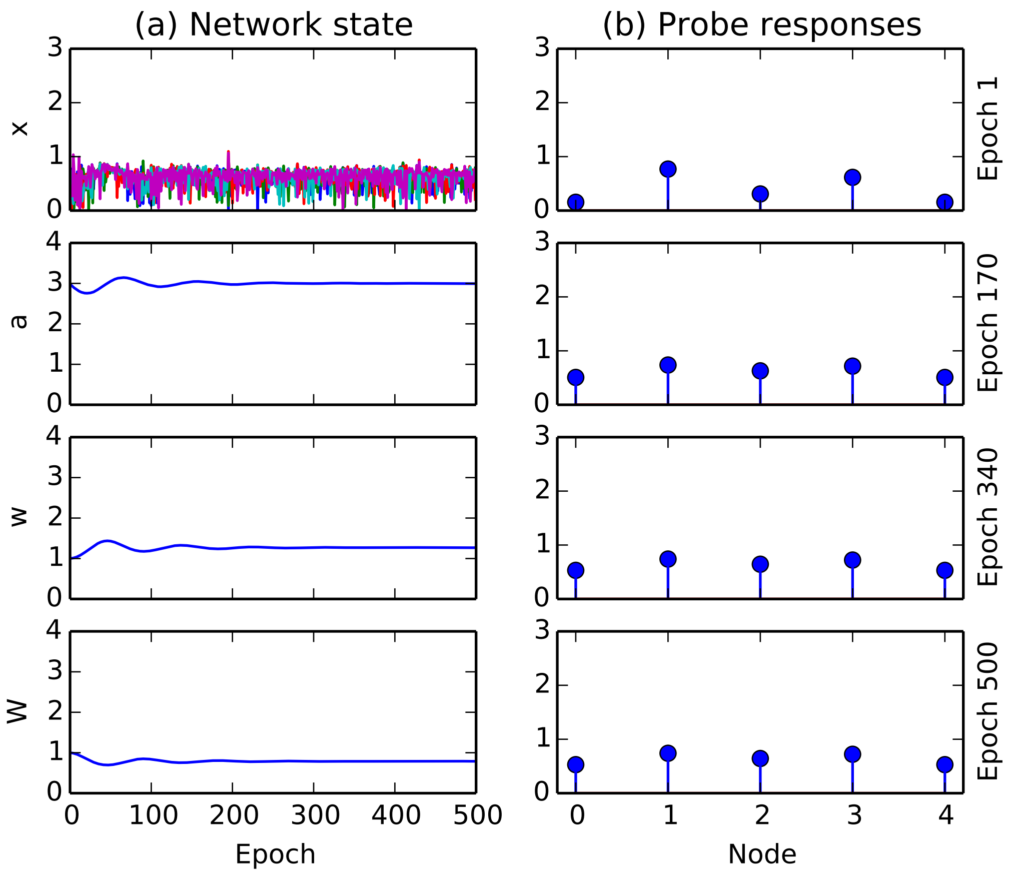


Figure 2·5: First-generation hRCF stability, linear signal function. The network state (a) indicates that the instantaneous activity x remains bounded over the 500 presentation intervals while the average activity level a converges to the goal level. The excitatory weight w and inhibitory weight W move in opposite directions to achieve this tuning. The network gradually loses pattern sensitivity during tuning, indicating that the network has maintained stability by sacrificing dynamic range. The pattern responses (b) after the first presentation interval is stronger than the responses after the 170th, 340th, and 500th presentation intervals. This reduction in sensitivity follows from the tuning of the excitatory/inhibitory balance and keeps the average activity level close to the goal level.

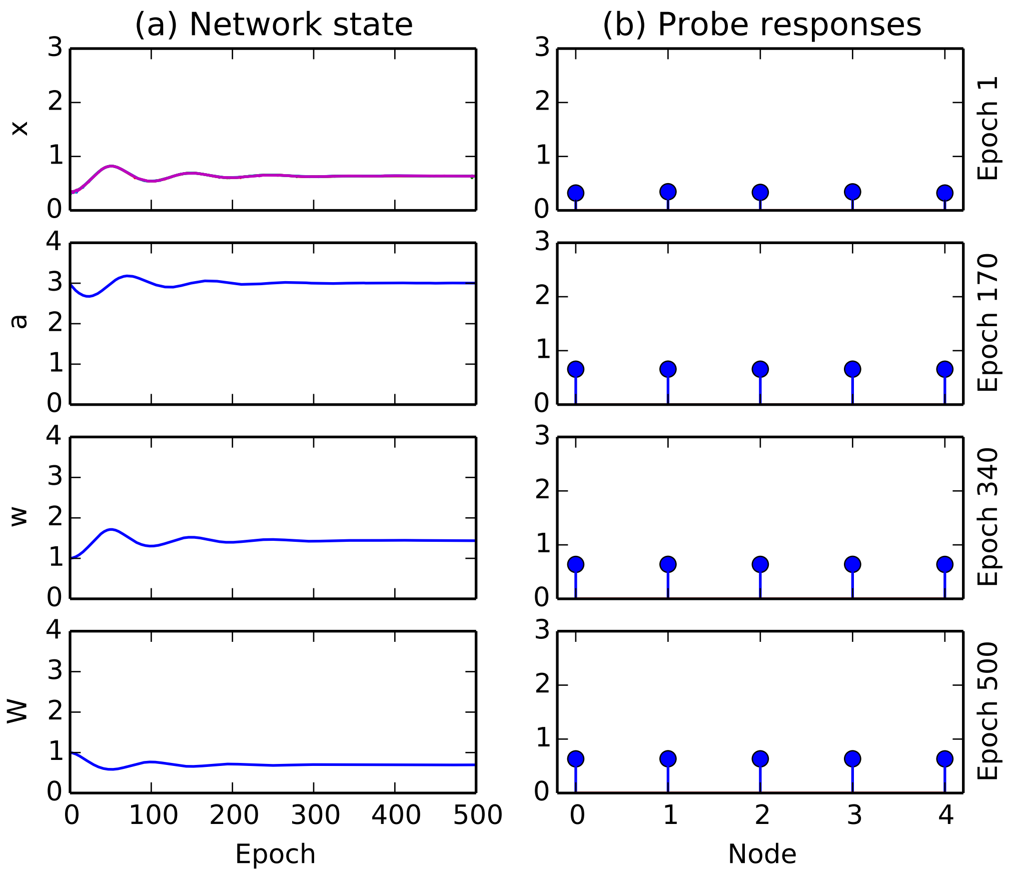


Figure 2·6: First-generation hRCF stability, slower-than-linear signal function. The tuning behavior (a) is identical to the linear case in Figure 2·5. The pattern processing behavior (b) is consistently homogenizing, however, consistent with the expected behavior for a standard recurrent competitive field.

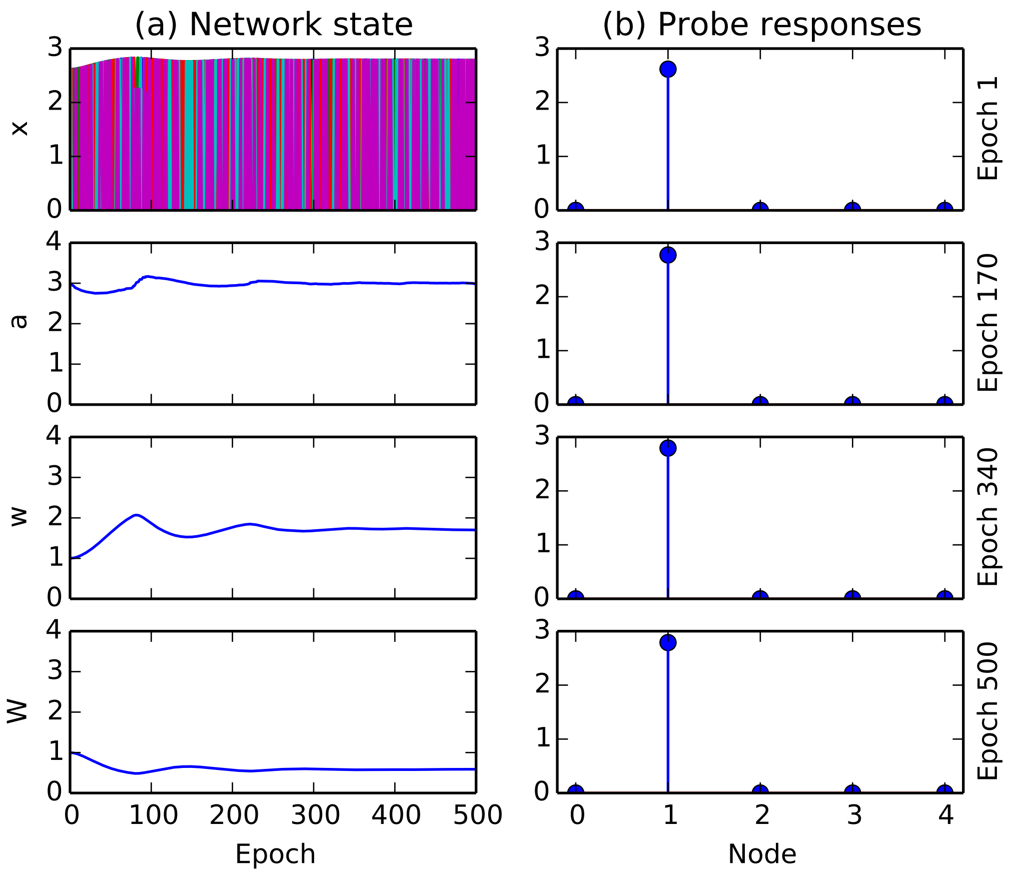


Figure 2·7: First-generation hRCF stability, faster-than-linear ($n = 2$) signal function. The tuning behavior (a) is identical to the linear case in Figure 2·5. The pattern processing behavior (b) is consistently winner-take-all, however, consistent with the expected behavior for a standard recurrent competitive field.

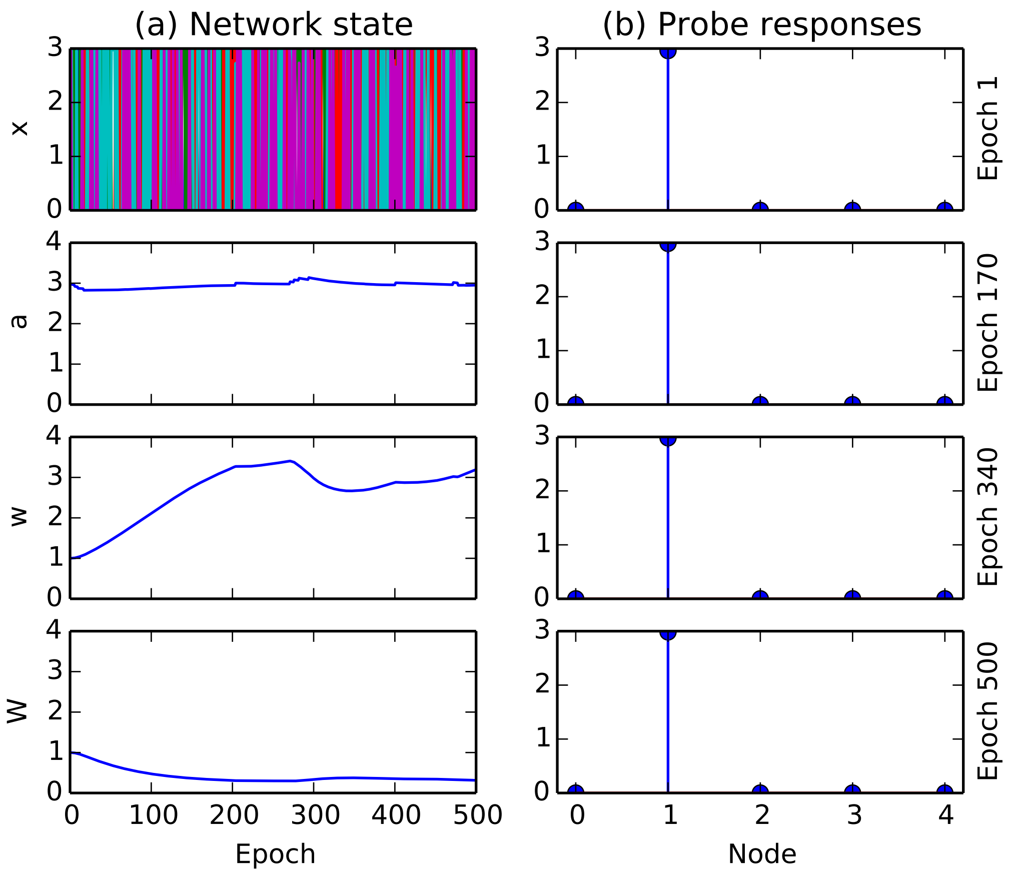


Figure 2·8: First-generation hRCF stability, faster-than-linear ($n = 4$) signal function. The tuning behavior (a) is identical to the linear case in Figure 2·5. The pattern processing behavior (b) is consistently winner-take-all, however, consistent with the expected behavior for a standard recurrent competitive field. Increasing the sharpness of the non-linearity from $n = 2$ in Figure 2·7 to $n = 4$ here has no effect on the pattern processing behavior.

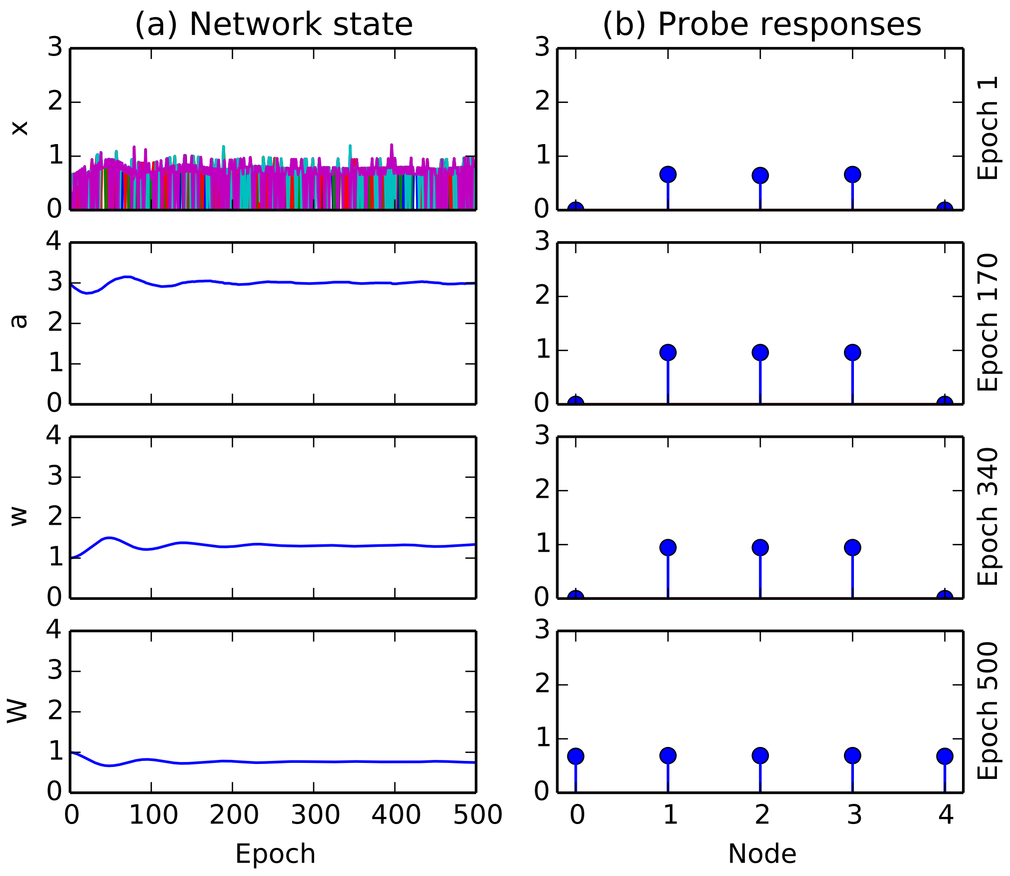


Figure 2·9: First-generation hRCF stability, sigmoid ($n = 2$) signal function. The tuning behavior (a) is identical to the linear case in Figure 2·5. The pattern processing behavior (b) is typically k-winner-take-all, however, consistent with the expected behavior for a standard recurrent competitive field. The network chooses three winners until the final epoch, where it homogenizes the input.

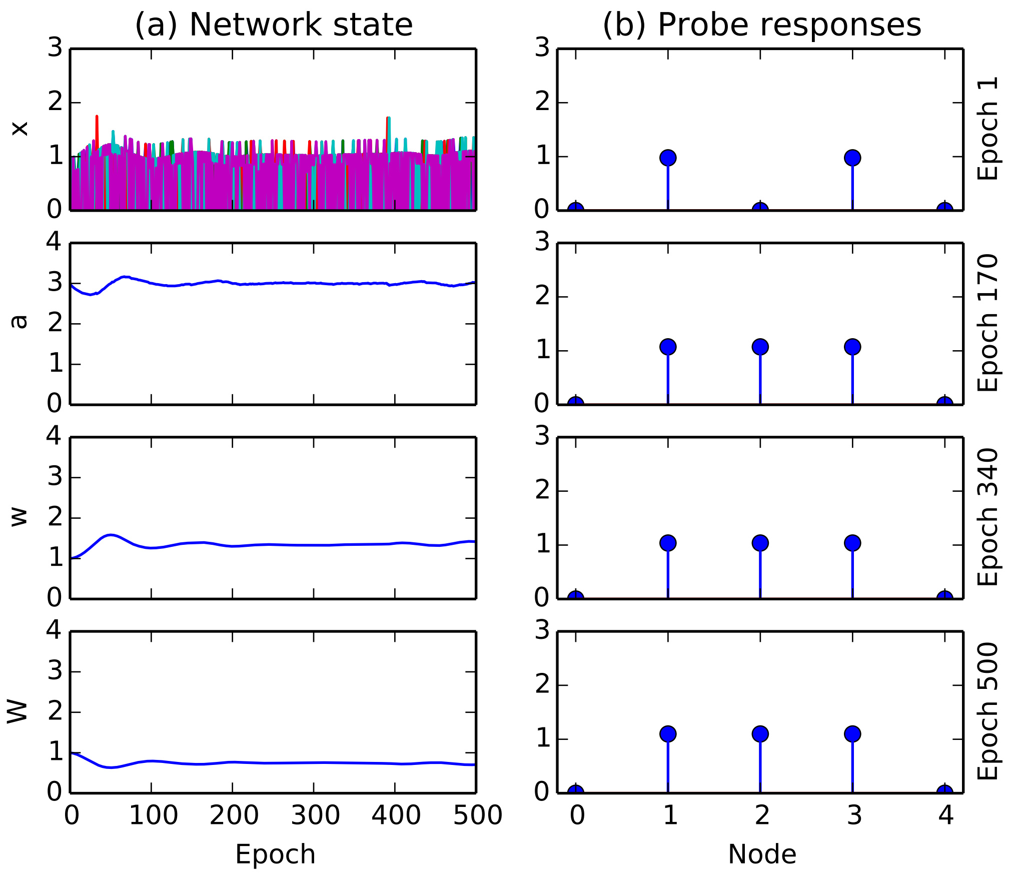


Figure 2·10: First-generation hRCF stability, sigmoid ($n = 4$) signal function. The tuning behavior (a) is identical to the linear case in Figure 2·5. The pattern processing behavior (b) is consistently k-winner-take-all, however, consistent with the expected behavior for a standard recurrent competitive field. Sharpening the non-linearity to $n = 4$ yields a system that maintains k-winner-take-all behavior over all 500 epochs. After the first presentation interval, the network chooses two winners. For the remaining three sample points, the network chooses three winners. Tuning k is a desirable behavior that keeps network activity close to the goal level without sacrificing pattern processing.

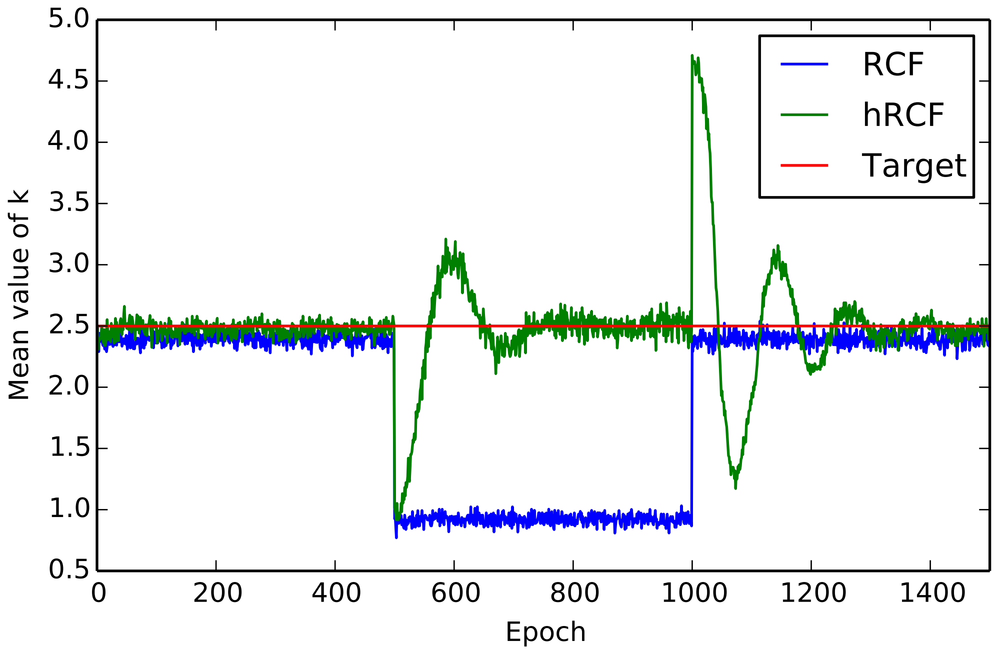


Figure 2·11: The hRCF model successfully tunes the number of active nodes. A standard k-winner-take-all recurrent competitive (RCF) model responds to the input protocol in Figure 2·4 with a highly variable value for k, or the number of active nodes. When the mean input intensity drops, the mean value of k drops. A homeostatic recurrent competitive field (hRCF) with a target value of 2.5 successfully adjusts its internal balance between excitation and inhibition to tune the value of k and compensate for the intensity of the input. The mean value of k oscillates, but converges to the target value within several hundred epochs.

the temporal noise-saturation dilemma. Each trace indicates the sensitivity of a specific network at a range of input magnitudes. The hRCF system at epoch 1000 has adjusted to very small inputs, and thus has a dramatically higher sensitivity than the remaining three networks. This pattern indicates a general *insensitivity* to high-magnitude inputs inherent to RCF-like models.

2.4 Discussion and conclusion

The homeostatic recurrent competitive field (hRCF) model is a partial success for resolving the temporal noise-saturation dilemma. By integrating a homeostatic plasticity process, the model is able to restore the desired response properties at small input ranges where a conventional RCF fails. The RCF foundation limits over-response, however. A system will under-respond to small inputs, but will never completely saturate even in response to inputs that are orders of magnitude larger than expected. The recurrent inhibitory weights in the system enforce normalization even under such extreme conditions.

As a model of adaptive behavior, this is an interesting result. A larger cognitive system composed of homeostatic RCF modules rather than standard Grossberg (1973) RCF modules would be able to bootstrap itself in response to small inputs. A system composed of standard RCF modules would convey very little information from module to module in this case and may fail to operate entirely. In practice, however, a simple approximation of the full hRCF model is likely sufficient. The output of the full second-generation model is approximately equivalent to a divisive normalization followed by a selection of the k highest values. This approximation would likely exhibit most of the desirable behavior at much lower computational cost.

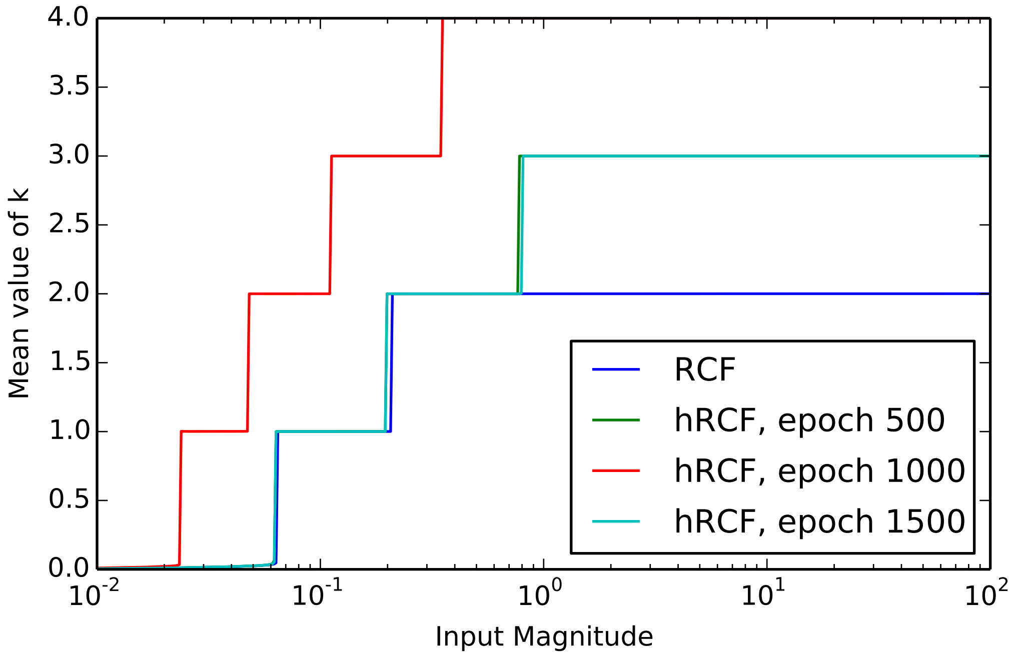


Figure 2·12: The hRCF model adjusts its sensitivity to compensate for the mean input intensity. Each trace represents the value of k exhibited by a single, fixed network when presented with a large number of input patterns of varying magnitude. A standard Grossberg (1973) RCF will choose one winner when presented with input of magnitude 10^{-1}, for instance. This is a measure of the sensitivity of the network. A standard RCF will choose two winners for inputs of a magnitude greater than approximately 0.5. Inputs of a smaller magnitude will cause the network to choose either one or zero winners. The homeostatic tuning in the hRCF model shifts these thresholds. The epoch-1000 hRCF system has adjusted to inputs in the range $[0, 0.1]$ and is thus substantially more sensitive. It chooses more winners in response to inputs of smaller magnitude than the baseline RCF system. The remaining two hRCF systems have adapted to input ranges of $[0, 10]$ and $[0, 100]$, respectively. These traces are nearly identical, indicating a general insensitivity to very large inputs inherent to RCF-like models.

Chapter 3

Testing and tuning cognitive computing algorithms

3.1 Introduction

Automated testing and parameter tuning are standard, important tools for building complex software systems. *Regression testing* detects changes to the system that break existing functionality as new functionality is added. The system developers implement a set of pass-or-fail tests, called a *test harness*, as they build the system. Running this test harness regularly allows the developers to detect unexpected failures as they continue to work on the system.

Where automated testing detects inadvertent backwards progress during the development of a complex software system, automated parameter tuning accelerates the rate of *forward* progress. Complex software systems tend to contain a number of free parameters that define the behavior of the system. In many cases, the system developers can either make an analytically correct choice or quickly explore all possible alternatives to find the best option. Automated tuning covers the cases where neither of these tactics are practical. A number of algorithms exist that can find optimal or near-optimal parameter values in dramatically less time than trying every option. The developer need only supply a metric to minimize or maximize.

Cognitive computing systems can benefit from both of these techniques. Any behavior for which a developer can write a pass-fail test can go into the test harness for the system. Any behavior that can be measured can be tuned. In practice, however, complex biologically-inspired algorithms are challenging to test and tune. This follows from the system architecture. Non-biological systems tend to contain a layered structure and weakly-coupled components with no cyclic dependencies. These architectural features contain complexity to individual components and allow for clean abstraction. Such an architecture is simple to design and simple to test. Biologically-inspired systems generally violate these architectural principles. Strongly-interacting components and weak layering are common.

The practical consequence of strongly-interacting components and weak layering is *unconstrained complexity*. This means testing any individual component or behavior of the system generally requires running the *entire system*. Testing or tuning an individual component of the system in isolation is typically not possible. This increases the magnitude of computational resources necessary to test or tune the system.

This chapter presents two new software systems designed to address the high demand for computational resources. The first system automates parameter exploration, distributing the compute load over a cluster of worker machines. The second system provides infrastructure for defining visual tests and automates test harness execution.

3.2 A system for cluster-enabled parameter tuning

The Iterative Evolution of Models, or ItEM, system is an attempt to accelerate neural model development through *scalable automation* of *reproducible research*. The ItEM architecture is structured around these design goals. Scalable automation is the principle driving force for ItEM; the system exists to automate as much of the neural modeling workflow as possible. Success is defined by increased modeling throughput, measured by the rate of adding desirable behaviors to the model. A clustered architecture enables ItEM to scale to however many compute nodes are available.

Reproducibility is a secondary design goal. One of the critical bottlenecks in large-scale neural modeling is an inability to quickly replicate and build on previous results. ItEM addresses this problem by linking simulation data to descriptive metadata indicating what version of the code was used to generate that data. Data and associated metadata cannot be deleted. In addition, ItEM only operates on code that has been checked in to a version control repository. This repository does not allow users to modify or delete the revision history. All of these principles could be achieved by disciplined manual management of simulation code and results, but at the cost of significant human effort. ItEM automates management policies and moves them server-side, so users benefit from a well-maintained repository of code and data without any manual effort.

3.2.1 System design

System architecture

As shown in Figure 3·1, ItEM is structured around a central daemon process. This daemon is responsible for maintaining system state, managing the data archive, communicating with clients, and scheduling work. Long or computationally expensive units of work are queued for one of the three pools of workers to handle.

Work is structured in an *asynchronous pull* model wherever possible. This means that whenever a task is too big or too complex to complete immediately, that task is placed in a central queue. Each compute node has a single *worker process* responsible for executing tasks on that node. Each worker process fetches a task from the queue, completes the task, reports back progress, and repeats. *Asynchronous* means that the workers complete jobs at their own rate with no inter-worker dependencies. *Pull* means that the workers fetch jobs from the queue as quickly as they can rather than waiting for commands.

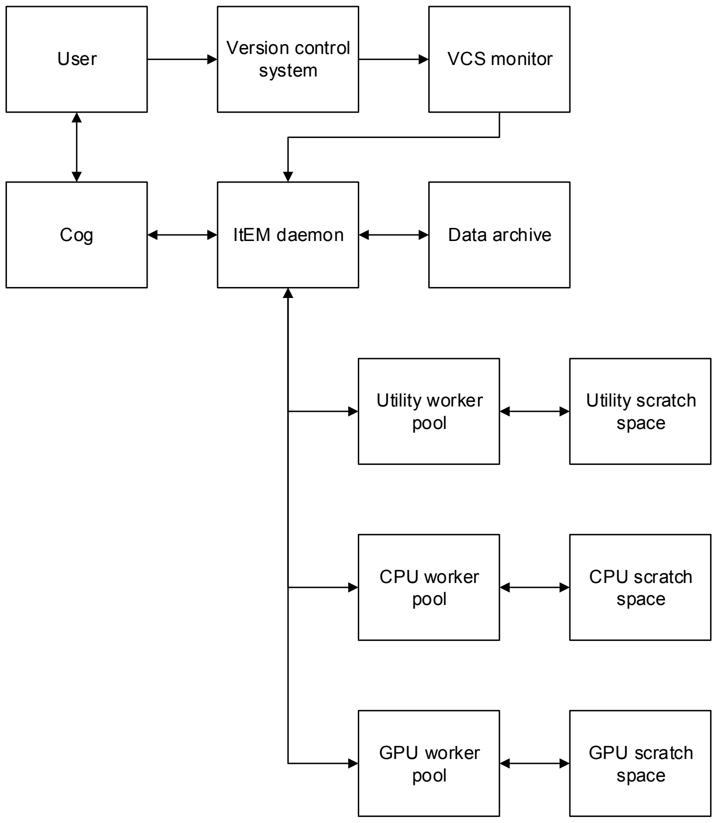


Figure 3·1: Architecture diagram for automated parameter tuning system. The Iterative Evolution of Models (ItEM) system automates much of the manual work necessary to correctly set the parameters for a complex cognitive computing algorithm. Users interact with the system by extending their Cog ex Machina (Cog) programs with ItEM parameter annotations. They then check their code in to a shared version control system (VCS). A VCS monitor script reports the change to the central ItEM daemon. The ItEM daemon schedules work to an appropriate pool of worker processes on remote compute nodes. Utility workers handle long-running work with low compute requirements. CPU jobs are for data processing. Graphics processor (GPU) nodes handle simulations. Each type of compute node has local scratch space to store temporary files. The ItEM daemon collects results, archives data to a central repository, and makes the data available to users.

These two attributes enable simple scalability. As the workload grows, more workers can be added quickly with zero code changes. For workload bursts, a pull model also enables full utilization of temporarily allocated resources. The workers only need to know how to communicate with the central daemon. The daemon is responsible for any flow-control logic or dependent jobs.

Localizing the system complexity to the daemon dramatically simplifies the software compared to a fully distributed process model, but carries the risk of a performance bottleneck. The computational throughput of the entire system is limited by the rate that the daemon can process messages from clients and the workers. In this arrangement, however, the bottlenecks are significantly easier to localize and address. In a distributed process model, bottlenecks are generally emergent system properties requiring changes in multiple locations to fix. A central daemon implies a central bottleneck, which can be resolved either by placing the daemon on a faster machine or improving the code efficiency in that single program. The daemon already distributes long-running or expensive computations to workers, so a single daemon running on a single machine is unlikely to limit the capacity of the system.

All communication with the daemon takes place using a custom binary protocol. Messages use the Protocol Buffer format, developed by Google as a standard mechanism for data encoding (Dean and Ghemawat, 2010). The Protocol Buffer compiler automatically generates all the interface code necessary to serialize and de-serialize binary data from a simple message format specification. This dramatically reduces the amount of effort needed to construct an efficient binary protocol. Protocol Buffers are vastly more efficient than text formats, so the same technology is also useful as an on-disk format for data analysis and archiving.

Capabilities

The system is fully integrated with the Cog ex Machina platform (Snider et al., 2011). Modifying an existing Cog model to support batch execution through ItEM requires as little as a few additional lines of code. Users need only specify a termination criteria for the batch simulation and specify what data should be logged. The ItEM framework extracts data from simulations using the state probing mechanism built in to Cog, so data logging is as simple as attaching probes and notifying ItEM when the data from a given probe should be written to disk. The ItEM framework automatically serializes this data to disk in a format ready for archiving or analysis and uploads the log back to the central system.

The ItEM framework carries the additional benefit of enabling intelligent re-running of batch simulations as the code changes. Each time changes are committed to the central code repository, ItEM automatically recompiles the source tree and queues batch jobs. Only batch simulation definitions that have changed are recompiled and re-simulated. This dramatically cuts down on redundant simulations.

To perform simulations or other resource-intensive computations, the system supports remote workers of three different classes. These classes, Utility, CPU, and GPU, each require a different resource allocation policy. Utility workers are used for tasks like managing dependent jobs. This type of work requires exclusive access to a private copy of the source tree, but is not compute-intensive. A utility worker spends most of its time waiting for new datasets to come back from other workers. This means many utility workers can be placed on a single conventional cluster node.

CPU and GPU workers are used for compute-heavy tasks. These workers fetch a unit of work from the appropriate queue, execute it, and repeat. They do not wait for results from other parts of the system as the utility workers do. CPU workers are used for tasks like data analysis that do not require GPU acceleration. They are placed one worker per processor in a conventional CPU-only cluster. GPU workers function similarly, but are used for Cog simulations requiring GPU acceleration. These workers are placed one per node in a GPU-enabled cluster.

Simulation definitions need not be fully specified. Parameters may be left as ranges, rather than discrete values. The system is capable of generating a fixed-length list of fully-specified models from a grid placed across the parameter space. This is an "open-loop" or grid search. The utility worker responsible for a given simulation definition loads the appropriate code, queues the appropriate number of jobs, and exits immediately. The GPU workers run a simulation at each point in the parameter space and log the results back to the central system.

3.3 A system for testing visual algorithms

Manual tests are generally less valuable for detecting behavioral regressions. Such tests cost more time and effort to run, and therefore cannot be run as frequently. Visual tests are typically in this category. To validate behavior, an investigator has to run the application, wait for results, and visually inspect those results for correctness. A human is involved every time such a test is run. In practice, this burden dramatically reduces the frequency and depth of testing for visual algorithms.

Solving the manual-testing dilemma for visual algorithms requires *automated* tests be as easy as possible to write. Visual algorithms are difficult to verify with automatic tests using standard testing frameworks. This follows from the data-intensive nature of visual tests. The expected result for a boolean or scalar-valued test can be written as a single constant. Verifying that a block of code correctly implements the function $f(x) = x^2$, for instance, would only require a single scalar input and a single scalar output per test case. Such test cases are terse and simple to include in-line with the testing code. A visual test, in contrast, typically operates on entire *images*. This is too much data to embed in the source code for the test. To create such a test, the investigator would need to manually run the test code, visually inspect the output, store that output in a file along with the test harness, and modify the test to check its output against the stored file on each subsequent run.

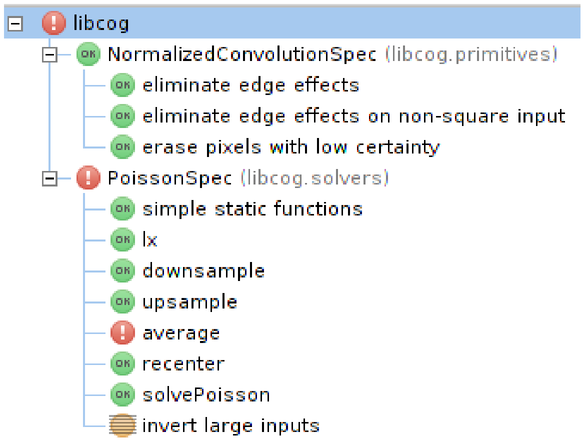
The visual testing system is aimed at automating this workflow. Currently, such test cases are too hard to write to be useful in practice. With the new system, the human is able to visually inspect the test output through a standard interface and all of the plumbing necessary to check test output against previous outputs is automated. If the code under test produces an output that is approximately equal to one a human has already checked, the system will pass or fail the test automatically. This prevents a human from having to *re-check outputs that have not changed.* Figure 3·2 shows the output of a sample test harness for a library of visual algorithms.

3.3.1 System design

As shown in Figure 3·3, the visual testing system adopts a similar architecture to ItEM. Like ItEM, the primary interaction point with users is the version control system. Users push their changes to a version control system, where those changes are picked up for processing. Also like ItEM, a central management process orchestrates the flow of work using a pool of compute nodes.

The visual testing system differs from ItEM primarily in the degree to which it incorporates existing software systems. ItEM uses very little third-party software. The version control system is an open-source component. Cog ex Machina is a separate project. All other components of ItEM are custom and specific to ItEM. The visual testing system, in contrast, is primarily composed of existing software with appropriate custom extensions designed for visual algorithms.

Like ItEM, the visual testing system uses an open-source version control system. Instead of a custom master process, it uses the TeamCity continuous integration



```
⊟ libcog
  ⊟ NormalizedConvolutionSpec (libcog.primitives)
      eliminate edge effects
      eliminate edge effects on non-square input
      erase pixels with low certainty
  ⊟ PoissonSpec (libcog.solvers)
      simple static functions
      lx
      downsample
      upsample
      average
      recenter
      solvePoisson
      invert large inputs
```

Figure 3·2: Sample test report for a library of visual algorithms. The implementation of normalized convolution exercised by the Normalized-ConvolutionSpec test suite passed all of its tests, as indicated by the green "OK" icon next to the test suite and each test contained in the suite. The PoissonSpec suite failed one behavioral test ("average"), indicated by the red error icon. PoissonSpec also contains one test without a complete implementation ("invert large inputs"), indicated by the yellow icon. Without a complete test implementation, the test runner cannot determine if the test passed or failed and thus marks it as pending. The test runner generated this report in under a minute on a standard workstation and with no human intervention.

system (JetBrains, 2014). TeamCity already includes functionality for monitoring a version control system, building the code at every change, and running regression tests in a number of formats, all across a pool of compute nodes. The visual testing system extends this functionality through a set of custom testing functions and the necessary database back-end.

To write a visual test, users structure their test code in the same way they would for a non-visual test. Each test case has an expected output. A typical test case would include the expected output in-line with the test code. For visual tests, users instead supply a unique string that points to a set of exemplars stored in the database. When the test is run, the visual testing framework queries the database for a matching output that has already been reviewed by a human. If such an output is found, the framework marks the test as passed or failed depending on the value stored in the database. If no matching output is found, the framework stores the output to the database and marks the test as pending.

Users interact with the visual exemplars through a custom web application. This web application offers a view of all exemplars that have already been classified, as well as a view of exemplars that still require review. Users can classify exemplars as passing or failing, or expunge exemplars that are no longer useful. All changes are stored back to the database and reflected on the next run of the test harness.

3.4 Results

A sample grid search produced by ItEM is shown in Figure 3·4. The ItEM infrastructure completed 62 simulations across a cluster of five compute nodes to generate this plot. The cluster reduced the time required to complete the necessary simulations to only several hours. Automating the grid search eliminated nearly all of the manual effort necessary to generate the plot.

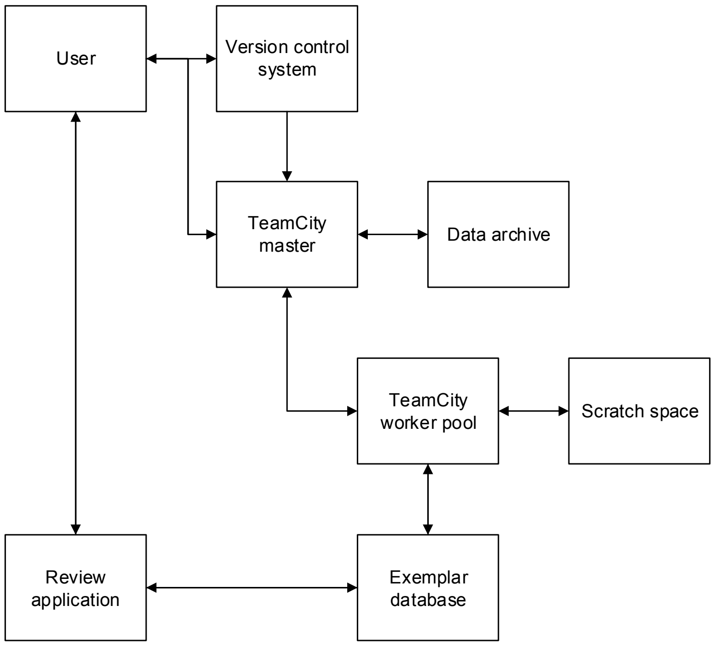


Figure 3·3: Architecture diagram for automated regression testing system. The regression testing system adopts a similar architecture as the ItEM design in Figure 3·1. The commercial TeamCity continuous integration system replaces many of the custom components, however. Users still engage the system by extending their existing programs with appropriate annotations and checking the code in to a version control repository. The open-source ScalaTest library provides these annotations. TeamCity manages compilation and testing on the worker nodes and performs any necessary data movement between the worker-local scratch storage and main data archive. A custom extension to the ScalaTest library pushes visual test output to an exemplar database. Users review these outputs with a web application. TeamCity provides a web interface where users can monitor their compilation and testing jobs, as well as review the testing history.

The principle limiting factor for ItEM is the lack of re-use of existing software systems. ItEM is nearly all custom software. This means the system is tightly tuned for the problems it was intended to address. The high degree of custom software, however, means that many aspects of the system are rough, unreliable, or hard to use.

The cluster interface, for instance, is still manual. While the system supports local utility processes running locally, CPU-only workers for data analysis, and GPU-enabled workers for simulation, it cannot launch workers itself. An administrator must manually launch or kill workers depending on the volume of work. While suitable for tightly supervised one-off runs, manual worker allocation and de-allocation requires too much manual effort to become part of a regular modeling workflow.

The ItEM system is also unreliable. It has many interlocking components, potentially running across dozens of machines. Such distributed systems are difficult to implement and even more difficult to make reliable. Many failure modes are rare and only detected after adding extensive tracing support and accumulating many operating hours. ItEM has not accumulated enough operating hours to flush out the failure modes, and does not include sufficient introspection support to adequately debug rare failure modes. More critically, data persistence is still poorly supported. If the master process crashes, all in-flight jobs and data will be lost.

There is essentially no human-friendly interface in the current ItEM system. The only way to monitor progress with the current code is to watch the logs for the central server. While infrastructure is present to support a user interface, more development time is needed to build one out.

The visual testing infrastructure avoids many of the limitations of ItEM by heavily re-using existing code where appropriate. TeamCity, for instance, is a stable, well-supported core, and provides an easy-to-use web interface for monitoring the progress

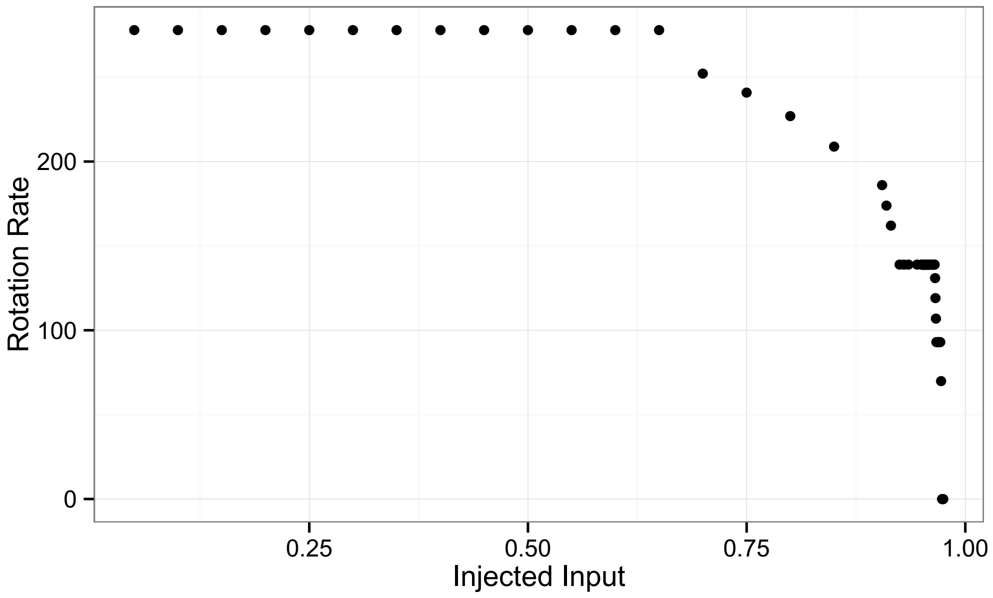


Figure 3·4: Example result from automated hyperparameter tuning system. A *ring attractor* is a type of neural population that maintains a burst of rotating activity. This burst moves around the ring of cells at a rate modulated by the injected input (Fortenberry et al., 2012). To use a ring attractor as a component of a larger neural model, the designer must choose an input range that produces a desirable response in the rotation rate. Each point in the plot represents one simulation of the ring attractor, so fully characterizing the response function requires a large number of simulations. The Iterative Evolution of Models (ItEM) system was able to dramatically reduce the time and manual effort needed to characterize the response function by automatically distributing a batch of simulations across five worker nodes.

of jobs and reviewing historical test results. The system is less optimized for visual tests, but it runs reliably without significant manual monitoring or tweaking.

The database is the weakest aspect of the visual tuning infrastructure. Best-practices for automated software testing dictate that all of the state a test depends on ought to be stored alongside that test. A user, for instance, ought to be able to back the state of the project up to an arbitrary historical revision, run the tests, and get identical output to when that revision was first created. This property is valuable when tracing defects in time or maintaining old releases of the code for external users. By storing test exemplars in an external database, the visual testing infrastructure breaks the expectation that historical tests will always return identical results.

3.5 Conclusion

ItEM is capable of tackling real scientific problems. Tight Cog integration means that existing modeling questions can be quickly adapted for batch simulation using ItEM. As the system accumulates production operating hours and an increasing number of failure modes are detected and resolved, ItEM has the potential to evolve into a critical enabling technology. In its current state, however, ItEM is still too fragile and labor-intensive for practical use.

The robust core is the most important aspect of the visual testing system in practice. Modifying Cog to run on top of this core added significant day-to-day value. The custom extensions for testing visual algorithms, however, did not survive in production. While these extensions were useful and covered a use case that a regular testing framework did not support well, the database dependency added significant unnecessary complexity.

A more reasonable design would keep the same workflow for human review of exemplars, but store data directly in the version control repository. Replacing the web application with a light-weight local user interface would accelerate the testing workflow while keeping the test exemplars directly in the version control repository.

The general lesson from both systems is to re-use existing infrastructure whenever possible and emphasize simplicity. ItEM is at present too complex and fragile to use in practice without expert supervision, so the ratio of effort to reward is poor. Hand-tuning the hyperparameters is good enough for most cases, even in hyperparameter-heavy cognitive computing algorithms.

The visual testing system demonstrated that building a fully custom system is usually not necessary. Standard testing libraries are not perfect for cognitive computing applications. Tests for data-heavy code are verbose to write. The standard open-source tools are reliable and easy to use, however, which make them much more valuable in practice. A few custom testing extensions and the right open-source infrastructure produced significantly more value than the fully custom ItEM infrastructure.

Chapter 4

Robust classification of occluded objects

4.1 Introduction

Objects that are heavily occluded are more difficult for a classification algorithm than unobstructed objects. Neither of the current high-profile object classification benchmarks is well-suited to quantify this degree of difficulty as a function of increasing occlusion, however (Socher, 2009; Everingham et al., 2009). Both were constructed from public photo-sharing sites and thus have a bias against occluded images. Hoiem et al. (2012) used results from the 2007 PASCAL VOC classification task to identify the most significant sources of error. Classification performance decreased precipitously as the percentage of occluded pixels increased. The infrequent occurrence of occlusion in the PASCAL data, however, caused the authors to conclude that occlusion resilience has a negligible impact on the total PASCAL performance of an algorithm (Hoiem et al., 2012).

This assertion follows from the source of the data. Humans captured the images in PASCAL by deliberately composing and capturing scenes, and as such *tended to take photographs of unobstructed objects close to the center of the frame.* This tendency towards fully visible objects in PASCAL is an example of observer bias in dataset construction (Torralba and Efros, 2011). Dataset bias in general, of which observer bias is one subclass, allows algorithms to produce impressive-looking classification results by over-relying on signals that have a little real-world utility. A specific dataset, for instance, might contain objects in only a very restricted set of poses or image backgrounds that strongly correlate with the class of the target object (Pinto et al., 2008; Torralba and Efros, 2011). This bias is difficult to measure and difficult to correct, weakening the argument that the current large classification datasets accurately represent the real world (Torralba and Efros, 2011).

Besides representing a biased sample of the real world, the current large-scale datasets are also missing much of the descriptive metadata needed to accurately quantify occlusion robustness. An ideal dataset would include a precise description of the type and level of occlusion as metadata with each sample. Hoiem et al. (2012) were only able to consider four different cases for occlusion: none, low, medium, and high. The study defined low as "slight occlusion," medium as "significant part is occluded," and high as "many parts missing or 75% occluded." The coarse definitions of occlusion followed from the nature of the data. Everingham et al. (2009) constructed PASCAL by aggregating and hand-annotating a large collection of images from the Flickr photo-hosting website. Hoiem et al. (2012) enhanced the PASCAL annotations with additional detail on level of occlusion. As in Everingham et al. (2009), they had to do this by hand. This is a standard problem with non-parametric classification datasets. Such datasets are collected from unlabeled, unconstrained image sources. The data is thus very diverse, but annotations have to be added by hand and tend to be information-poor as a practical consequence (Everingham et al., 2009; Socher, 2009).

The NORB dataset is a contrasting example of a recent parametric dataset (LeCun and Bottou, 2004). NORB contains five image classes with ten objects in each. For each object, the dataset contains a stereo image pair for each of 36 azimuths, 9 elevations, and 6 lighting conditions. These parameters are carried along with the image data, allowing investigators to directly measure the sensitivity of their algorithms to properties like intra-class variation or lighting. Figures 4·1a and 4·1b contrast parametric and non-parametric types of classification datasets.

Performing a more sophisticated analysis of occlusion robustness than Hoiem et al. (2012) were able to execute requires better descriptive metadata. This chapter introduces the SORBO (Synthetic Object Recognition Benchmark with Occlusion) dataset to capture class and level of occlusion with much finer resolution than was possible by hand-annotating PASCAL data, and to support work in occlusion-resistant object classification. The design matches NORB as much as possible to simplify comparison with the many NORB results in the literature. Figure 4·1c shows example SORBO images with varying classes and levels of occlusion.

The ability to precisely measure the impact of occlusion on object classification performance enables the construction of an occlusion-robust algorithm. Such an algorithm must confront two issues that non-robust algorithms are able to ignore:

1. What visual information belongs to the target object?

2. How should uncertain or non-target information be discounted?

The first problem is one of *segmentation*. An occlusion-robust classification algorithm must be able to classify each pixel of visual information as *figure*,*ground*, or *occlusion*. The figure class identifies pixels that belong to the target object. Ground pixels are either behind or do not overlap the target. Occlusions are pixels in which the object of interest is hidden because it is obstructed by an object nearer to the observer. These last two classes are distinct because occlusions cover the target and may split the figure information in to two or more non-adjacent regions. The ground is either behind or non-overlapping with the figure and thus cannot cause such splits. Once the algorithm estimates which information belongs to each class, it must then discount uncertain or non-target information. Uncertainty arises because the segmentation of the scene is a potentially inaccurate estimate.

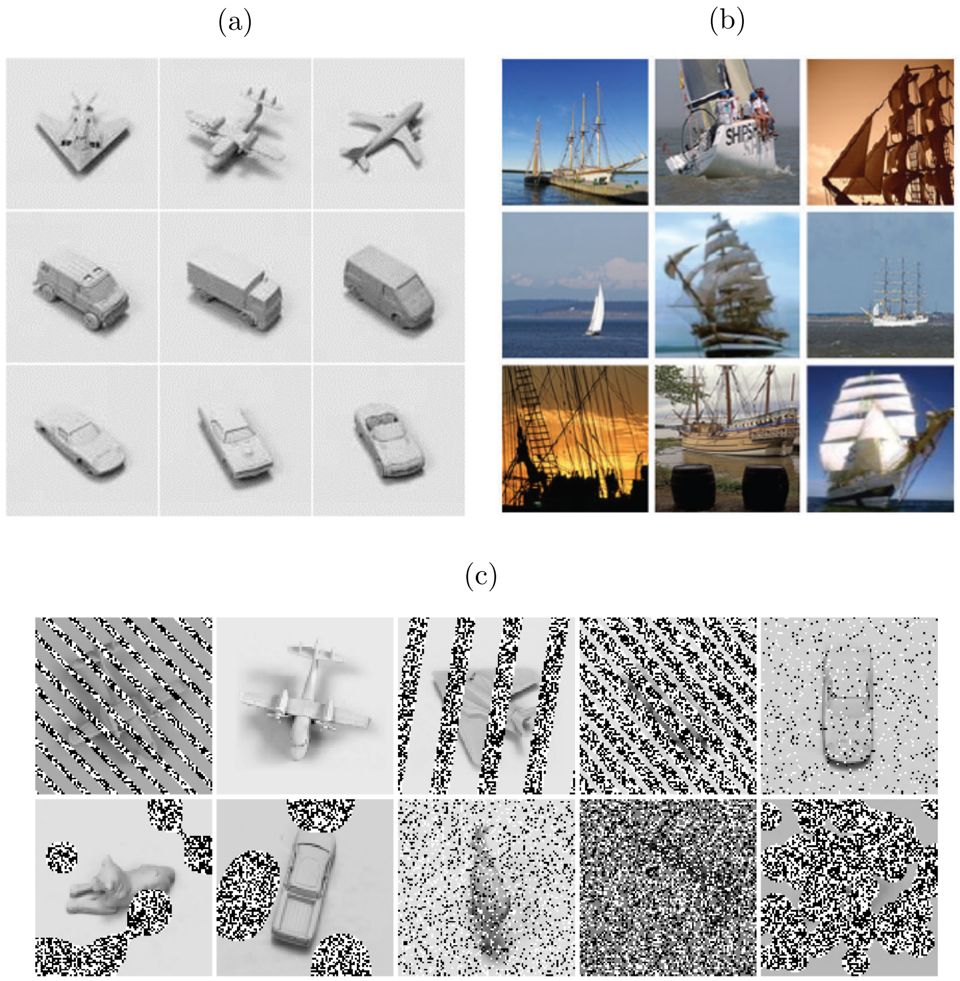


Figure 4·1: Sample images from NORB, ImageNet, and SORBO datasets. NORB (a) is a parametric dataset designed for experiments in invariant object classification. It includes five categories of objects, each with ten specific objects. NORB contains a stereo pair at nine camera elevations, eighteen camera azimuths, and five light levels for each instance (LeCun and Bottou, 2004). ImageNet (b) is a non-parametric dataset containing a large number of labeled examples scraped from the Internet. In comparison to NORB, ImageNet has much more data and many more categories, but typically only a single image for each object instance. The image parameters are unknown except for the category (Socher, 2009). SORBO (c) is a new dataset that extends the NORB data. It preserves the rich parametric metadata from NORB, but adds various levels of bar, blob, and random stereo occlusions.

Segmentation and discounting are relevant tasks even for a classification algorithm, as opposed to a *detection* algorithm. In a detection problem the algorithm must estimate the location of a target object given an input image that may contain a target object at an arbitrary position in the frame. A subset of input images may contain no target objects. A *classification* problem is simpler to solve in that it enforces viewpoint bias by restricting the input to images with a single dominant object class close to the center of the frame. The Hoiem et al. (2012) study indicates that occlusion is a significant issue even for classification problems.

The analysis in Hoiem et al. (2012) was sufficient to establish that existing state-of-the-art algorithms do not perform well in high-occlusion scenarios. It also established that the PASCAL VOC dataset is the wrong benchmark to use if high-occlusion scenarios are important. PASCAL VOC contains relatively few occluded objects, so the aggregate results are relatively insensitive to how well an algorithm handles occlusion. For the objects that *are* occluded, the locations of the occluding pixels are unknown. This lack of ground-truth segmentation information further limits the applicability of the PASCAL VOC data to the development of occlusion-resistant object classification algorithms. SORBO includes this additional metadata.

This chapter introduces a new occlusion-robust object classification algorithm that leverages the rich metadata in the SORBO dataset. This new algorithm is an *extension* of an existing algorithm rather than an entirely custom construction. Extending an existing algorithm carries the primary benefit of preserving the performance of the underlying classifier in low-occlusion cases. A state-of-the-art base classifier will continue to perform as intended on standard datasets like NORB. The new extensions allow the classifier to degrade in performance more slowly as occlusion increases.

Taken together, parametric occlusion metadata and ground-truth segmentation information make SORBO an ideal dataset for exploring occlusion-resistant object classification. This chapter first focuses on replicating the Hoiem et al. (2012) study using the new SORBO dataset. Follow-on experiments build on these results to construct an occlusion-resistant object classification algorithm from an existing non-resistant algorithm.

4.1.1 Existing object classification datasets

The majority of current object classification work is organized around two standard challenge datasets: the ImageNet Large Scale Visual Recognition Challenge (ImageNet) and the PASCAL Visual Object Classes Challenge (PASCAL VOC or PASCAL). Each challenge has an associated dataset. While the two challenges use different data, different tasks, and different accuracy metrics, both are *large-scale* and *non-parametric*.

In the context of object classification, a non-parametric dataset is one in which the images are sampled from a population with *unknown* image parameters. The term does not imply the the images cannot be described by a parameter space. It only implies that the investigators cannot practically enumerate the image parameters. Typically this is because the the images are drawn from a very large, weakly constrained population with a very large parameter space. Both ImageNet and PASCAL were constructed using data from public photo-sharing web sites. The population is thus photographs from the Internet with sufficient contextual information to assign an approximate category. The initial image category labels in both ImageNet and PASCAL came from an image search engine. Humans cleaned up the labels and removed inaccurate examples.

The principle difference between the PASCAL VOC and ImageNet is the number and structure of the object categories. PASCAL contains four macro categories (person, animal, vehicle, indoor) with a total of twenty subcategories. This relatively flat category structure and low total category count is consistent with most prior object classification datasets. ImageNet differs in that it contains thousands of categories arranged in a deep, hierarchical structure. The TinyImages dataset from MIT uses a similar structure, but only includes very small images (Torralba et al., 2008). Both ImageNet and TinyImages use category labels from the noun portion of the WordNet database (Miller, 1995). WordNet includes superordinate and subordinate terms for each term in the database. Given the label "German shepherd," for instance, the superordinate label list is {shepherd dog, working dog, dog, canine, carnivore, placental mammal, vertebrate, chordate, animal}. Each step up the tree of superordinate labels represents an is-a relationship. Algorithms designed for a small, flat hierarchy of category labels tend not to do well with the large, deep, hierarchical tree of category labels in the ImageNet and TinyImages datasets without modification. All three datasets (PASCAL, ImageNet, TinyImages) are still non-parametric, however.

The NYU Object Recognition Benchmark (NORB) dataset is a recent example of a *parametric* object classification dataset. ImageNet and PASCAL VOC are both considered *non-parametric* because they consist of samples drawn from a large population with unknown image parameters. The ImageNet samples in Figure 4·1b illustrate this point. The images are all photographs of ships. No ship appears more than once, however, and parameters like the image composition and pose vector for the target object are all unknown. More critically, there are additional, uncontrolled parameters like the time of day, wind, and background. Simply describing all the sources of parametric variation in the images is impractical due to the size of the parameter space and the fact that each image would require manual labeling.

This large parameter space is typically taken as an argument that the images are unbiased when compared to the real world. By this logic, object classification algorithms that work well on the ImageNet or PASCAL VOC datasets ought to perform well on arbitrary images collected from the natural world. In practice, however, this argument doesn't hold up well. Hoiem et al. (2012) made a critical observation along these lines: occlusions don't appear very frequently in the PASCAL VOC data. This observation about PASCAL contradicts common experience. Humans and other visual creatures are quite adept at recognizing occluded objects and navigating through occluded environments because such conditions are normal, not exceptional. This is an example of *observer bias*. The PASCAL VOC and ImageNet datasets are drawn from a population of human-composed and human-captured photographs, not the population of all natural images. This means that there is a bias towards *unoccluded objects that appear close to the center of the frame*. Other authors have noted additional sources of bias, such as predictive statistics from the image background (Pinto et al., 2008; Torralba and Efros, 2011). Such contextual information typically helps prime a classifier towards the correct class of an object. When developing a classification algorithm, however, contextual priming acts as a confounding variable.

The NORB dataset takes a more restrictive approach. Rather than sampling from a large image population with unknown parameters and poorly understood bias, NORB is defined by an explicit parameter space. The point in the parameter space describes the image completely, such that the image could be reconstructed perfectly given knowledge of the parameters. Such an approach is less "natural" in that the images are heavily constrained and look less like scenes from the real world. This follows from the fact that the set of parameters is vastly smaller than the set of parameters needed to describe natural images. The benefit, however, is that the creator of the dataset gets to *choose* which parameters to include. This means that each exemplar in the dataset includes precise, complete, parametric metadata for the design parameters. Such metadata is a necessary condition for constructing experiments to analyze the conditions under which a classification algorithm does well or poorly.

NORB was designed to tackle pose and lighting-invariant object classification. This goal lead to the choice of design parameters:

1. object class

2. object instance

3. camera elevation

4. camera azimuth

5. lighting condition

The *object class* is the category to which the object belongs. NORB contains five categories: four-legged animals, human figures, airplanes, trucks, and cars. The *object instance* is the specific object within the category. Each category contains ten unique instances, for a total of fifty objects. The *camera elevation* and *camera azimuth* describe the position of the cameras relative to the target object. NORB contains images captured from nine elevations and eighteen azimuths. Finally, the *light level* describes the degree of illumination. NORB contains images shot at five different light levels.

These five parameters describe the desired dimensions of variation in the NORB images. For all other possible image properties, the authors worked to carefully control variation. The target objects were painted a uniform color and photographed on a uniform background. The camera and lights were all positioned by robot. The object and cameras were carefully positioned such that the object appears in the center of the frame for both. In practice, this means the five desired sources of parametric variation describe essentially all of the variability in the images.

The tightly controlled variability and high-quality parametric metadata in NORB enables a set of experiments that are not practical with non-parametric datasets like ImageNet and PASCAL VOC. Rotation invariance is an example from this set. Addressing rotational invariance with NORB would simply require slicing the data in to appropriate testing and training sets. The training set, for instance, might contain half of the stereo pairs collected from a single camera elevation and azimuth. The testing set would contain all the remaining data. The rich metadata contained in NORB means such a slicing would be easy to construct, and the output would be sufficient to investigate object classification performance as a function of rotation.

Replicating this experiment using the ImageNet or PASCAL VOC datasets would be both significantly more expensive and significantly more difficult. Hoiem et al. (2012) offers a template for how to construct the experiment. For a selected subset of the data, it would be necessary to manually label each image with a pose vector. Once each image has a pose vector, it would be possible to slice the data and run an analysis. Experimental control, however, would still be difficult. ImageNet and PASCAL VOC do not contain multiple poses of the same object. This means the training and testing sets would contain non-overlapping sets of objects. It thus wouldn't be possible to establish how well an object classification algorithm performs on a single object at various degrees of deflection.

The Hoiem et al. (2012) study did not attempt to address rotation invariance. It faced a similar problem when attempting to diagnose performance under occlusion, however. The authors had to manually label the level of occlusion for each of the images in the dataset. The source dataset (PASCAL VOC) contained many types and levels of occlusions, so the authors asked the human labelers to classify each object as no, low, medium, or high occlusion. Such an approach is sufficient to establish a relationship between occlusion level and classification performance. It is not sufficient, however, to measure the impact of different types of occlusions or to experiment with strategies to recover lost performance.

4.1.2 The SORBO dataset

The Synthetic Object Recognition Benchmark with Occlusions (SORBO) dataset is a new derivative of the NORB dataset. Like NORB, it is a tightly controlled, parametric dataset. The SORBO dataset applies the same design principles expressed in NORB to the problem of occlusion-resistant object classification. To do this, it includes two image parameters beyond those contained in NORB: *occlusion type* and *occlusion level*. The occlusion type indicates the class of occluding object in a given image. SORBO includes bars, blobs, and random noise occlusion classes. The occlusion level indicates the number of occluded pixels. A pixel does not need to contain an object to be included in this count, so the number of occluded pixels is equivalent to a ratio of occluded pixels over the entire image.

Beyond these two additions, SORBO stays as close as possible to NORB such that NORB results in the literature are directly comparable to SORBO results. In practice, SORBO achieves this aim by extending the NORB data. Rather than starting from entirely new images, SORBO contains NORB images with artificially overlaid occlusions. This is the *synthetic* aspect of SORBO.

The SORBO dataset is intended to address two questions:

1. How do object classification algorithms perform as a function of occlusion level?
2. Are strategies to mitigate the impact of occlusions on classification performance effective?

The first point was covered by the analysis in Hoiem et al. (2012): existing state-of-the-art object classification algorithms are less effective as occlusion increases and nearly useless in heavily occluded conditions. SORBO includes sufficient metadata to replicate this analysis. The second point is uniquely addressable with SORBO. The

SORBO dataset includes ground-truth segmentations for all images. This ground-truth segmentation contains an alpha-matte consistent with the approaches in Grady et al. (2005) and Rother et al. (2004). Pixel values in the alpha-matte are either 0 or 1, where 0 is entirely background and 1 is entirely foreground.

4.1.3 Inpainting over occlusions

Inpainting is a standard technique for restoring damaged paintings and images. Originally an exclusively manual process performed by skilled artists to restore art, modern computing has enabled a number of digital equivalents. The essential goal of inpainting is to minimize the appearance of damaged sections by painting over those sections in a manner consistent with the rest of the image (Bertalmio et al., 2000). Figure 4·2 is a successful example.

Automated inpainting algorithms differ primarily on how they define *consistent*. The images in Figure 4·2 use an algorithm described in Warren (2010). This algorithm models texture as a single color value and consistency as local similarity of color. Pixels that are known to be good are treated as *Dirichlet* boundary constraints. A Dirichlet pixel is one with a *fixed value*. These pixels do not change during diffusion and source color to adjacent floating pixels. After many iterations of color diffusion, floating regions are filled in with information from the surrounding Dirichlet pixels. As shown in Figure 4·2, this strategy is highly effective when the region to fill in is either relatively small or located in a neighborhood without many edges.

Linear diffusion with boundary constraints tends to produce poor results when filling in large regions or regions that ought to contain many edges. This weakness is primarily a result of the limited texture model. The algorithm is modeling texture as a single color value, but a single color value cannot represent an edge. This means that the algorithm will not propagate edges across large regions of missing information. More sophisticated inpainting algorithms use richer representations to prevent smearing in these large regions. Linear diffusion is substantially more resource-efficient, however. Better algorithms produce better results. Linear diffusion produces good results, but requires very little compute time per image.

4.2 Methods

The principle technical objectives for this chapter are to reproduce the results from Hoiem et al. (2012) as a baseline case and enable experiments in occlusion-resistant object classification algorithms. These goals necessitate a scalable experimental infrastructure that supports large parametric studies.

In this case, *scalable* means that the data collection infrastructure can efficiently distribute work over a pool of compute nodes. Given a perfectly efficient distribution of work, a task that would take a hundred hours to execute on a single compute node would require only four hours to execute on a cluster of 25 compute nodes. Most compute tasks cannot be distributed this efficiently over a large pool of compute nodes. The overhead required to keep all of the compute nodes synchronized limits performance as the size of the cluster increases. In this case, however, each trial is *independent*. The compute nodes do not require synchronization during the trials. This is an example of an *embarrassingly parallel* job, for which the theoretical scalability is nearly perfect. Given the large volume of work and the embarrassingly parallel nature of the problem, an experimental infrastructure designed to operate on a compute cluster is a reasonable investment.

All SORBO experiments used a two-tier parallel infrastructure. The management tier consists of a *version control server* and a *head node*. The compute tier contains 16 *compute nodes*. The version control server is responsible for storing the source code and small binary assets, as well as a history of changes. The rest of the infrastructure only executes code checked out from the version control server to ensure that all simulations are traceable and repeatable. The head node is responsible for storing large binary assets, coordinating experiments, distributing work to the compute nodes, and archiving the results. Seed data for SORBO is stored here because it is too large to place on the version control server. The compute nodes are responsible for the actual execution of trials.

Validating the Hoiem et al. (2012) study required three classifiers and three types of training data. The three classifier types are *chance*, *perceptron*, and *ConvNet*. The three types of training data are *unoccluded*, *occluded*, and *combined*.

The chance classifier simply discards the training data and generates random predications for the testing data. The perceptron classifier is a linear network trained by stochastic gradient descent. The ConvNet classifier is a multi-layer convolutional neural network modeled after the classifier in LeCun and Bottou (2004). The "Benchmark algorithms" section contains additional detail on these algorithms.

Each of the training data conditions is a different slice of the SORBO dataset. The unoccluded training condition contains only NORB images, so algorithms trained in this condition are directly comparable to NORB results in the literature. The occluded condition contains only images with occlusions applied. The mixed training data condition contains both types of data. The "SORBO construction" section contains additional detail on each condition.

4.2.1 SORBO construction

SORBO derives from the NORB normalized-uniform dataset, also termed "small NORB." Like NORB, it contains training and test sets of 96x96 pixel stereo images shot with fixed disparity. Also like NORB, SORBO contains descriptive metadata for the object class, object instance, camera elevation, camera azimuth, and light level for each image. SORBO adds two features to NORB:

1. Descriptive metadata on the class and level of occlusion

2. A ground-truth segmentation for each image pair

The additional metadata contains sufficient information to completely describe the occlusion composited over each image. Every image contains two new metadata fields: *occlusion_type* and *occluded_pixels*. The occlusion type field is a factor variable with a value chosen from the the set $\{none, random, bars, blobs\}$. The occluded pixels field contains an integer count of occlusion pixels summed across the left and right images. The count is taken over *every pixel in the field of view*, not just the pixels where the target object is present. This means the maximum value for *occluded_pixels* is $2 * 96 * 96$, or 18432. The underlying assumption for this choice is that all pixels from the normalized-uniform NORB data are considered figure. The target object in the data is of normalized size and centered on a uniform background, so this is a reasonable assumption.

Additional metadata fields describe the specific occlusion in each images for every occlusion type except *none*. For occlusion type *random*, there are supplementary *seed* and *threshold* fields. The seed is an integer used to prime the pseudo-random number generator that produces the random occlusion. The threshold field is a floating-point value in the range $[0, 1)$ that controls the fraction of pixels that are occluded.

For occlusion type *bars*, there are supplementary *omega*, *theta*, *phase*, and *threshold* fields that control the spacing, orientation, and thickness of the occluding noise bars. Bar occlusions are generated by thresholding a sinusoidal function along an offset axis. Omega is the angular frequency of the sinusoid. Theta is the angular offset with respect to the horizontal. The phase determines the starting point on the sinusoidal function. The threshold is a floating-point value in the range $[0.1, 0.9)$. Like the random case, it controls the percentage of pixels that are occluded.

For occlusion type *blobs*, there are supplementary *seed*, *blob_n*, and *blob_scale* fields. Like the random case, the seed is an integer used to prime a pseudo-random number generator. The *blob_n* field is an integer describing the number of blobs in the image. The *blob_scale* field is an integer describing the size of the blobs.

The *occlusion_type* field, along with the supplementary fields, fully describe the occlusions and are sufficient to reconstruct them given an additional disparity parameter. The disparity parameter is an integer that describes the ocular disparity of the stereo occlusions measured in pixels. It is a global parameter with a value of five. This value places the occlusions at a depth closer to the observer than the target objects. It is an integer to simplify segmentation. With an integer value, every pixel in the data contains information for either the background or the occlusion with no mixing. There are no border pixels that contain some information from the target object and some from the occlusion layer. A natural photograph would very rarely have objects fall precisely on pixel boundaries in this way.

Beyond the additional metadata, SORBO also contains ground-truth segmentations for both the training and testing sets. These ground-truth segmentations adopt the alpha-matte standard used in Grady et al. (2005) and Rother et al. (2004). For each stereo pair in the data, SORBO contains an additional pair of $96 * 96$ binary alpha-matte images. A value of 1 in the alpha-matte indicates that the pixel is an occlusion. A value of 0 indicates the lack of occlusion.

Like NORB, the images in SORBO are grayscale. For compact storage, each pixel in NORB is described by a single byte. This means the images in NORB contain 256 gray levels and each pixel has a value in the range $[0, 255]$. Zero is black and 255 is white. For SORBO, ease of integration with off-the-shelf object classification algorithms was a higher priority than compactness of representation. SORBO is stored as 32-bit floating-point values in the range $[0, 1]$, where 0 is black and 1 is white. This choice of simplicity over compactness is reasonable given that networks are much faster and disks are much larger than those available when NORB was created.

Occlusions in SORBO are textured with binary noise. Each occluded pixel has a value of either 0 or 1, chosen from a uniform distribution. This texture is necessary both to maximally confuse the object classification algorithms and provide an information source for the performance-recovery algorithms. See section 4.2.4 for more detail on this choice.

SORBO contains three different training sets and one test set. The first training set, called *clean*, contains only the 24,300 original images from the NORB normalized-uniform training set. No occlusions are added. This training set is a baseline that allows direct comparison with the many NORB results in the literature.

The second and third training sets are called *comb*, or combined, and *occ*, or occluded. Both training sets contain 243,000 stereo pairs. This size is consistent with the jittered and cluttered versions of the NORB dataset. Each stereo pair is generated by selecting a random stereo pair (with replacement) from the 24,300 pairs in the NORB normalized-uniform training set, selecting a random occlusion type, generating random parameters for the chosen occlusion type, and generating random noise to serve as the texture for the occlusion. The combined training set contains stereo pairs with all four occlusion types, including *none*. The occluded training set excludes the *none* occlusion type, so all stereo pairs are occluded. All random choices are made with a uniform distribution over the possible options.

The *none* occlusion type has no supplemental metadata, so no additional random choices need to be made and no occlusion texture is generated. The *random* occlusion type requires a seed and a threshold. The seed is generated from the space of all possible 32-bit integers. The threshold is floating-point and generated from the range $[0, 1)$. The *bars* occlusion type requires floating-point omega, theta, phase, and threshold parameters. Omega is generated from the range $[0.5, 10)$, theta from $[0, \pi)$, and phase from $[0, \pi)$. The threshold is generated from the range $[0.1, 0.9)$. Finally, the *blobs* occlusion type requires *seed*, *blob_n*, and *blob_scale* parameters. The seed is generated from the space of all possible 32-bit integers. The *blob_n* parameter is an integer generated from the range $[1, 9]$. The *blob_scale* parameter is an integer generated from the range $[-5, 5]$.

The SORBO testing set is generated using the same procedure as the combined training set, with two exceptions. First, the SORBO testing set uses the NORB *testing* stereo pairs instead of the training stereo pairs. The NORB testing and training sets are structured identically. The difference is the object instances present in each set. The training data contains instances 4, 6, 7, 8, and 9 of each category. The testing data contains the remaining instances. This means the testing and training data do not contain the same objects, just the same object categories. Second, the SORBO testing set only contains 97,200 stereo pairs. This dataset size is consistent with the jittered and cluttered versions of NORB.

SORBO construction procedure

The full SORBO dataset contains 607,500 grayscale stereo pairs and an equal number of alpha-mattes stored as 32-bit floating-point values, for a total of approximately 83 gigabytes of raw data. A total of 64 bits are consumed to represent each pixel, including both the gray level and alpha-matte. Reducing the per-pixel data from 64 bits to 9 bits, including 8 for the gray level and 1 for the binary alpha-matte, would reduce the raw data volume to approximately 12 gigabytes. Compression might reduce this value by another order of magnitude.

The raw data volume, however, is not the limiting factor for performance. Good machine learning practice dictates that the training data should be shuffled every time it is used to train a classification algorithm. If the SORBO image data were pre-computed and placed on disk, each training run would need to seek randomly in the data file for each stereo pair. Disk seeks are very expensive. Reading large blocks of sequential data from a hard disk is substantially more efficient than reading small blocks of random data.

Spreading work across a large compute cluster raises an additional set of practical issues. The data has to be physically placed on one or more disks. Placing it on a single disk and allowing the remaining nodes to access that master copy over the network is simpler, but the number of disk seeks per training epoch per disk increases linearly with the number of compute nodes and the network limits performance. This is *disk seek amplification*. Copying the data to a scratch disk on each node requires additional checks to ensure that stale versions of the data are expunged and each node always has a fresh copy. The number of disk seeks per training epoch per disk remains constant, however, so throughput is dramatically higher.

Leveraging the detailed occlusion metadata, the SORBO construction process uses a third option. The NORB normalized-uniform testing and training image data lives on a single node with a shared file system, along with the full SORBO metadata. This metadata is vastly smaller than the image data, at only tens of megabytes. The NORB normalized-uniform image data is vastly smaller than the full image data for SORBO, at approximately 3 gigabytes when stored in 32-bit floating-point format.

When a compute node needs to construct a shuffled version of SORBO to train or test a classification algorithm, it reads the metadata from the shared filesystem, *shuffles only the metadata*, and builds a local, shuffled copy of the SORBO image data on the fly. For each row in the shuffled metadata, the compute node fetches the appropriate image from the NORB files on the shared filesystem, adds the appropriate occlusion, and writes a copy of both the occluded data and the alpha-matte to the local disk.

Superficially, this scheme appears to suffer from the same disk seek amplification problem as the simpler scheme in which SORBO is pre-computed and stored on a single shared filesystem. In practice, however, only approximately 1.5 gigabytes of data are “hot” at any given time, meaning being read by the compute nodes. This is a small enough volume that the operating system caches on the node hosting the shared filesystem can absorb the load. Given the interconnect and node properties of the compute cluster, reading the NORB data from a single node does not adversely affect total performance.

This hybrid scheme splits the construction of SORBO in to two phases. The first phase is a pre-processing stage to prepare the metadata and convert the NORB normalized-uniform image sets to a suitable format. A single compute node executes this stage, since it only needs to occur once. The compute node writes the metadata to the shared file system as a textual file in comma-separated values (CSV) format. It also writes the NORB image data as two binary files, one for training and one for testing, each contained a sequence of 32-bit floating-point values in little-endian format. The sequence order is row major, assuming a $24300 * 2 * 96 * 96$ array of data. The binary file contains no header or delimiter information. This format is efficient because it directly corresponds to the layout of the data in memory.

In the second stage, the worker compute nodes load the CSV metadata, map the appropriate block of NORB image data in to memory, and build a local, shuffled copy of SORBO on the fly. To keep memory usage under control and accelerate the simulations, this process is parallelized over a pool of worker processes on each node. Object classification algorithms do not get access to the entire dataset at once. Instead, they get a sequence of mini-batches, each containing a thousand rows of input data. This eliminates the need to wait for the entire dataset to finish processing before the object classification algorithm can start working. As each mini-batch is completed, it is passed in to the object classification algorithm.

4.2.2 Benchmark algorithms and training

The processing pipeline includes three reference object classification algorithms. The first is a chance classifier that ignores the training data and makes random predictions for the testing data. The second is a simple linear perceptron trained by stochastic gradient descent. The third is a convolutional neural network (ConvNet). Both the linear classifier and ConvNet implementations are off-the-shelf open-source libraries.

The linear classifier comes from the scikit-learn library (Pedregosa et al., 2011). Scikit-learn is an open-source machine learning toolkit written for the Python language. SORBO classification results use the *sklearn.linear_model.Perceptron* classification algorithm with default parameters. A perceptron classifier is a good choice because the algorithm is simple, highly scalable, and amenable to incremental training. Perceptrons only require one data point at a time in order to learn. This means the processing pipeline can stream data past the classifier without keeping the entire dataset in memory. The classifier was trained with a single pass through the shuffled training set.

The convolutional network uses the open-source cuda-convnet package (Krizhevsky et al., 2012). This package offers a highly optimized implementation of convolutional networks for NVIDIA graphics processors. The specific network structure is user-configurable. All SORBO experiments used a two-layer network. The first layer is convolutional, and used a bank of 16 5-by-5 stereo filters with a hyperbolic tangent output function. This hyperbolic tangent is followed by an absolute value nonlinearity. An absolute value nonlinearity outperformed a rectifying nonlinearity in pilot experiments. The second layer is a fully-connected network with five outputs. These five outputs correspond to the five object classes. The output from the fully-connected layer passes through a softmax function to generate final prediction probabilities.

The ConvNet classifier required 60 passes through the full training dataset to converge. For the first 40 passes, the convolutional and fully-connected layers used a learning rate of 0.001 on the weights and 0.002 on the bias values. Passes 41 - 50 used rates of 0.0001 and 0.0002. Passes 51 - 60 used rates of 0.00001 and 0.00002. The decreasing learning rates are a form of *early stopping* (Prechelt, 2012). After 40 passes through the training data, the network starts to over-fit when trained with a high learning rate and becomes less accurate. The decreasing learning rate for the final 20 passes allows the network to fine-tune performance without over-fitting.

4.2.3 Recovery using attenuation or inpainting

The SORBO pipeline includes two recovery algorithms. The first is a simple attenuation. This algorithm produces a final input image by setting occlusion pixels to black. Figure pixels pass through without modification. Attenuation leaves visible occlusions but eliminates the high-frequency texture on the occlusions.

The second recovery algorithm is an open-source digital inpainting algorithm from the OpenCV library. This algorithm treats occluded pixels as damage and attempts to fill them in using information from neighboring pixels. The result is an image with no visible occlusions, but varying amounts of distortion due to the missing information at the locations of the occlusions. Figure 4·3 contrasts these two strategies.

4.2.4 Stereo segmentation

Automatic segmentation is a difficult and frequently ambiguous problem. State-ofthe-art segmentation algorithms like GrabCut still require input from a human to cue the algorithm to the correct target object and clean up the results (Rother et al., 2004).

In practice, the stereo occlusions in SORBO are easy to segment. This follows from the orientation of the occlusions relative to the observer. SORBO uses planar occlusions placed perpendicular to the camera. This means that the occlusion portions of the left and right images will match perfectly when the two images are aligned correctly. The automatic segmentation process leverages this property to estimates the locations of the occluding pixels by correlating the left and right input images and looking for a sharp peak in the correspondence. A peak is defined as a value that is twice the magnitude of its immediate neighbors. After identifying a peak in the correspondence, the algorithm predicts a class for each pixel. Pixels that are very similar when the left and right images are aligned are classified as occlusion.

The segmentation algorithm is effectively a simple approximation of stereo depth estimation. Estimating the depth of a pixel from a stereo image pair requires solving a *correspondence* problem between the left and right images. For a given landmark in the environment, the correspondence problem is to locate that landmark in both the left and right images from the stereo pair. These two pixel-space coordinates, along with the distance between the cameras when taking the stereo pair, are sufficient to solve for the distance to that landmark. The planar, perpendicular, textured nature of the occlusions yields a very simple correspondence problem. Occlusions are simply pixels in the near depth plane.

Sample outputs from the algorithm are shown in Figure 4·4.

4.3 Results

SORBO contains five classes of objects. Each class contains an approximately equal number of examples. This means that a classification algorithm which makes a random guess for each testing sample should still achieve an accuracy of approximately 20%. As shown in Figure 4·5, this holds up in practice. The *chance* classification algorithm ignores the training data and generates a uniformly random category label for each testing example. This produces accurate predictions of approximately 20%

for every trial, at every level of occlusion, and with every variant of the SORBO training set.

With the remaining classification algorithms, the training protocol has a significant impact on performance. As shown in Figure 4·6, the Combined variant of the SORBO training set produces the most reliable results. This variant includes both occluded and unoccluded training examples. The Unoccluded training set includes only the original 24,300 NORB stereo pairs. While this training condition produces better results with a ConvNet when tested on only unoccluded testing images, the classification performance degrades much more quickly than either of the training conditions that include occlusions. The Combined and Occluded training set variations include 243,000 examples. Each class of occlusion appears with equal probability. This means the Occluded set contains 60,750 more occluded examples than the Combined set. The "chance" line indicates the 20% classification accuracy achievable by guessing randomly for each testing example.

Figure 4·7 casts these results by classification algorithm rather than training regime. Only the Combined training results are shown for clarity. This plot confirms the central result of Hoiem et al. (2012) with regard to occlusion: state-of-the-art classification algorithms lose performance as occlusion increases and are only marginally better than chance in high-occlusion scenarios. Both the perceptron and convolutional network classifiers exhibit the same decline. The perceptron classifier, however, performs significantly worse in all conditions. It is also substantially more vulnerable to training order effects. The variance from run to run is much larger than with the convolutional network.

All remaining experiments exclude the perceptron classifier. Figure 4·8 covers the two no-recovery conditions with the convolutional network classifier. These are the baseline conditions for the construction of an occlusion-robust classifier. In both conditions, increasing occlusion causes a decrease in performance. Training with occluded data reduces the magnitude of this effect and increases robustness. Performance on unoccluded test images decreases, however.

Figure 4·9 builds from the combined case in Figure 4·8 and adds recovery mechanisms. Performance with the attenuation mechanism is indistinguishable from the no-recovery case at most levels of occlusion. At high levels of occlusion, attenuation is worse than no recovery. Inpainting, however outperforms the no-recovery case at *all* levels of occlusion. This includes unoccluded test images, suggesting that the disoccluded training images improve generalization.

Figure 4·10 further explores the generalization effect. The control and occluded training conditions come from Figure 4·8. The recovery condition is the inpainting case from Figure 4·9. Inpainting outperformed occluded training on unoccluded test data. Unoccluded training, however, also outperformed occluded training on unoccluded test data. Figure 4·10 shows that the recovery condition performs the best even in a direct comparison with the control.

All prior experiments used the ground-truth segmentation masks. As suggested by Figure 4·4 the automated segmentation algorithm does not have perfect accuracy and thus may degrade performance of the classifier. Figure 4·11 quantifies the type and level of error. The automated segmentation algorithm makes no errors on unoccluded images. Error peaks at a low but non-zero degree of occlusion. False alarms are more common than misses. Error decreases and the balance of false alarms to misses shifts towards misses as the occlusion level increases. These facts suggest that the algorithm has a bias towards indicating a pixel as an occlusion.

Figure 4·12 expands Figure 4·10 with a second recovery condition. The second recovery condition uses the automated recovery algorithm rather than the ground-truth segmentations. Performance with the new automated segmentation algorithm matches the system using ground-truth segmentations in high-occlusion conditions. The generalization effect on unoccluded testing images no longer appears, however.

4.4 Discussion

Objects that are heavily occluded are more difficult to classify than unobstructed objects. Pervasive observer bias in the major object classification datasets has masked this effect, however, limiting the utility of state-of-the-art object classification algorithms in the real world.

This chapter introduced the Synthetic Object Recognition Benchmark with Occlusions (SORBO) dataset. SORBO is a derivative of the earlier NYU Object Recognition Benchmark (NORB). Like NORB, SORBO is *parametric* and optimized for detailed analysis of classification performance. SORBO adds various classes and levels of stereo occlusions to NORB images to enable precise measurement of classification performance as a function of occlusion. The dataset is paired with an infrastructure suitable for high-throughput experimentation on a compute cluster.

Results on SORBO reproduce the analysis in Hoiem et al. (2012). A convolutional neural network exhibits high performance on unoccluded testing data, but degrades rapidly with increasing occlusion. At the highest level of occlusion, the performance of a high-quality classifier is little better than random chance. Training the classifier with a mix of occluded and unoccluded images produces the most reliably good results. All training conditions yield poor performance on highly occluded test images, however.

Augmenting a high-quality classifier with an inpainting pre-processing stage successfully recovers much of the lost performance. Inpainting using either a ground-truth or automatically extracted segmentation mask preserves the majority of performance all the way up to the highest level of occlusion. These results suggest that occlusion-robust classification is possible so long as the segmentation problem is solvable.

While the automatically extracted segmentation masks are fully accurate for unoccluded images, recovery using the ground-truth masks indicates a surprising gain in performance. The ground-truth recovery condition outperforms both the control and occluded training conditions for unoccluded test images. This gain suggests the inpainting with ground-truth segmentation masks reduces over-fitting and improves generalization.

4.5 Conclusion

The results suggest two key findings. First, occluding and then dis-occluding a dataset using inpainting is a viable technique for dataset augmentation. State-of-the-art classification systems typically include one or more augmentation techniques to increase the size of the training dataset, reduce over-fitting, and improve generalization. Reflections, linear shifts, and elastic distortions are three common classes of augmentation applied to object classification problems. All three build an invariance to an expected transformation in the data. The digit three is still a three when shifted several pixels to the left, for instance. The SORBO results indicate that occlusion is another useful class of dataset augmentation.

Second, extending a standard classification algorithm with an inpainting pre-processing stage produces a system that performs at parity with the base system on unoccluded test images, but exhibits substantially more robust performance as occlusion increases. The critical variable for real-world systems is the quality of the segmentation masks. A classifier working in concert with a high-quality segmentation algorithm can perform nearly as well on heavily-occluded images as unoccluded images.

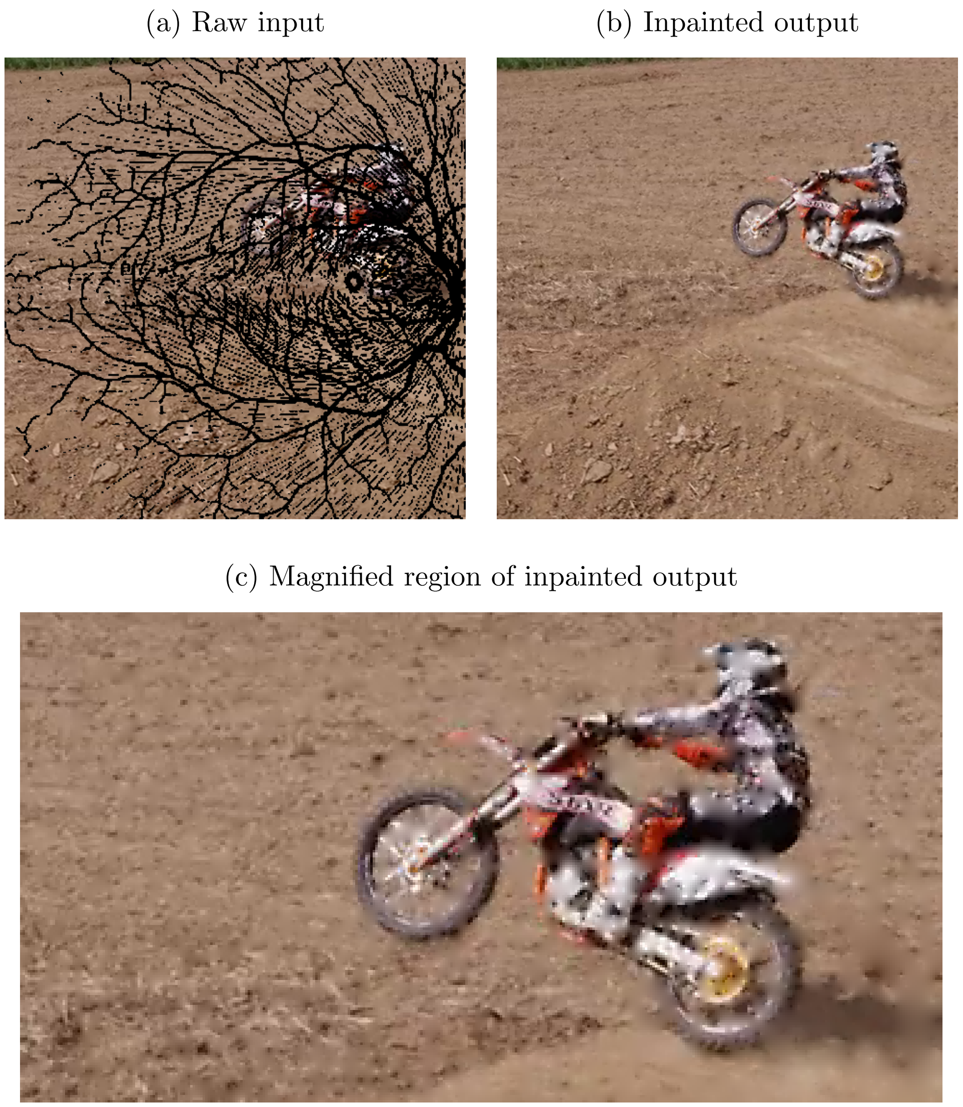
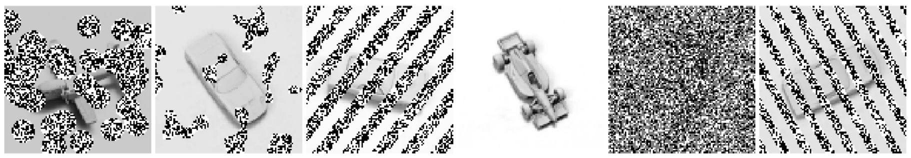
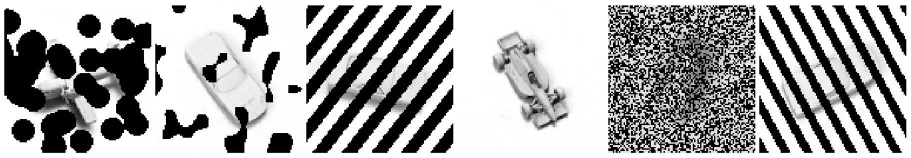


Figure 4·2: Inpainting over occlusions. The raw input (a) is a frame of video masked with an image of retinal veins. The algorithm does not have access to the pixels covered by the mask. A fast linear diffusion solver (b) uses the raw input and a segmentation estimate to fill in over the occluded regions. Magnifying the inpainted output (c) reveals the weaknesses of this simple technique. The dirt to the left of the image contains little edge information and inpainting works well. The boundaries of the rider, however, are badly blurred where an occlusion is present. More sophisticated inpainting algorithms do not have this problem but at a cost of significant additional complexity and a much longer compute time per frame.

(a) Sample images from SORBO dataset



(b) Occluded regions discounted by attenuation



(c) Occluded regions discounted by inpainting

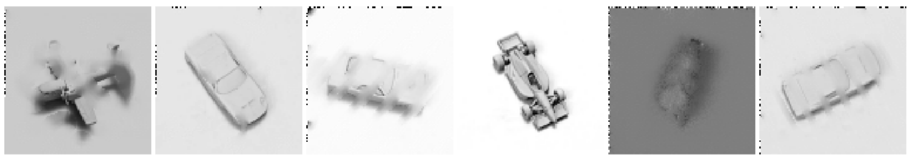


Figure 4·3: Occlusion recovery using attenuation and inpainting. Sample images from the SORBO dataset (a) are recovered using either attenuation (b) or inpainting (c). These recovery methods are candidate techniques for *discounting irrelevant information* from the images. Attenuation removes the occluding texture by setting occluding pixels to black. Inpainting infers a plausible value for the occluded pixels by using the remaining visible pixels. The dots along the borders of the inpainted images are an artifact of a boundary condition bug in the inpainting library routine.

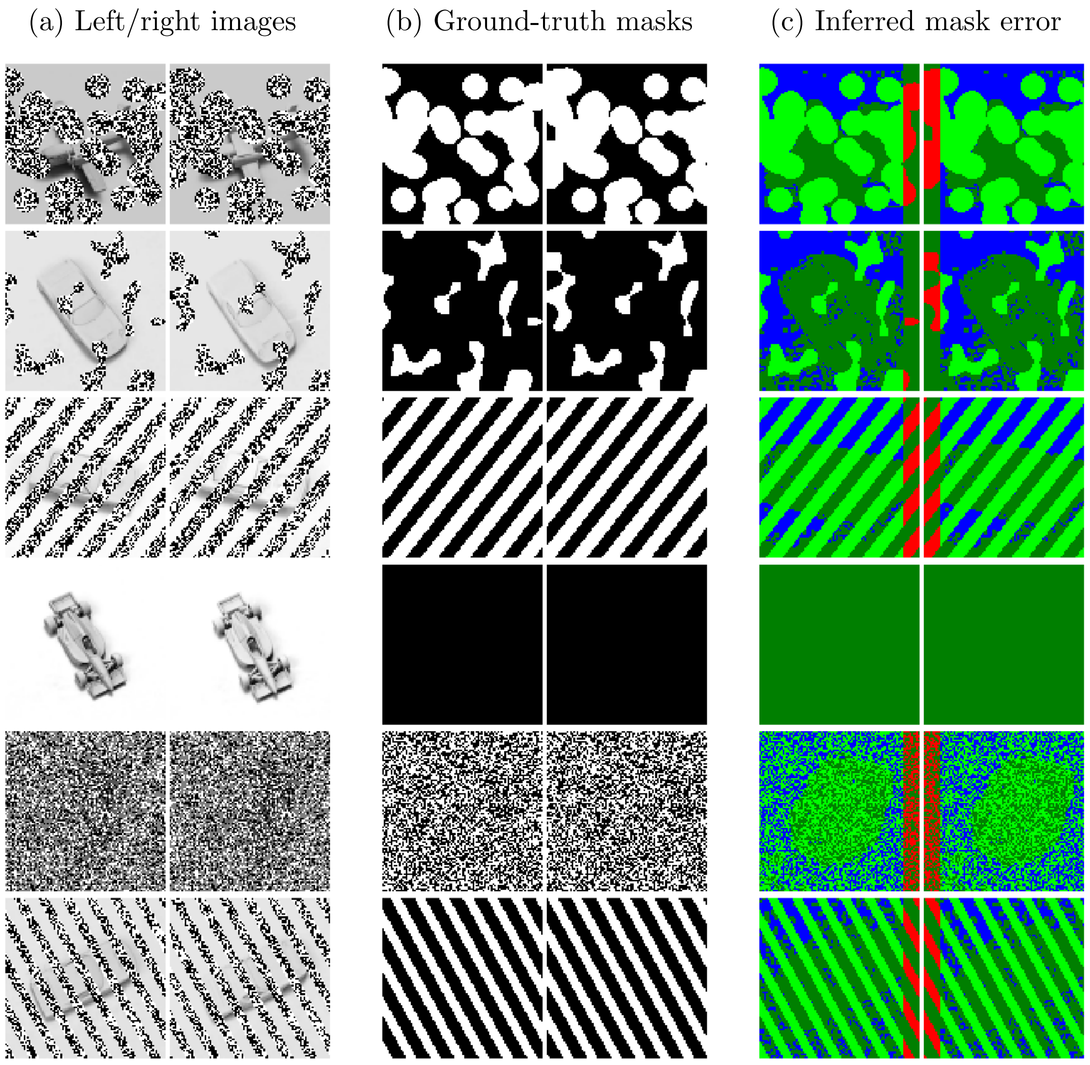


Figure 4·4: Accuracy of automatic stereo segmentation. NORB and SORBO are stereo datasets. Sample images from SORBO (a) contain various types and level of occlusions. SORBO also includes corresponding ground-truth segmentation masks (b). In these masks, white identifies occlusion pixels and black identifies target pixels. The stereo estimation algorithm produces a mask using only the raw input images (a). These estimates tend to have a high false positive rate, but miss occlusions only at the borders (c). In these error images, Light green corresponds to a hit. Dark green is a correct rejection. Blue is a false prediction of occlusion, or false positive. Red is a missed prediction.

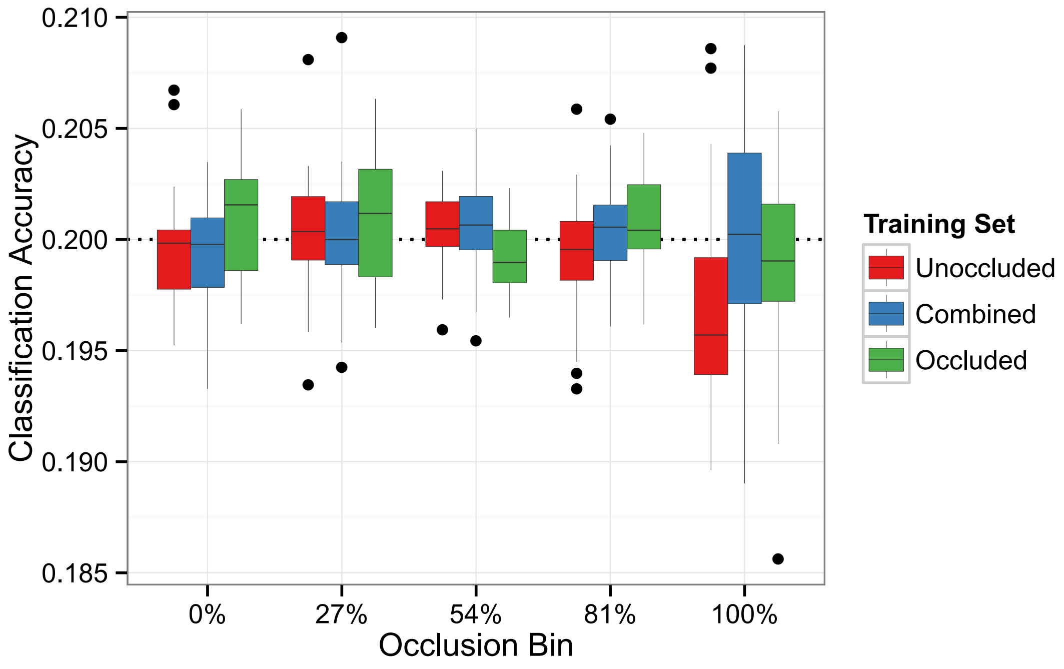


Figure 4·5: Classification results on SORBO as a function of occlusion level and training set with the chance algorithm. Points outside the box plots indicate outliers. The chance classification algorithm ignores the training data and makes a random guess for each testing sample. SORBO contains five classes with an approximately equal number of examples in each. The expected chance performance is therefore 20%. The occlusion bin percentage indicates the upper bound, inclusive, of the bin. For example, the 0% bin contains testing samples with zero occluded pixels and the 27% bin contains samples with greater than 0% and less than or equal to 27% occluded pixels. As expected, the chance algorithm scores an accuracy of approximately 20% for all trials, at every occlusion level, and with every variant of the SORBO training set.

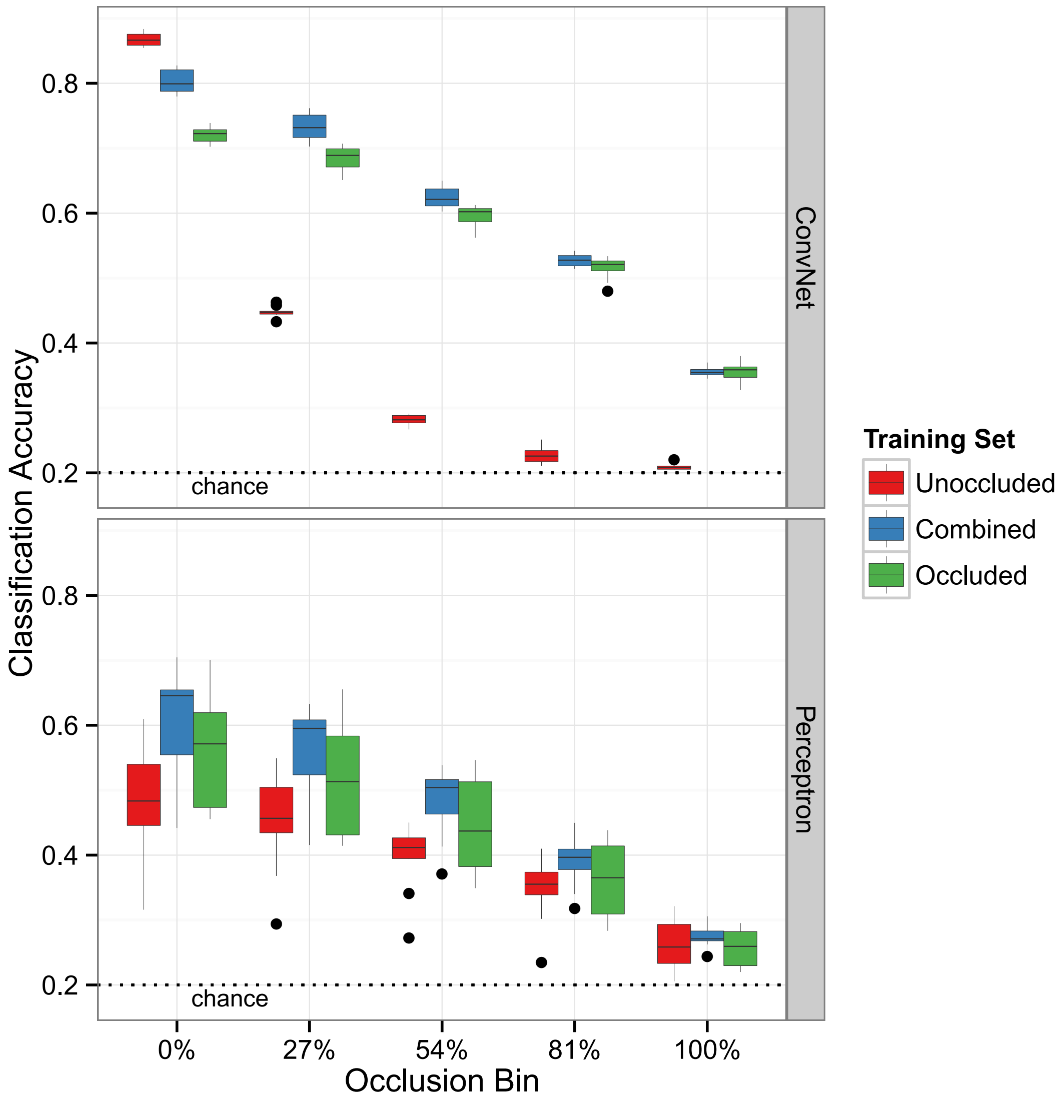


Figure 4·6: Classification results on SORBO as a function of occlusion level and training set with the Perceptron and ConvNet algorithms. The occlusion bins partition the testing samples as defined in Figure 4·5. Results with the Combined training set are either indistinguishable or better than the corresponding results with the Unoccluded or Occluded training sets with only one exception. A ConvNet training on unoccluded data outperforms the other two training options when tested on unoccluded data. Performance degrades more rapidly than the other conditions as the level of occlusion in the testing images increases, however.

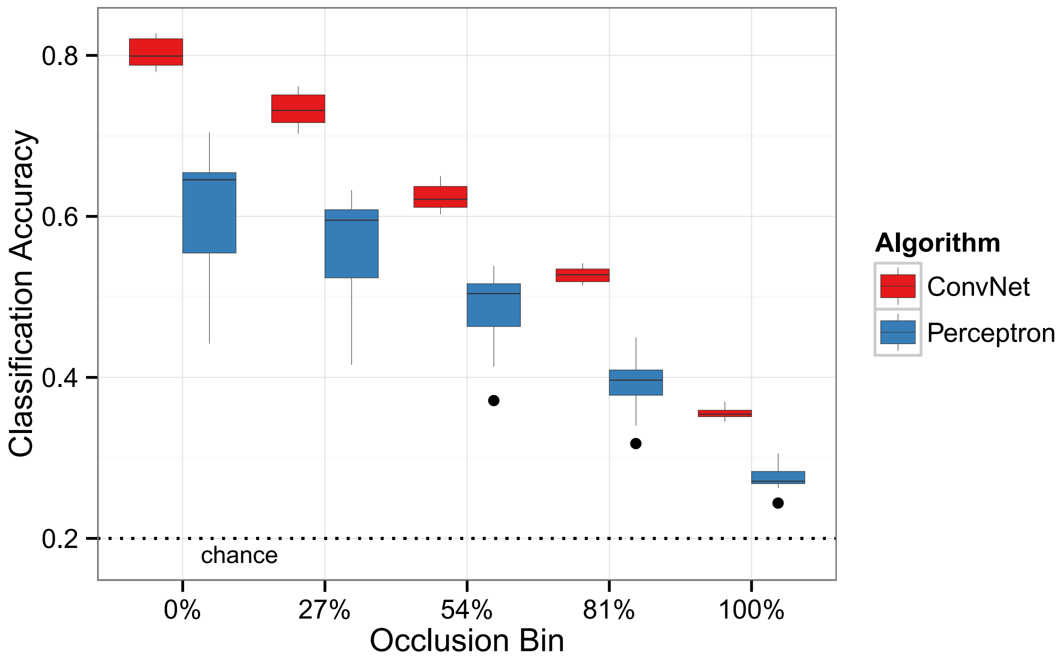


Figure 4·7: Classification results on SORBO as a function of occlusion level and classification algorithm with the Combined training set. The occlusion bins partition the testing samples as defined in Figure 4·5. Consistent with the analysis of Hoiem et al. (2012), classification performance decreases with increasing occlusion and drops to a level marginally better than chance in high-occlusion conditions.

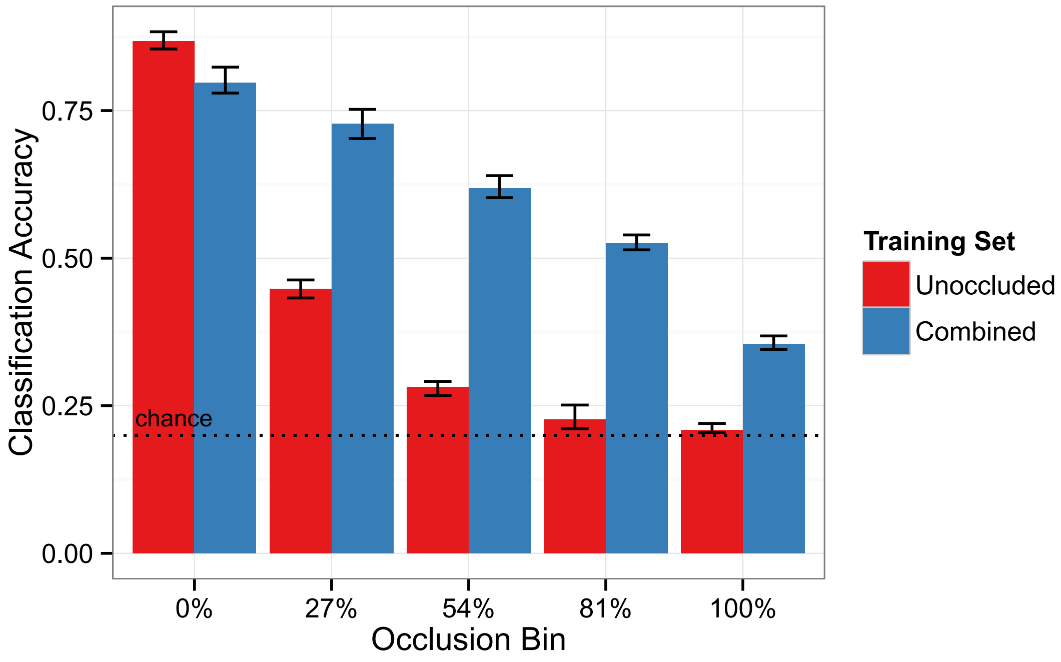


Figure 4·8: ConvNet classification accuracy when training with either unoccluded or occluded data and no recovery. Both convolutional networks have identical architecture. Both conditions pass the input images to the classifier with no attempt to discount occlusion pixels. In the "Unoccluded" case, the network is trained with the 24,300 image pairs in the NORB-simple training set. In the "Combined" case, the network is trained with the 243,000 image pairs contained in the SORBO-combined training set. Approximately a quarter of these image pairs are unoccluded. The remaining pairs contain various classes and levels of occlusion. Training with unoccluded images produces higher accuracy on the unoccluded testing images. The performance degrades quickly towards chance as the level of occlusion in the testing images increases, however. The network trained on combined data is less effective on unoccluded images but more robust to increasing occlusion.

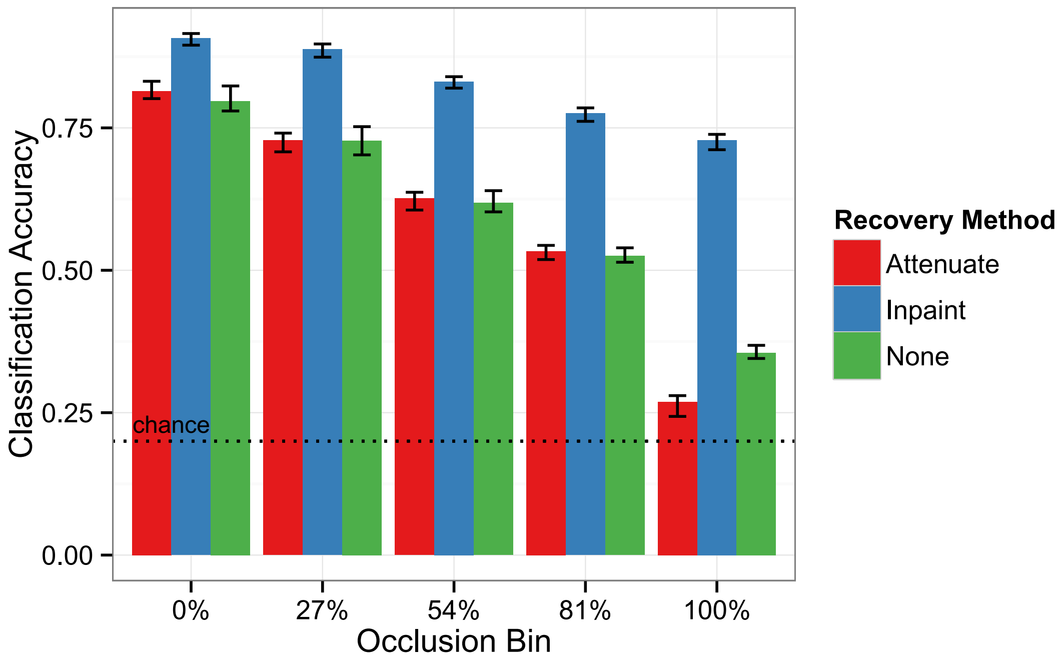


Figure 4·9: Performance comparison of mechanisms for discounting occluded pixels. All three cases use a convolutional network of identical architecture and trained on the SORBO-combined dataset. Both mechanisms for discounting occluded pixels use the ground-truth segmentation provided with the image data. The discounting mechanism is used during both the training and testing phases. In the "None" case, the training and testing images are passed through to the classifier with no attempt to discount occlusions. In the "Attenuate" case, occlusion pixels are set to black before going to the classifier. In the "Inpaint" case, occlusion pixels are filled in using a digital inpainting procedure. With a convolutional network classifier, attenuation is worse than the unmodified data. Inpainting, however, is dramatically more effective than the other candidates at every level of occlusion.

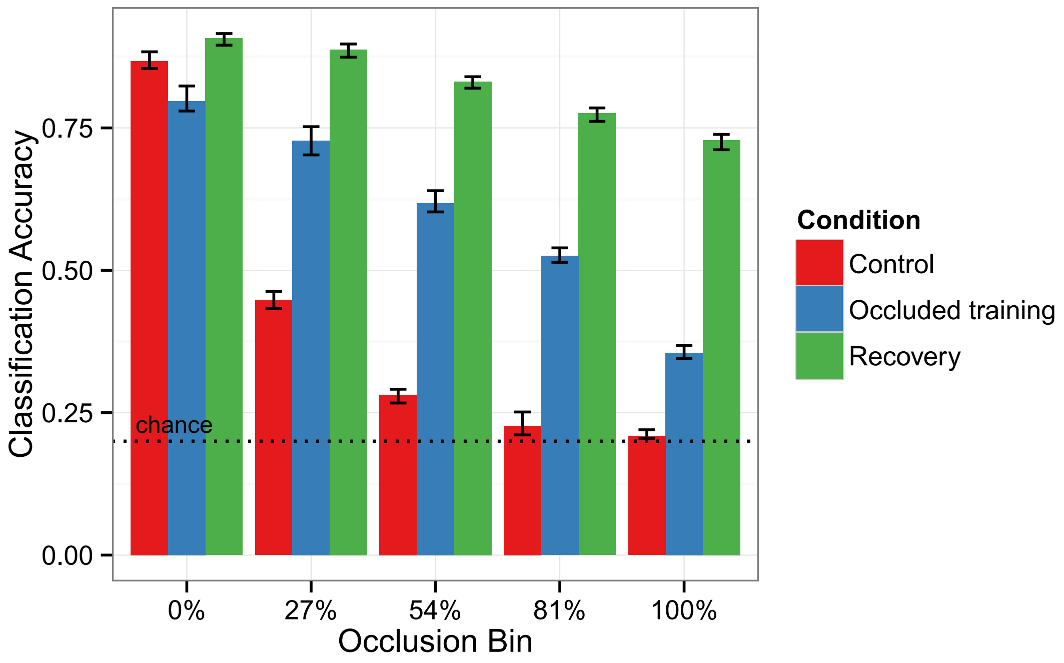


Figure 4·10: Inpainting improves performance on both occluded and unoccluded test images. The "Control" and "Occluded training" conditions are the unoccluded and combined training conditions from Figure 4·8. The "Recovery" condition is the inpainting result from Figure 4·9. The "Recovery" condition is consistently better as occlusion increases. Discounting occluded pixels using inpainting has the unexpected benefit of *also increasing performance on unoccluded images.* This is a dataset augmentation effect. The training set in the "Recovery" and "Occluded training" conditions is ten times larger than in the "Control" condition. Inpainting allows the network to leverage this larger training set without over-fitting.

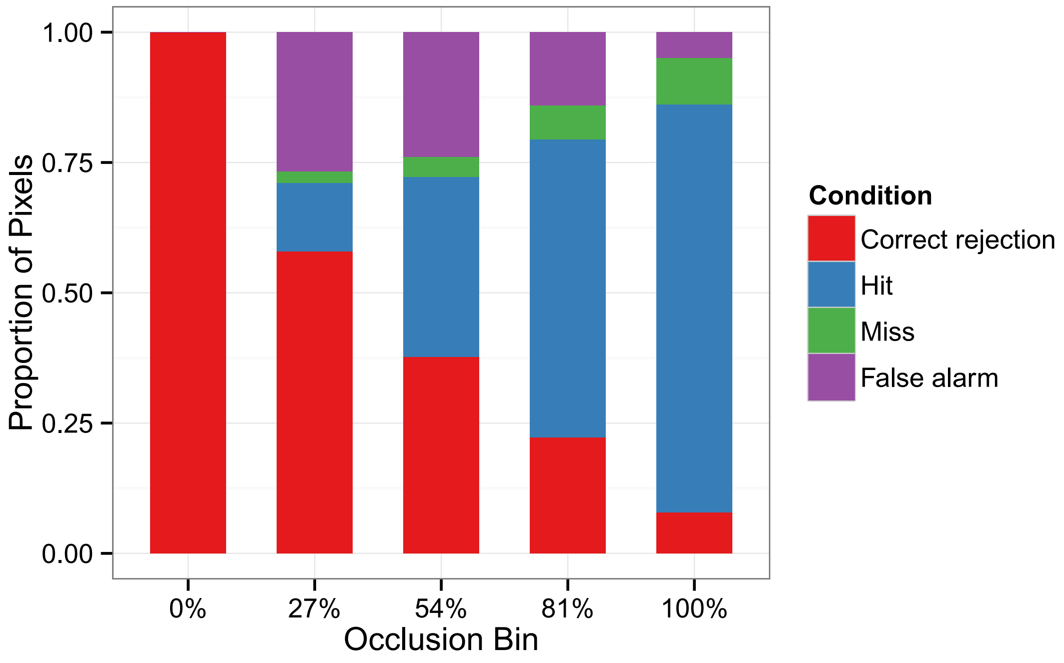


Figure 4·11: Analysis of segmentation errors. The automated segmentation algorithm leverages the planar structure of the occlusions to estimate which pixels are occlusion and which pixels are figure. Each pixel falls in to one of four conditions. In the "Hit" and "Correct rejection" cases, the estimate is correct. A hit occurs when the algorithm predicts the presence of an occlusion and an occlusion is actually present. A correct rejection occurs when the algorithm accurately predicts the lack of an occlusion. The "Miss" and "False alarm" cases are both errors. A miss occurs when the algorithm predicts the lack of an occlusion, but an occlusion is present. A false alarm occurs when the algorithm predicts an occlusion, but no occlusion is present. The accuracy of the automated segmentation process varies depending on the level of occlusion in the image pair. For unoccluded image pairs, the process is entirely accurate. At higher levels of occlusion, overall accuracy drops. False alarms, however, are much more common than misses. This indicates a bias towards a marking a given pixel as an occlusion.

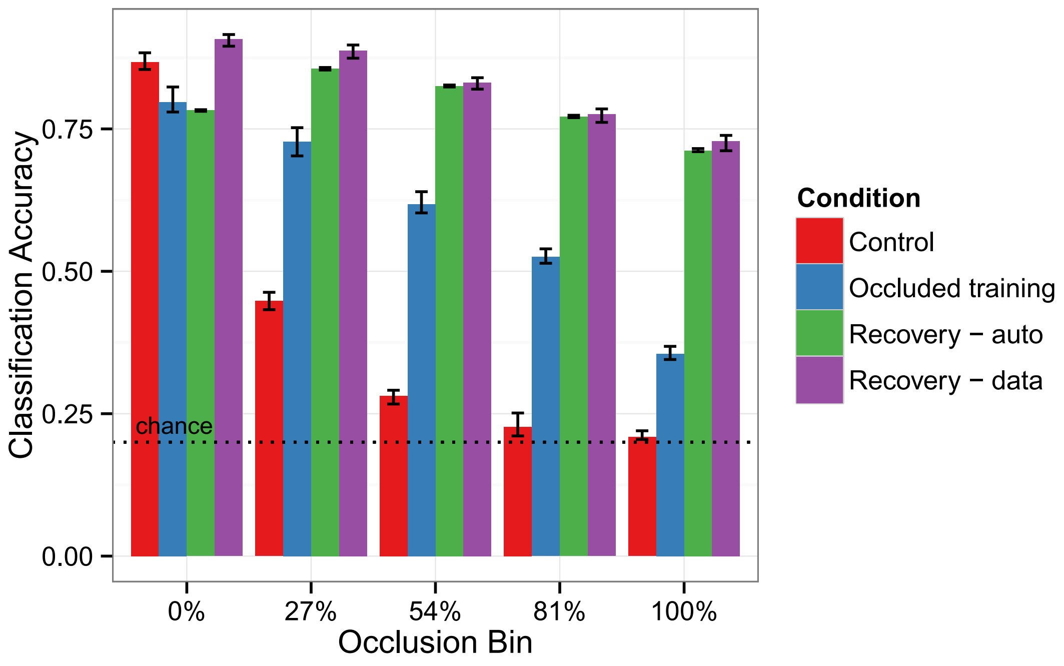


Figure 4·12: Performance with inferred versus ground-truth segmentation. The “Control”, “Occluded training”, and “Recovery - data” conditions are the same as in Figure 4·11. The “Recovery - auto” condition is new, and matches the “Recovery - data” condition except for the source of the segmentation masks. The data case uses the ground-truth segmentations provided with the dataset. The automatic case uses only the raw stereo image pairs and infers the segmentation masks. These two recovery conditions perform at parity at higher levels of occlusion. Using inferred segmentations erases the dataset augmentation effect observed in Figure 4·11, however. Performance on unoccluded test images is no better than the occluded training condition.

Chapter 5

Conclusion

Closed-loop operation, both internally and in concert with the environment, is the core of cognitive computing. Systems of this type are difficult to design because they have high internal complexity and must handle real-time interaction with the external world. This dissertation addresses a cluster of problems aimed at stepping towards practical cognitive systems.

Homeostatic plasticity is necessary to maintain stability in a large-scale learning system. Homeostatic plasticity decreases the amount of effort needed to construct a system by embedding self-stabilizing learning throughout that system. The system *adapts to its environment in real time.* A self-stabilizing system has fewer hyperparameters that must be set correctly a priori and thus lower design complexity.

Cluster-enabled development tools address design complexity from a different perspective. Conventional complex software systems incorporate a number of architectural features to simplify testing and tuning. Cognitive systems tend to lack these architectural features. Homeostatic plasticity allows the system to adapt to its environment and is thus one strategy for managing the resulting complexity. A highly scalable, simple-to-use infrastructure allows investigators to manage complexity by marshaling a cluster of computers to test and tune the system. The system still contains many hyperparameters, but the investigator can spread the resulting compute load over many computers.

Robust classification of occluded objects addresses visual sensing in the real world.

The standard classification datasets do not accurately capture all of the essential variables for real-world performance. Occlusion is devastating to state-of-the-art classification algorithms, but occlusions are so severely underrepresented in the standard benchmarks that this effect is hidden. A cognitive computing system, however, must be robust to the properties of the real world. A dataset with explicit bias is sufficient to measure how occlusion impacts performance. Augmenting a standard classification algorithm with a novel pre-processing stage both improves generalization and dramatically limits performance loss as occlusion increases.

The solutions for all three sections enable or simplify the construction of cognitive algorithms. *Locality* and *parallelism* are the features that tie closed-loop operation back to emerging hardware and enable *practical* cognitive computing. Biological computation uses a very large number of slow, spatially distributed processing elements and infrequent long-range communication to limit power consumption. Most information is kept local and high parallelism compensates for the slow speed of the individual processing elements. Very little existing software runs well on hardware of this class, however, despite the significant potential efficiency gains. *Cognitive computing* embraces massive parallelism and closed-loop operation to enable a new generation of intelligent machines; a generation that is sufficiently robust, fast, and power-efficient to operate in the real world.

References

Barlow, R., Hitt, J., and Dodge, F. (2001). Limulus vision in the marine environment. *The Biological bulletin*, 200(2):169–76.

Bertalmio, M., Sapiro, G., Caselles, V., and Ballester, C. (2000). Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, pages 417–424. ACM Press.

Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107.

Dean, J. and Ghemawat, S. (2010). MapReduce: a flexible data processing tool. *Communications of the ACM*.

Dongarra, J., Beckman, P., Moore, T., Aerts, P., Aloisio, G., Andre, J., Barkai, D., Berthou, J., Boku, T., Braunschweig, B., Cappello, F., Chapman, B., Choudhary, A., Dosanjh, S., Dunning, T., Fiore, S., Geist, A., Gropp, B., Harrison, R., Hereld, M., Heroux, M., Hoisie, A., Hotta, K., Ishikawa, Y., Johnson, F., Kale, S., Kenway, R., Keyes, D., Kramer, B., Labarta, J., Lichnewsky, A., Lippert, T., Lucas, B., Maccabe, B., Matsuoka, S., Messina, P., Michielse, P., Mohr, B., Mueller, M., Nagel, W., Nakashima, H., Papka, M., Reed, D., Sato, M., Seidel, E., Shalf, J., Skinner, D., Snir, M., Sterling, T., Stevens, R., Streitz, F., Sugar, B., Sumimoto, S., Tang, W., Taylor, J., Thakur, R., Trefethen, A., Valero, M., van der Steen, A., Vetter, J., Williams, P., Wisniewski, R., and Yelick, K. (2011). The International Exascale Software Project roadmap. *International Journal of High Performance Computing Applications*, 25(1):3–60.

Everingham, M., Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2009). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338.

Felleman, D. J. and Van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex*, 1(1):1–47.

Fischl, B. and Schwartz, E. (1997). Learning an integral equation approximation to nonlinear anisotropic diffusion in image processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):342 – 352.

Fortenberry, B., Gorchetchnikov, A., and Grossberg, S. (2012). Learned integration of visual, vestibular, and motor cues in multiple brain regions computes head direction during visually guided navigation. *Hippocampus*, 22(12):2219–37.

Francis, G., Grossberg, S., and Mingolla, E. (1994). Cortical dynamics of feature binding and reset: Control of visual persistence. *Vision Research*, 34(8):1089–1104.

Gaudiano, P. (1993). A nonlinear model of spatiotemporal retinal processing: Simulations of X and Y retinal ganglion cell behavior. *CAS/CNS Technical Report Series*.

Grady, L., Schiwietz, T., Aharon, S., and Westermann, R. (2005). Random walks for interactive alpha-matting. *Proceedings of VIIP*, pages 423–429.

Grossberg, S. (1973). Contour enhancement, short term memory, and constancies in reverberating neural networks. *Studies in Applied Mathematics*, pages 213–258.

Hoiem, D., Chodpathumwan, Y., and Dai, Q. (2012). Diagnosing Error in Object Detectors. *ECCV 2012*.

Hyvärinen, A. and Oja, E. (1998). Independent component analysis by general nonlinear Hebbian-like learning rules. *Signal Processing*.

JetBrains (2014). Continuous Integration for Everybody – TeamCity.

Kogge, P., Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hiller, J., Karp, S., Keckler, S., Klein, D., Lucas, R., Richards, M., Scarpelli, A., Scott, S., Snavely, A., Sterling, T., Williams, R. S., and Yelick, K. (2008). ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. Technical report.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, pages 1–9.

LeCun, Y. and Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 97–104.

Mckenzie, A., Branch, D. W., Forsythe, C., and James, C. D. (2010). Toward Exascale Computing through Neuromorphic Approaches. Technical report, Sandia National Laboratories.

Miller, G. (1995). WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.

Modha, D. S. and Singh, R. (2010). Network architecture of the long-distance pathways in the macaque brain. *Proceedings of the National Academy of Sciences of the United States of America*, 107(30):13485–90.

Oja, E. (1982). Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Perona, P. and Malik, J. (1990). Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639.

Pinto, N., Cox, D. D., and DiCarlo, J. J. (2008). Why is real-world visual object recognition hard? *PLoS computational biology*, 4(1):e27.

Prechelt, L. (2012). Early stopping – But when? In *Neural Networks: Tricks of the Trade*, pages 53–67. Springer Berlin Heidelberg.

Rother, C., Kolmogorov, V., and Blake, A. (2004). “GrabCut”: interactive foreground extraction using iterated graph cuts. *Computer*, 23(3):309–314.

Rutherford, L. C., Nelson, S. B., and Turrigiano, G. G. (1998). BDNF has opposite effects on the quantal amplitude of pyramidal neuron and interneuron excitatory synapses. *Neuron*, 21(3):521–30.

Snider, G., Amerson, R., Carter, D., Abdalla, H., Qureshi, M. S., Leveille, J., Versace, M., Ames, H., Patrick, S., Chandler, B., Gorchetchnikov, A., and Mingolla, E. (2011). From Synapses to Circuitry: Using Memristive Memory to Explore the Electronic Brain. *Computer*, 44(2):21–28.

Snider, G. S. (2007). Self-organized computation with unreliable, memristive nanodevices. *Nanotechnology*, 18(36):365202.

Socher, R. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Torralba, A. and Efros, A. (2011). Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528. IEEE.

Torralba, A., Fergus, R., and Freeman, W. T. (2008). 80 million tiny images: a large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970.

Turrigiano, G. G., Leslie, K. R., Desai, N. S., Rutherford, L. C., and Nelson, S. B. (1998). Activity-dependent scaling of quantal amplitude in neocortical neurons. *Nature*, 391(6670):892–6.

Turrigiano, G. G. and Nelson, S. B. (2004). Homeostatic plasticity in the developing nervous system. *Nature Reviews Neuroscience*, 5(2):97–107.

van Rossum, M. C., Bi, G. Q., and Turrigiano, G. G. (2000). Stable Hebbian learning from spike timing-dependent plasticity. *The Journal of Neuroscience*, 20(23):8812–21.

Warren, J. (2010). *Solving Diffusion Curves on GPU*. PhD thesis, University of Dublin, Trinity College.

Curriculum Vitae

Benjamin O. Chandler

Year of Birth	**1985**
Email	**benjamin.chandler@hp.com**
Address	**HP Labs** 1501 Page Mill Rd. Palo Alto, CA 94304
Education	**Boston University** *Ph.D. candidate* Sep. 2007 - present **Carnegie Mellon University** *B.S., Cognitive Science* Sep. 2003 - May 2007
Experience	**Systems Research Laboratory, HP Labs** *Research Scientist* Dec. 2012 - present **Dept. of Cognitive and Neural Systems, Boston University** *Research Assistant* Sep. 2007 - Nov. 2012 **Pittsburgh Supercomputing Center** *Student Researcher* Jun. 2004 - May 2007

Publications

Chandler, B. and Grossberg, S. (2011) Joining distributed pattern processing and homeostatic plasticity in recurrent on-center off-surround shunting networks: Noise, saturation, short-term memory, synaptic scaling, and BDNF. *Neural Networks.*

Snider, G., Amerson, R., Carter, D., Abdalla, H., Qureshi, S., Leveille, J., Versace, M., Ames, H., Patrick, S., Chandler, B., Gorchetchnikov, A., and Mingolla, E. (2011) Adaptive Computation with Memristive Memory. *IEEE Computer*, 44(2), 21-28.

Gorchetchnikov, A., Versace, M., Ames, H., Chandler, B., Leveille, J., Livitz, G., Mingolla, E., Snider, G., Amerson, R., Carter, D., Abdalla, H., and Qureshi, M.S. (2011) A Unified Learning Framework for Memristive Neuromorphic Hardware. *International Joint*

Conference on Neural Networks (IJCNN) 2011, San Jose, CA.

Livitz, G., Ames, H., Chandler, B., Gorchetchnikov, A., Leveille, J., Vasilkoski, Z., Versace, M., Mingolla, E., Snider, G., Amerson, R., Carter, D., Abdalla, H., and Qureshi, M.S. (2011) Visually-Guided Adaptive Robot (ViGuAR). *International Joint Conference on Neural Networks (IJCNN) 2011*, San Jose, CA, USA.

Vasilkoski, Z., Ames, H., Chandler, B., Gorchetchnikov, A., Leveille, J., Livitz, G., Mingolla, E., and Versace, M. (2011) Stability analysis of neural plasticity rules for implementation on memristive neuromorphic hardware. *International Joint Conference on Neural Networks (IJCNN) 2011*, San Jose, CA, USA.

Versace, M. and Chandler, B. (December 2010) MoNETA: A Mind Made from Memristors. *IEEE Spectrum.*

Leveille, J., Ames, H., Chandler, B., Gorchetchnikov, A., Mingolla, E., Patrick, S., and Versace, M. (2010) Learning in a distributed software architecture. *Lecture Notes for Computer Sciences, Social Informatics, and Telecommunications Engineering (LNICST).*

Heffner, J., Mathis, M., Chandler, B. (2007) IPv4 Reassembly Errors at High Data Rates. RFC 4963,
http://www.rfc-editor.org/rfc/rfc4963.txt

Awards **IGERT Fellowship,** Boston University 2010
Awarded through the Advanced Computation in Engineering and Science training program to support interdisciplinary computational research.