FUNDAMENTAL MOTIVATION AND PERCEPTION FOR A SYSTEMS-LEVEL COGNITIVE ARCHITECTURE

by

Ryan James McCall

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Computer Science

The University of Memphis

August, 2014

Acknowledgments

I am gratefully indebted to my advisor Dr. Franklin for his tireless feedback and guidance especially in matters of writing, planning, and slow, careful thinking.

I would also like to thank the members of my committee, Dr. Lan Wang, Dr. Dipankar Dasgupta, and Dr. Vasile Rus. Their comments and suggestions helped me to improve the content of this dissertation and the presentation of it.

I especially thank colleague Javier Snaider for challenging discussions, guidance in computational matters, and for his inspiring, exemplary work ethic all coupled with a deep kindness, friendliness, and selflessness.

Many thanks are due to Pulin Agrawal for his useful feedback and discussions on the computational implementation of PC-CLA. I am also thankful to Tamas Madl for helpful comments on earlier drafts of material in Chapters 4 and 5.

I must also express my gratitude to the entire CCRG research group, our discussions have helped shape me as a researcher: Steve Strain, Usef Faghihi, Daqi Dong, Siminder Kaur, Nisrine Khayi-Enyinda, Sean Kugele, Rodrigo Silva, Dr. Bernard Baars, Dr. Uma Ramamurthy, David Friedlander, and Paul Mouchon.

Finally, I must give special thanks to my parents and my partner Shumyla Jan for their patient, unconditional support over the years.

Abstract

Ryan McCall. Ph.D. The University of Memphis. August/2014. Fundamental Motivation and Perception for a Systems-Level Cognitive Architecture. Major Professor: Stanley P. Franklin.

A comprehensive systems-level cognitive architecture attempts to provide a blueprint for generally capable intelligent software agents or cognitive robots. While such architectures can be conceived and studied at a high level of abstraction, this work focuses primarily on some of the low-level algorithms underlying the architecture. For instance, one might study logical reasoning, decision-making, etc., ignoring or making simplifying assumptions about the perceptual processes producing the representations involved in the higher-level processes. In contrast, critical aspects of cognitive architectures lie in the low-level details of the traditionally identified, abstract modules and processes.

Natural selection has imbued biological agents with motivations driving them to act for survival and reproduction. Likewise, artificial agents also require motivations to act in a goal-directed manner. In this context, I present a motivational extension to the LIDA cognitive architecture integrated within LIDA's cognitive cycle at a fundamental level. This motivational extension provides a repertoire of motivational capacities including alarms, feelings, affective valence, incentive salience, emotion, appraisal, reinforcement learning, and model-free and model-based learning. A LIDA-based agent implementing the proposed motivational extension replicates a reinforcer devaluation experiment testing its ability to learn, and later revise, the reward predicting attributes of stimuli that drive its behavior.

Intelligent software agents must also autonomously navigate complex, dynamic, uncertain environments with bounded resources. In my view, this requires that they continually update a hierarchical, dynamic, uncertain internal model of their current situation, via approximate Bayesian inference, incorporating both the sensory data and a generative model of its causes. To explicate my approach, I identify perceptual principles for cognitive architectures influencing perceptual representation, perceptual inference, and the associated learning processes. Guided by these, I propose a predictive coding extension to the HTM Cortical Learning Algorithms, termed PC-CLA, as a potential foundational building block for the systems-level LIDA cognitive architecture. PC-CLA fleshes out LIDA's internal representations, memory, learning and attentional processes; and takes an initial step towards the comprehensive use of distributed and probabilistic (uncertain) representation throughout the architecture. I conclude with reports on a battery of new tests of the original CLA as well as proof-of-concept tests of PC-CLA.

Table of Contents

List of Tables

List of Figures

1 Introduction

Cognitive architectures are control structures for autonomous agents. An autonomous agent is a system situated within, and a part of, an environment, that senses that environment and acts on it, over time, in pursuit of its own agenda, so as to affect what it senses in the future (Franklin & Graesser, 1997). Given this, how can the agenda of an autonomous agent be implemented in a cognitive architecture? How does such agenda arise from basic cognitive processes? I address these foundational questions in the third chapter on artificial motivations.

Motivation is a broad term influencing everything from simple reflexive actions to decade-long career commitments. In this work, I discuss a number of motivational processes covering a range of temporal scales, and having varying complexity. My approach bases motivation on feelings. Motivation based on feelings for autonomous agents is not a new idea. Several researchers have addressed this topic to date (Bach, 2009; Canamero, 2003; Franklin & Ramamurthy, 2006; Gmytrasiewicz & Lisetti, 2002; Lang & Davis, 2006; Sloman & Croucher, 1981). Moreover, the topic of human emotion has seen a vast amount of study (e.g., Davidson, Maxwell, & Shackma, 2004; Davis & Lewis, 2004; Izard, 1993; Pessoa, 2008; Shariff & Tracy, 2011; Yiend, 2010; Zhu & Thagard, 2002).

It is not my aim to start *a priori* with human-level emotional categories, and attempt to explain their function, or how they arise. Rather, I present a motivational extension to the broad systems-level LIDA cognitive architecture (McCall, Franklin, Snaider, & Faghihi, in preparation), based on feelings, and integrated within the system at a fundamental level — an approach to motivation involved in the entirety of cognition.

However, I hold that only by building upwards from solid conceptual primitives can we provide precise systems-level explanations of such complicated phenomenon. The motivational extension is biologically inspired, but does not aim to model human or animal motivation at the level of neuroscience.

Chapters 4 and 5 explore modern perception (recognition) algorithms with properties hopefully suitable for future integration with the broad systems-level LIDA cognitive architecture. In these chapters, I propose a 2D extension to the HTM Cortical Learning Algorithms (CLA), termed 2D-CLA as well as a predictive coding extension, which I call PC-CLA (McCall & Franklin, 2013). The work is presented in the context of the LIDA model; however, it does not depend on the commitments of the LIDA model (Franklin, Strain, McCall, & Baars, 2013), and as a consequence, should be widely applicable. I review several inspirational models for this work including Hierarchical Temporal Memory, predictive coding, and others. Some of these, as models of cortical computation, have biological inspiration to their credit. Others feature equally compelling information-theoretic justifications, e.g., the free-energy principle (Friston, 2010), as well as mathematical justifications, e.g., implementing approximate hierarchical Bayesian[1] inference (Lee & Mumford, 2003). Crucially, I strive to identify key principles across these various approaches that will guide research and development of networks, like PC-CLA, capable of robust recognition of patterns not only having hierarchical structure, but temporal dynamics as well. Also, in the context of systems-level cognitive architectures, it should be possible to reconcile such a network with the commitments of the architecture. To this end, I discuss several theoretical considerations for integrating PC-

[1] Bayesian probability is one of the different interpretations of the concept of probability that can be seen as an extension of logic enabling reasoning with propositions whose truth or falsity is uncertain.

CLA, which has its own requirements, into the broad-based LIDA model. While such connectionist approaches may also have more narrow applications to pattern recognition problems, I focus here on PC-CLA as a potential building block for the implementation of the current representations, memory and learning, and attentional processes of LIDA as a systems-level cognitive architecture.

Problem Domain and Philosophy

Not long after its birth, artificial intelligence abandoned its original aim of reproducing human-level intelligence in favor of developing highly practical systems that behave intelligently in narrow, but nonetheless important, domains (Franklin et al., 2013). After a half century, a movement in AI research toward that original quest has emerged under the heading of *artificial general intelligence* (Goertzel & Pennachin, 2007; Wang, Goertzel, & Franklin, 2008). After an initial invitational workshop in 2006, five successful AGI conferences have been held in several locations including Europe, China, and the United States (e.g., de Garis & Goertzel, 2009). The last two conference proceedings were published in Springer's *Lecture Notes in AI* book series (Bach, Goertzel, & Iklé, 2012; Schmidhuber, Thorisson, & Looks, 2011). Additionally, *The Journal of Artificial General Intelligence* has published several volumes.

A parallel movement flies under the rubric of BICA (Biologically Inspired Cognitive Architectures). First appearing as several AAAI symposia carrying that name (Samsonovich, 2008), the movement has produced successful conferences of its own (Samsonovich & Johannsdottir, 2011; Samsonovich, Jóhannsdóttir, Chella, & Goertzel, 2010), and has started a journal, Biologically Inspired Cognitive Architectures, published by Elsevier. The aims and scope declaration of the journal begins with the sentence, "The

focus of the journal is on the integration of many research efforts in addressing the challenge of creating a real-life computational equivalent of the human mind." Note that the "BICA challenge," as it has come to be called, is quite equivalent to the goal of AGI.

Another such parallel movement was ushered in by the First Annual Conference on Advances in Cognitive Systems, held in Palo Alto, California in December 2012. Its call for papers asserts "The purpose is to provide a venue for research on the initial goals of artificial intelligence and cognitive science, which aimed to explain the mind in computational terms and to reproduce the entire range of human cognitive abilities in computational artifacts." A new online journal, also entitled Advances in Cognitive Systems, has published its second volume.

Yet another movement in this same direction is arising in the form of an AAAI Spring symposium entitled "Designing Intelligent Robots: Reintegrating AI." The Overview of its second incarnation (Spring 2013) includes the following lines:

> AI is a fragmented field: well-developed and largely independent research communities exist for learning, planning, reasoning, language, perception and control. Since the challenges posted by each of these subfields are immense, most researchers have found it necessary to devote their careers to specializing in a single subfield. While immense progress has been made in each of these subfields in the last few decades, it remains unclear how they can be integrated to produce an intelligent robot. Unifying these disparate technologies will open up new avenues of research and create new application opportunities. Therefore, we believe that integration should be considered a valid research endeavor in its own right. (Designing Intelligent Robots, 2012)

It would seem that the quest for a generally applicable, integrated AI system capable of human-level intelligence is an idea whose time has come. The question is how to go about it. Necessary components include human-level perception, action selection, and everything in between. This calls for controlling our generally applicable, integrated AI agent with a systems-level cognitive architecture capable of human-level intelligence.

Many cognitive architectures exist, though none, as of yet, at human-level intelligence (Samsonovich, 2010). Some of these architectures, for instance SOAR, ACT-R, CLARION, are decades old, while others are relatively new (Franklin, Strain, et al., 2013).

The need for using systems-level cognitive architectures has been championed in the past by several researchers. As social psychologist Kurt Lewin so succinctly pointed out, "There is nothing so practical as a good theory" (Lewin, 1951, p. 169). Artificial intelligence pioneer Allen Newell (1973) strongly supported the need for systems-level theories over isolated ones asserting, "You can't play 20 questions with nature and win." Echoing Newell in decrying the reliance on modeling individual laboratory tasks, memory researcher Douglas Hintzman (2011, p. 253) wrote, "Theories that parsimoniously explain data from single tasks will never generalize to memory as a whole…" Hintzman's arguments, which rest upon the need for systems-level cognitive architectures in memory research, carry over into the realm of intelligent agents, again calling for systems-level architectures. In their review article, Langley, Laird, and Rogers (2009, p. 2) argue that "Instead of carrying out micro-studies that address only one issue at a time, we should attempt to unify many findings into a single theoretical framework, then proceed to test and refine that theory." They are calling for a *broad-based, systems-level architecture*, such as the LIDA architecture. Any such systems-level cognitive model must necessarily be quite complex, and so, time-consuming to design and implement. Thus, at present, Samsonovich (2010) catalogs only about two-dozen such models, including the LIDA model.

Contributions of this Work

Motivation in intelligent agents is a complex subject at the intersection of many disciplines and theories. Some study human emotions and feelings as motivators, others have focused on the neurobiological aspects of emotion and motivation. Computer science is included with the disciplines of reinforcement learning and temporal difference learning.

In the Artificial Motivations Chapter (Chapter 3) I discuss how to extend the broad systems-level LIDA cognitive architecture to incorporate these various motivational concepts and mechanisms (McCall et al., in preparation). The aspects implemented include feelings as motivators, valence, incentive salience, reinforcement learning, temporal difference learning, model-free control, and model-based control. These additions include changes to aspects of the LIDA model's memory, current representations, and notion of salience. To complement these additions to the LIDA conceptual model, this work additionally extends the software framework implementation of the LIDA model to include support for motivation. Specifically, this work 1) adds an implementation of feelings nodes having affective valence, 2) extends regular nodes by allowing for current and base-level incentive salience, 3) extends links to pass incentive salience, and 4) adds learning algorithms related to incentive salience, including temporal difference learning to LIDA's PAM. To begin validating the proposed motivational extension, a computational LIDA agent replicates a psychological experiment testing motivational phenomena in biological agents.

Perception and pattern recognition are oft-ignored subjects in cognitive models, which are typically high-level. On the other hand, connectionist approaches are not

always concerned with algorithms suitable for systems-level cognitive architectures, which require online learning and unsupervised learning, and cannot have a human in the loop. In Chapters 4 and 5 I explore an algorithm capable of processing high dimensional inputs, which has relevance to cognitive architectures. In Chapter 4 I present a modification to the HTM Cortical Learning Algorithms (CLA), termed 2D-CLA. 2D-CLA adds limited 2D receptive fields to CLA to more closely model the way biological intelligence deals with high-dimensional "natural" data, e.g., audio and visual. I then describe several tests of the algorithm with relevance to both 2D-CLA and CLA. While the HTM theory has enjoyed some interest, to date there have been relatively few published works on the CLA. In Chapter 5 I present a predictive coding extension to the CLA algorithm, PC-CLA, which extends the algorithm by allowing it to be repeated recursively in a hierarchy (McCall & Franklin, 2013). I then present additional tests of the CLA, and initial tests of hierarchy with PC-CLA, which has not seen previous study.

In summary, while my research has several goals, it has produced several particular contributions to Computer Science and Cognitive Science:

- Development of the generic parts of the LIDA software framework allowing for rapid and highly-customizable implementations of LIDA software agents (Ch. 2)
- Conceptual modeling of motivation in much of cognition using the LIDA model. Motivation is a fundamental, often-overlooked aspect of cognitive architectures. (Ch. 3)
- Extends the computational LIDA software framework to support the conceptual motivation model (Ch. 6)
- Part of the proposed work will validate the motivational extension of LIDA via

the replication of an existing experimental result (Ch. 3)

- Conceptual modeling of high-dimensional perception in cognition necessary for intelligent agents with rich senses to succeed in complex environments (Ch. 4–5)
- Suggests a theoretical view (the free-energy principle) and accompanying algorithm (PC-CLA) that sees the processes of a systems-level cognitive architecture as performing general data assimilation based on patterns in time. This principled approach may help unify existing approaches by providing a core conceptual framework for mental processes (Ch. 4–5)
- Fleshes out a method for biologically plausible, limited, overlapping, 2D receptive fields for the Cortical Learning Algorithms (CLA) making the algorithm more suitable for processing visual stimuli (Ch. 4)
- Tests and evaluates some of the basic aspects of the CLA, previously undescribed by published work, including its sparse distributed representations, noise robustness, parameters and their effects, "boosting" mechanism, and memory quality (Ch. 4–5)
- Extends the Cortical Learning Algorithms, by adding predictive coding, to support hierarchical versions of the algorithm. This allows for hierarchical decomposition, which is crucial in managing complexity in data analysis and pattern recognition (Ch. 5)
- Extends the LIDA software framework to support high-dimensional pattern recognition, which was previously unsupported. This allows LIDA agents to cope with sensory data streams of much greater complexity (Ch. 6)

Dissertation Outline

The rest of this dissertation is organized as follows: Chapter 2 is the Background Chapter that introduces the LIDA model of cognition, and surveys models relevant to motivations and perception. The Artificial Motivations Chapter (3) presents a motivation model for LIDA, its computational implementation, and an experimental replication. Chapter 4 discusses 2D-CLA, its theoretical underpinnings, its applications for perception and learning in LIDA, and experiments exploring the basic properties of the algorithm. Chapter 5 covers PC-CLA, presenting a predictive coding extension to CLA termed PC-CLA and tests thereof. Chapter 6 proposes additional experimental work to be completed. In closing, Chapter 7 outlines directions for future research, and discusses the conclusions and contributions.

2 Background and Context

The LIDA Architecture and its Cognitive Cycle

The LIDA model is a comprehensive, systems-level, conceptual and computational model covering a large portion of human cognition[1]. Based primarily on Global Workspace theory (Baars, 1988), the model implements and fleshes out a number of psychological and neuropsychological theories. The LIDA model and its ensuing architecture are grounded in the LIDA cognitive cycle. The cycle is based on the fact that every autonomous agent (Franklin & Graesser, 1997), be it human, animal, or artificial, must frequently sample (sense) its environment and select an appropriate response (action). The agent's "life" can be viewed as consisting of a continual sequence of these cognitive cycles. Each cycle can be divided in three phases: understanding, attending, and acting. Neuroscientists call this tripartite process the action-perception cycle (Dijkstra, Schöner, & Gielen, 1994; Freeman, 2002; Fuster, 2004; Neisser, 1976). A cognitive cycle can be thought of as a moment of cognition, a "cognitive moment" (Snaider, McCall, & Franklin, 2012). Sophisticated agents, such as humans, process (make sense of) the input from such sampling in order to facilitate their decision-making and learning. Higher-level cognitive processes are composed of many of these cognitive cycles, each a cognitive "atom."

The LIDA model hypothesizes that the cognitive cycle has a rich inner structure, which I briefly review next. More detailed descriptions are available elsewhere (Baars & Franklin, 2003; Franklin, Baars, Ramamurthy, & Ventura, 2005; Franklin, Madl, D'Mello, & Snaider, 2013). During each cognitive cycle, a LIDA agent first makes sense

[1] "Cognition" is used here in an unusually broad sense, so as to include perception, feelings and emotions.

of its current situation as best as it can by updating its representation of both external and internal features of its world and itself. By a competitive process, as specified by Global Workspace Theory, it then decides what portion of the represented situation is the most salient, the most in need of attention. Broadcasting this portion, the current contents of consciousness, enables the agent to chose an appropriate action, and execute it, completing the cycle. Though GWT speaks to phenomenal consciousness, and thus to the "hard problem" of consciousness (Chalmers, 1996), the LIDA model follows Shanahan (2010) in taking the "post-reflective inner view" and doing "… without the habit of metaphysical thinking." More specifically, consciousness in the LIDA model refers to *functional consciousness* in which no assumption is made of the conscious broadcast being phenomenally conscious (Franklin, 2003).

Figure 1 shows the process in more detail. The cycle begins with the understanding phase, where incoming sensory stimuli from external and internal sources in the agent's environment activate low-level feature detectors in Sensory Memory. This begins the process of making sense of the incoming stimuli. These low-level features are passed on to Perceptual Associative Memory (also called recognition memory) where higher-level features, such as objects, feelings, actions, events, categories, relations, etc. are recognized. These features, which have been recognized preconsciously, make up the percept that passes asynchronously to the Workspace, where a model of the agent's current situation is updated. This percept serves as a cue to spatial memory, transient episodic memory, and declarative memory. Responses to the cue, termed local associations, consist of remembered contents from these memory systems that were associated with the various elements of the cue. In addition to the current percept, the

Workspace contains recent percepts and portions of the structures assembled from them, which have not yet decayed away. A new model of the agent's current situation is assembled from such percepts, their local associations, and the undecayed parts of the previous model. This assembling process will typically require structure building codelets. These are small, special purpose processors, each of which has some particular type of structure it is designed to build. To fulfill their task these codelets may draw upon Perceptual Associative Memory and even Sensory Memory, to enable the recognition of relations and situations. They may also draw on the Conscious Contents Queue, which stores the conscious contents of the past few tens of cognitive cycles (a few seconds of actual time). The newly assembled model forms the agent's understanding of its current situation within its world. It has made sense of the incoming stimuli.

For an agent operating within a complex, dynamically changing environment, this Current Situational Model (CSM) may well be much too much for the agent to consider all at once in deciding what to do next. It needs to selectively attend to a portion of the model, the most salient portion. Portions of the CSM compete for attention. These competing portions take the form of coalitions of structures from the CSM. Such coalitions are formed by attention codelets, whose function is to try to bring certain kinds of structures to consciousness. When one of the coalitions wins the competition, the agent has effectively decided on what to attend.

One purpose of this processing is to help the agent decide what to do next. To this end, a representation of the contents of the winning coalition is broadcast globally (hence the name Global Workspace Theory). Though the contents of this conscious broadcast

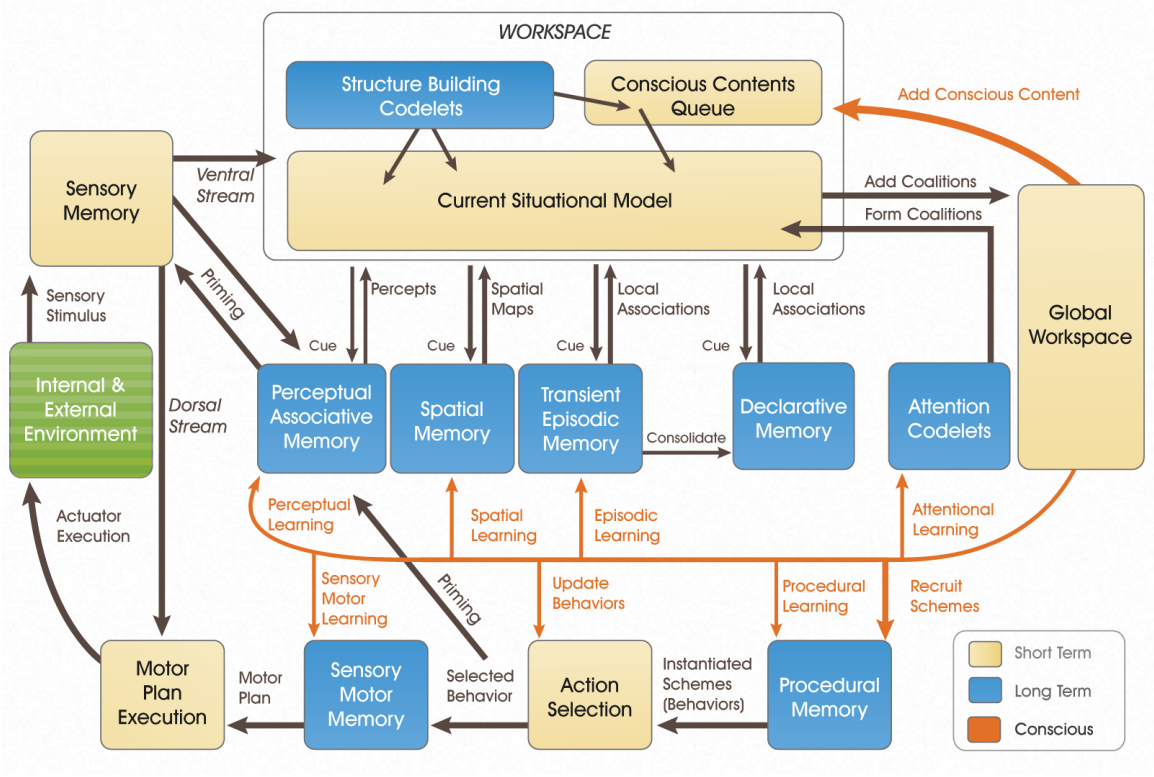


Figure 1. The LIDA cognitive cycle diagram.

are available globally, a primary recipient is Procedural Memory, which stores templates of possible actions including their contexts and expected results (Drescher, 1991). It also stores an activation value that attempts to measure, for each such template, the likelihood that an action taken within its context produces the expected result. Templates whose contexts intersect sufficiently with the contents of the conscious broadcast instantiate copies of themselves with their variables specified to the current situation. Instantiated templates remaining from previous cycles may also continue to be available. These instantiations are passed to the action selection mechanism (Maes, 1989), which chooses a single action from one of them. The chosen behavior then goes to Sensory-Motor

Memory, where an appropriate algorithm, called a motor plan, executes it. The action taken affects the environment, and the cycle is complete.

Another function of LIDA's cognitive cycle is to facilitate learning. LIDA's multiple modes of learning all occur continually, simultaneously, and online using each global broadcast of the contents of consciousness (Franklin et al., 2005; Franklin & Patterson, 2006). Perceptual learning is learning to recognize objects, actions, feelings, categorizations, relationships, events, etc., and the relationships among them. As new objects, categories, and the relationships among them and between them, and other elements of the agent's ontology are learned, nodes (objects and categories) and links (relationships) are added to Perceptual Associative Memory (Figure 1). Spatial learning refers to the building and updating of cognitive maps, which serve to locate objects in the environment (Derdikman & Moser, 2010; Madl, Franklin, Chen, & Trappl, 2013). Episodic learning is the encoding of information into episodic memory, the associative, content-addressable, memory for events — the what, the where, and the when (Baddeley, Conway, & Aggleton, 2001; Franklin et al., 2005). Relatively little studied by memory theorists, attentional learning refers to the learning of what to pay attention to (Kruschke, 2011). In the LIDA architecture, attentional learning is the learning of new attention codelets, and the updating and reinforcing of the existing ones (Faghihi, McCall, & Franklin, 2012). Procedural learning is the encoding of procedures, i.e., "what" to do "when," into Procedural Memory (Figure 1). It is the learning of new actions and action sequences with which to accomplish new tasks (Franklin, Madl, et al., 2013), and the reinforcement of old ones. Here we must distinguish between action selection and action execution. LIDA's Procedural Memory is composed of schemes concerned with the

selection of actions. Algorithms (motor plans) for the execution of actions are found in Sensory-Motor Memory where sensory-motor learning takes place.

The LIDA model hypothesizes that all human cognitive processing is via a continuing iteration of such cognitive cycles. These cycles occur asynchronously, with each cognitive cycle taking roughly 300–500ms (Madl, Baars, & Franklin, 2011). These cycles cascade, that is, several cycles may overlap having their currently active processes at different stages of the cycle running simultaneously in parallel. This cascading must, however respect the serial nature of conscious processing necessary to maintain the stable, coherent image of the world it provides (Franklin, 2005; Merker, 2005). This cascading, together with the asynchrony, allows a rate of cycling in humans of five to ten cycles per second.

The Nonlinear Dynamics Bridge to Neuroscience. The LIDA model has invoked nonlinear dynamics to bridge its nodes and links, perceptual symbols *a lá* Barsalou (1999), to neuroscience. This is realized by considering that such perceptual symbols represent "not neurons or cell assemblies, but, rather, wings of chaotic attractors in an attractor landscape" (Franklin, Strain, Snaider, McCall, & Faghihi, 2012). The notion of nodes as wings of chaotic attractors in an attractor landscape is due, in part, to Freeman and colleagues (Freeman, 1999; Skarda & Freeman, 1987). For instance, when perturbed by a previously learned exogenous stimulus such as one that may result from an inhalation, the spiking trajectory of a cell assembly, such as an olfactory bulb, falls into a wing of an attractor, and so recognizes an odor. LIDA postulates nonlinear dynamics as an intermediate theory useful in grounding comprehensive cognitive models such as LIDA in the underlying neuroscience (Franklin et al., 2012).

Friston and Kiebel (2009) share a similar view that the brain uses attractors to represent and predict causes in the sensorium. They suggest the brain models sensory data using a hierarchical dynamical (having temporal dynamics) network. In such a network, the states of a high-level attractor enter the equations of motion of (providing control parameters) of a low-level attractor, changing its trajectory, in a nonlinear way.

This form of generative model[2] has a number of sensible and appealing characteristics: 1) any level can generate (and therefore encodes) structured sequences of state variable patterns, as a level's state evolves along a trajectory. 2) The network can generate and therefore predict or represent different categories of sequences (Friston & Kiebel, 2011). This is because any low-level attractor embodies a family of trajectories with each member corresponding to a particular instance of a structured sequence. The current activity encoding the particular state at a particular time determines where the current dynamics are within the sequence, while the particular basin of attraction determines which sequence is currently being expressed. In short, the attractor manifold encodes *what* is being represented while the current neuronal activity encodes *where* the current representation is located on the manifold or within the sequence. 3) If the state of a higher attractor changes the manifold of a subordinate attractor (e.g., via a top-down prediction), then the states of the higher attractor come to encode the category of the sequence of dynamics represented by the lower attractor. This means it is possible to generate and represent sequences of sequences, and by induction, sequences of sequences of sequences, and so on. 4) Lastly, this suggests that the dynamics of high-level representations unfold more slowly than the dynamics of lower level representations,

[2] In probability and statistics, a generative model is a model for probabilistically generating observable data from some distribution, typically given some hidden parameters.

since the state of a higher attractor prescribes a manifold that can guide the flow of lower states (Friston & Kiebel, 2009).

The LIDA Software Framework

A *software framework (framework)* is a reusable implementation of all or part of a software system. In many cases, a framework constitutes the skeleton of the application, capturing its generic functionality. The framework specifies a well-defined application programming interface (API) that is implemented generically using abstract classes, interfaces, and generic, customizable module design. This hides the complexity of its code from the user. Most frameworks are based on object-oriented languages because the major properties of object orientation, data abstraction, inheritance, information hiding and polymorphism, complement the goals of frameworks. The core idea of a framework is to have a generic design as well as a base implementation of a complex software system. The user of the framework then only needs to "fill in the blanks" with problem or domain-specific elements. This is, perhaps, the major advantage of using frameworks: users can concentrate their efforts on the specifics of the problems, and reuse the generic mechanisms implemented in the framework. This also speeds up the development of the new application, and makes it less error-prone because part of the system has already been produced and tested.

The LIDA software framework (Snaider et al., 2011) is a generic and customizable computational implementation of the LIDA model. It is implemented in Java, a strong and proven object-oriented language. The main goal of this framework is to provide a generic implementation of the LIDA model, easily customizable for specific problems or domains. As mentioned before, this has several advantages: it speeds up the

implementation of new agents based on the LIDA model, and shortens the learning curve to produce such implementations. While the framework assists in the development of LIDA agents, it is generic enough to support in developing complex software systems with several modular pieces.

The framework permits a declarative description of the specific implementation. The full architecture of the software agent is specified using an XML formatted file; this is similar to other frameworks where the use of declarative description files are common (e.g., Walls & Breidenbach, 2005). In this way, the developer does not need to define the entire agent in Java; he or she can simply define it using the XML specification file.

Another important goal of the framework is its ready customization. The customization can be performed at different levels of detail. At the most basic level, developers can use the LIDA configuration file to customize their applications. Several small pieces in the framework can also be customized implementing particular versions of them. For example, new algorithms for decaying or for codelets can be implemented following a specified structure. Finally, more advanced users can also customize and change internal implementation of whole modules. In each case, the framework provides default implementations that greatly simplify the customization process.

The conceptual LIDA model suggests that minds operate in parallel. To complement this, the LIDA framework was developed with multithreading support. *Framework tasks*, encapsulations of short processes, along with a dedicated task manager, implement multithreading support that allows for a high level of parallelization. Lastly, the LIDA framework implementation adheres to the most important design

principles (Gamma, Helm, Johnson, & Vlissides, 1995), e.g., factory pattern and strategy pattern, and best programming practices (Snaider et al., 2011).

The LIDA framework defines several data structures and procedures (algorithms), and is composed of several pieces. Its main components include *Framework Modules (modules)*, which are interconnected elements representing modules in the LIDA model. Another main component is the *Framework Task Manager* that controls the execution of all processes in the framework. Most of these processes are implemented as Framework Tasks, which allows for multithreaded execution by the Framework Task Manager in a user-transparent manner. The *Node Structure* is a main data structure in the framework. It maintains a set of Nodes, a set of Links, and their connections to each other. Nodes, Links and other LIDA elements such as coalitions, codelets, and behaviors, have scalar activation attributes. In the LIDA framework, activations are updated (excited or decayed) using *strategies*. Strategies are implementations of the strategy design pattern, which allows for customizable behavior; in this case they specify the manner in which an activation value is updated. Lastly, the framework implements several supporting tools such as a customizable GUI, logging, and an architecture loader that parses an XML file with the definition and parameterization of the application, its data, and processes (Snaider et al., 2011).

The first version of the LIDA framework features several useful tools. The first is a customizable GUI consisting of a main GUI application and a series of GUI panels which display such things as the content of modules, running tasks, parameter values, etc. A properties file allows users to add their own GUI panels as well as configure which panels are used and where they appear in the GUI window. The Java logging API is used

throughout the framework, recording important activities as they occur. Every log entry is made with one of several levels of severity. A dedicated GUI panel for Logging is part of the standard framework GUI. It allows the user to view logs of particular levels for specific modules or all modules. An architecture loader allows agents to be specified via an XML file. The loader reads this file, and constructs an agent with modules, parameters, and initial tasks based on the file's specification. This utility obviates the need to specify agents by hand, and allows for quick interchange of modules, connections between modules, change of parameters, etc. The *element factory*, an implementation of the factory design pattern (Gamma, Helm, Johnson, & Vlissides, 1995), provides a centralized, configurable way to obtain new objects common in a LIDA agent implementation, e.g., Nodes, Links, and Codelets. The objects that the factory produces can be configured by XML, and, in addition, which strategies such objects use (Snaider et al., 2011).

The LIDA software framework allows the creation of new intelligent software agents, and the replication of experiments using software agents based on the LIDA model. Its design and implementation aim to simplify this process, and to permit to the user to concentrate on the specifics of the application, hiding the complexities of the generic parts of the model. It also enforces good software practices that simplify the creation of complex architectures. It achieves a high level of abstraction permitting several ways and levels of customization with a low level of coupling among modules. The framework also provides supplemental tools such as a customizable GUI and logging support. The result is a powerful and customizable tool with which to develop LIDA based applications and, perhaps, many others as well (Snaider et al., 2011). To date,

several LIDA agents using the framework have been successfully developed in support of published work (e.g., Faghihi et al., 2012; Madl et al., 2011; Madl & Franklin, 2012).

Models of Motivation and Emotion

James (1884) and Lange pioneered early emotions research (Cannon, 1927). In recent times several theories of motivation for agents and cognitive architectures have been developed. Here I briefly review a few of the more influential approaches, comparing them to LIDA's perspective where applicable.

Picard's seminal work on "affective computing" (1997, 2003) motivates current research concerned with artificial emotions in computer systems. Here, the goal is to develop systems to recognize, express, model, communicate and respond to human emotions in human-computer interactions. In affective computing, the various attributes of emotions can be studied separately, unlike in humans. For instance, a researcher might develop a computer agent that smiles without having the agent actually experience "happiness."

Fellous (2004) suggests focusing on the functions of emotions rather than what they are. He argues that one of the main functions of emotions is to achieve multi-level communication of simplified but high impact information. Fellous defines feelings as a subclass of emotions that involves some form of consciousness. He presents a framework, termed "neuromodulation," for understanding how emotions arise, are maintained, and interact with other aspects of behavior and cognitive processing. He argues that it may be crucial to understand emotions as dynamical patterns of neuromodulations, rather than patterns of neural activity. He maintains that emotion and "cognition" are integrated systems implemented by the same brain structures. "Emotion" is related to the state of

neuromodulation of these structure while "cognition" is related to the state of information processing. Given that LIDA's perceptual nodes, including those for "emotions" (see below) are hypothesized to correspond to particular basins of attraction in the dynamics of neuron populations (Franklin, Strain, Snaider, McCall, & Faghihi, 2012), the proposed feeling nodes (see below) can be viewed as being close to Fellous' vision.

One of the first intelligent agents that used "emotions" as a motivation mechanism was due to Cañamero (1997), who produced an autonomous agent with basic drives and an action selection mechanism modulated by its "emotions." Almost all cognitive architectures have motivation mechanism(s). For instance, SOAR (Laird, Newell, & Rosenbloom, 1987) has explicit goals, and an impasse mechanism that creates sub-goals for complex tasks. More recently emotions have been integrated into SOAR (Marinier, Laird, & Lewis, 2009), and used to drive reinforcement learning (Marinier & Laird, 2008), but not as motivators. The role of emotions of cognition has also been studied in ACT-R including its affect on rationality (Fum & Stocco, 2004), decision-making (Belavkin, 2001), and the "inverted U" effect (Cochran, Lee, & Chown, 2006).

Some cognitive architectures have sophisticated motivational mechanisms. For example, CLARION uses a bipartite motivational representation where "explicit goals", e.g., finding food, are generated based on past and current "internal drive states", e.g., being hungry. This explicit representation of goals derives from, and hinges upon, implicit drive states. Explicit goals may be generated on the fly during an agent's interaction with various situations. CLARION adopts (Sun, 2003) a "balance-of-interests" approach to goal setting where each drive "votes", with some strength (e.g.,0 to 9), for multiple goals. The goal receiving the highest total score is the winner. CLARION posits

a set of primary drives including food, water, sleep, avoiding physical dangers, reproduction, avoiding unpleasant stimuli, etc. (Sun, 2009). They are presumed to be evolutionarily hard-wired. Neural networks implement a "mapping" between world state and the strengths of drives. Such networks are trained offline, not unlike the process of evolution, which developed biological motivation. Reinforcement learning is used for online tuning of drive strengths. Sun also addresses high-level drives, e.g.,affiliation and belongingness, and personality within CLARION (2009). With such theoretical underpinnings, the CLARION architecture has modeled human performance degradation under pressure, which depends heavily on competing drives (Wilson, Sun, & Mathews, 2009).

Another architecture with a rich motivational mechanism is Psi (Dörner & Hille, 1995) and its progeny Micro-Psi (Bach, 2009). Psi distinguishes between demands, urges, motives, cognitive modulators, affect, and directed emotions. *Demands* are built-in drives that give rise to goals while *urges* are the signals that make demands apparent. The theory suggests: "an abrupt increase of an urge corresponds to… a 'displeasure or distress signal'… while a decrease of an urge — its satisfaction — yields … 'pleasure signals'" (Bach, 2012). A *motive* is defined as the combination of an urge and an event associated with the satisfaction of the urge. Goals depend on the nature of a given situation, and the existing demands. Higher-level emotions such as jealousy and anger are said to arise in Psi from the combination of motives and the agent's perception of a situation. Psi maintains that emotions should be distinguished from motivational phenomena, for example, while hunger is cognitively represented (as an urge) and implies a valence and an effect on the allocation of mental and physical resources of an organism, it is not an

emotion, but a *motivator*. The Psi model provides inspiration for some of the concepts in the motivation model I propose for LIDA. Later, in the Discussion section, I further explore the relationships between the two models.

Causal knowledge and its use in prediction are mentioned in our discussion of motivations. I assume that LIDA has the capacity to form and access temporal memory between *events* (defined below); however, this chapter is not meant to constitute a theory of causal learning (Danks, Griffiths, & Tenenbaum, 2003; Faghihi, Fournier-Viger, & Nkambou, 2011), and I will not discuss how causal memory might be formed in LIDA. Previous work on LIDA has approached high-level time production and representation (Snaider et al., 2012) and temporal memory writ broad (McCall & Franklin, 2013). Hume argued that causality is not based on actual reasoning: only correlation can actually be perceived (Morris, 2011). I adopt this stance for the LIDA model suggesting that causal knowledge is built from frequent, invariant temporal patterns among conscious events (Faghihi & Franklin, in preparation; Snaider et al., 2012).

Models of Perceptual Analysis and Learning

Here I review several inspirational perceptual analysis theories to provide background and motivation for the proposed Predictive Coding Cortical Learning Algorithms (PC-CLA) (McCall & Franklin, 2013). I feature theories with biological plausibility, more specifically, those imitating the cerebral cortex. Some aspects of some of these models are not of interest, e.g., the use of supervised learning is not of interest because it violates requirements of autonomy and agency.

HMAX (Riesenhuber & Poggio, 1999) is a biologically inspired model of the visual ventral stream in primates, which includes the initial feedforward processing that,

occurs in the initial 100–150ms of visual object recognition (Serre, Kreiman, et al., 2007). HMAX has shown state-of-art performance on object recognition and action recognition (Serre, Wolf, Bileschi, Riesenhuber, & Poggio, 2007). HMAX employs two simple operations, performed in alternating fashion over successive hierarchical levels: 1) a "max" operation, which selects the strongest input from a given set of inputs, which builds representational invariance, and 2) a template matching operation, which compares inputs with stored patterns or templates, building selectivity.

Leabra (O'Reilly & Munakata, 2000) stands for "Local, Error-driven and Associative, Biologically Realistic Algorithm," and is a model of learning that employs both Hebbian and error-driven learning. It is used to mathematically predict outcomes based on inputs and previous learning influences. Hebbian learning is performed using a form of the Principal Components Analysis algorithm. Error-driven learning is performed using an approximate recurrent backpropagation algorithm. Layer or unit-group level inhibition is computed directly using a *k*-winners-take-all function to produce sparse representations.

Deep learning networks employ a hierarchical architecture that is comprised of common circuits with similar (and often cortically influenced) functionality. The goal of such systems is to represent sensory observations in a manner that will later facilitate robust pattern classification, mimicking a key attribute of the mammalian brain. Arel, Rose, and Coop (2009) describe the Deep Spatio-Temporal Inference Network (DeSTIN) as a scalable deep learning architecture that relies on a combination of unsupervised learning and Bayesian inference. Dynamic pattern learning forms an inherent way of

capturing complex spatiotemporal dependencies. The network is proposed for high-dimensional signal classification.

Hierarchical Temporal Memory (George & Hawkins, 2009; J Hawkins & Blakeslee, 2004) is a theory of the primate neocortex, which postulates that the neocortex builds a model of the world using a spatio-temporal hierarchy. The most recent implementation of the theory is called cortical learning algorithms (CLA) (Hawkins, Ahmad, & Dubinsky, 2011). According to the theory, a tree-structured hierarchy built from a basic computational unit, called a cortical region, can approximate the operation of the neocortex. Each cortical region in such a hierarchy uses the same learning and inference algorithm, which entails storing co-occurrence patterns and then sequences of those co-occurrence patterns. The hierarchy is organized so that higher levels have 1) successively larger receptive fields in the original input space and 2) tend to produce representations exhibiting greater invariance over time. It is speculated that this kind of organization leads to efficient learning and generalization because it mirrors the spatio-temporal organization of the causes in the world (George & Hawkins, 2009). More recent accounts of HTM (Hawkins et al., 2011) stress the importance of sparse distributed representations, prediction, and inference for the recognition of patterns in noisy, time-varying data. However, to date, there has been no hierarchical implementation of CLA.

Rao and Ballard (1999) described a "predictive coding" view of the visual cortex, modeling it as a hierarchical network with units at each hierarchical level continually predicting the responses of the units in the next lower level via top-down, feedback connections. The lower level units then send back the discrepancies between the top-down predictions and the actual activity through feedforward connections. These

discrepancies, or error signals, are then used to correct the higher-level representation of the cause of the input signal, ideally improving the next top-down prediction. Lower levels have smaller receptive fields (in the input space and possibly in time), whereas higher levels have larger receptive fields, predicting and estimating signal properties at a larger scale by combining the responses of several lower level units (Huang & Rao, 2011).

Predictive coding provides an explanation for oriented receptive fields and contextual effects (Rao & Ballard, 1999), as well as the hierarchical reciprocally connected organization of the cortex. Predictive coding has also been found to be consistent with a variety of neurophysiological and psychophysical data obtained from different areas of the brain (Huang & Rao, 2011).

Generalized Filtering (GF) (Friston, Stephan, Li, & Daunizeau, 2010) is a recursive Bayesian estimation[3] scheme for nonlinear state-space models in continuous time. It provides a method for estimating the posterior or conditional probability density functions on the hidden states (current representations) and unknown parameters (e.g., connections implementing memory) generating the observed data, e.g., audio signals or human kinematics. Unlike classical filtering, e.g., Kalman-Bucy (1961) or Particle Filtering (van der Merwe, Doucet, de Freitas, & Wan, 2000), Generalized Filtering does not make Markovian assumptions about the time evolution of its models' random fluctuations (probabilistic components). Furthermore, the scheme operates online, assimilating data to optimize these conditional densities on states and parameters without the need for a backward pass. Another defining feature of GF is the formulation of this

[3] A general probabilistic approach for estimating an unknown probability density function recursively over time using incoming measurements and a mathematical process model.

optimization dynamically, in generalized coordinates[4] of motion. In this choice of generalized coordinate system, a point represents the trajectory, at a particular instant, of some variable (or variables) in the sense that it prescribes the variable's value, velocity, acceleration, and higher-order temporal derivatives. GF optimizes the conditional density with respect to a (approximating) variational free energy[5] bound on the model's log-evidence[6] using the generalized motion of hidden states and parameters. GF assumes that the motion of the parameters is small, which avoids assuming conditional independence between partitions of the model's parameters as is required by variational Bayes methods[7]. GF has been shown to outperform existing approaches, such as particle filtering, extended Kalman filtering (Puskorius & Feldkamp, 1991), and a similar approach using variational Bayes termed Dynamic Expectation Maximization (DEM) (Friston, Trujillo-Barreto, & Daunizeau, 2008).

Based Dynamic Expectation Maximization, Bitzer and Kiebel (2012) developed a model called recognizing recurrent neural network (rRNN), which fuses the recurrent neural network (RNN) approach with Bayesian inference techniques for nonlinear dynamical systems. In this model, units compute and exchange prediction and prediction error messages, implementing a predictive coding scheme for dynamic inputs. They demonstrate that the rRNN model has several desirable features over conventional RNNs including fast decoding of dynamic stimuli, and robustness to initial conditions and noise.

[4] A set of coordinates used to describe the state of a system relative to some reference configuration.

[5] An information theory measure that provides an upper bound for the Shannon surprise or self-information on sampling some data, given a probabilistic generative model.

[6] The natural logarithm of a marginal likelihood function, i.e., one where some variables have been marginalized (removed).

[7] A family of techniques for approximating intractable integrals arising in Bayesian inference.

They suggest that Bayesian model inversion (the process of inferring model parameters from observations) of recurrent neural networks may be useful both as a model of brain function and as a machine learning tool.

To summarize, I have briefly reviewed several models, many computational, that, taken together, in my opinion, call for the use of hierarchy, nonlinear dynamics, approximate Bayesian inference, and biological inspiration from the primate neocortex to confront the central problem of data analysis or data modeling. I take a closer look at these ideas and others in Chapters 4 and 5 where I present them as guiding principles for the Predictive Coding-Cortical Learning Algorithms (McCall & Franklin, 2013).

3 Artificial Motivations for Cognitive Software Agents

Introduction

Cognitive architectures are control structures for autonomous agents. An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda, and so as to affect what it senses in the future (Franklin & Graesser, 1997). Given this, how can the agenda of an autonomous agent be implemented in a cognitive architecture? How does such agenda arise from basic cognitive processes? These are the foundational questions I address in this work on artificial motivations in the LIDA model (McCall, Franklin, Snaider, & Faghihi, in preparation). Figure 2 outlines the parts of the LIDA model to which this work contributes.

The organization of this chapter is as follows: I first make some preliminary considerations concerning the representation of events and schemes in LIDA. Next, I review a series of motivation-related concepts, including feelings, affective valence, incentive salience, emotion, appraisal, reinforcement learning, and model-free and model-based learning. For each concept, I explain their relation to the LIDA model, and LIDA's implementation of the concept. In doing so, I invoke some existing aspects of LIDA, but also propose new faculties that extend the LIDA model. With this grounding, I then discuss the role of motivations in a single LIDA cognitive cycle, as well as over multiple cycles. To begin validating this motivational extension, I present a computational implementation of a LIDA-based agent with some of these capabilities, which replicates the results of an existing "reinforcer devaluation" experiment.

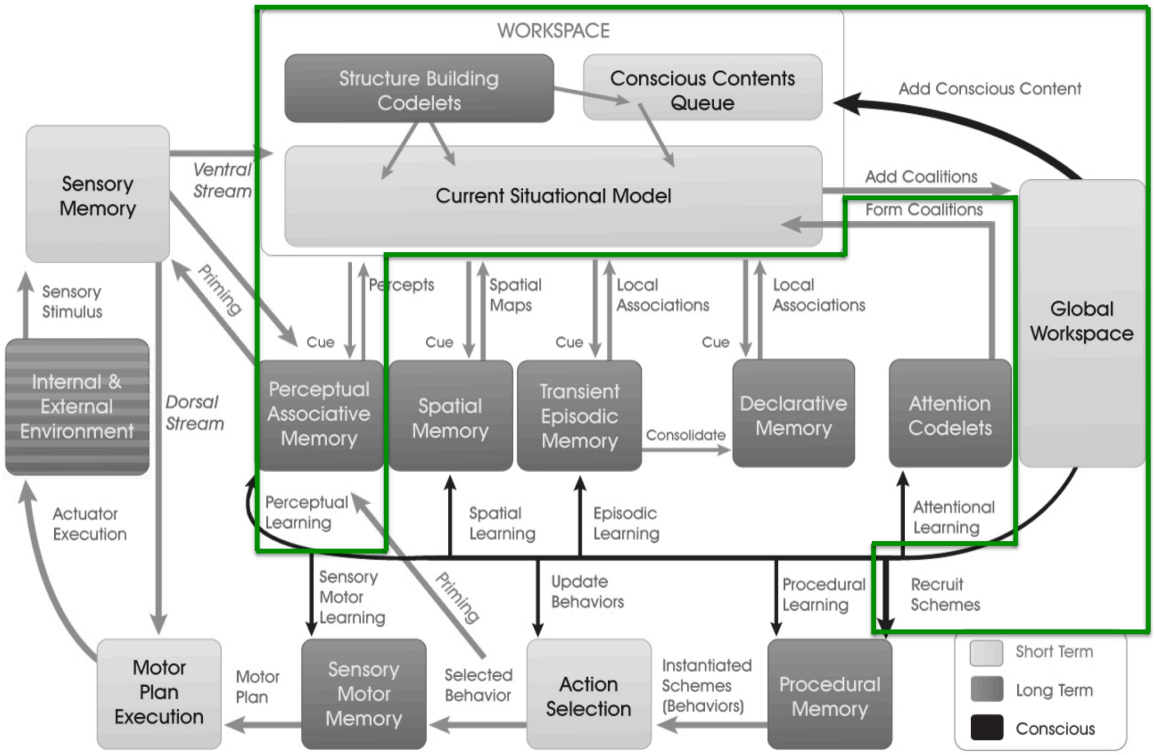


Figure 2. Contributions of Artificial Motivations Chapter to the LIDA model. The work in this chapter contributes additional constructs to the areas of the LIDA architecture outlined in green. This includes Perceptual Associative Memory, perceptual learning, the Workspace, coalitions, the Global Workspace, and scheme recruitment.

Event Representation in LIDA. A recent development in the LIDA model involves the representation of events (Zacks, Speer, Swallow, Braver, & Reynolds, 2007; Zacks & Tversky, 2001). Previously, events were represented by an action node, a perceptual symbol, with thematic-role links connected to role-filling nodes (McCall, Franklin, & Friedlander, 2010). Here I suggest maintaining the use of action nodes with thematic roles; however, an additional more abstract *event node* should be added encompassing the action node and its roles, and representing the entirety of the event. In this view, the action node is a central, but not complete, part of an event. For example, suppose the event "Charles takes the pen" is recognized in PAM, and becomes part of the current

percept. Then it should appear as shown in Figure 3 in the Current Situational Model of the Workspace.

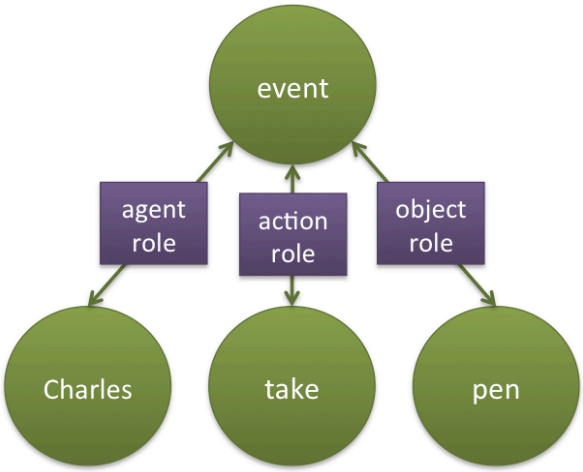


Figure 3. The representation of the event "Charles takes a pen." Double-headed arrows indicate bidirectional information passing. The labels are for illustrative purposes and play no role in LIDA's dynamics.

With this representation it is clear how events can be recreated by passing activation "top-down" from the event node. Furthermore, since the single event node effectively encapsulates the entire event, the event can become a part of a higher-level structure in a straightforward manner. For example, a link between two event nodes can express a temporal relationship between the two events. For the purposes of this chapter I will assume that learned or newly recognized events, while involving several nodes providing grounding, are effectively encapsulated, and can be referenced by their event node.

Network Models of Representation and LIDA. Network models are predicated on the basic tenet that cognitive representations consist of widely distributed networks of cortical neurons. Cognitive functions, namely perception, attention, memory, language, and intelligence, consist of neural transactions within and between these networks. Fuster (2006) presents a network model of cortical representations called the "cognit." The cognit model purports that all cognitive representations (all items of memory and knowledge) consist of networks of cortical neurons that have been associated by experience, whether that is the experience of the *species* or the experience of the *individual organism*. Cognits are of an immense variety, in terms of size — that is, number and cortical distribution of the neurons that form them — and in terms of content. The content of a cognit derives strictly from its component neurons and, above all, the relationships (connections) between them. Thus, the memory code is essentially a relational code. The content of an individual memory is determined by the combination of neurons that make it. The seemingly infinite variety of individual memories derives from the practically infinite combinatorial power of some 10 billion neurons (Fuster, 2006).

Perceptual cognits are cognitive networks made of neurons associated by information acquired through the senses, while executive cognits are made of neurons associated by information related to action. In both cases cognits are hierarchically organized. At the bottom of the hierarchy, cognits are small and relatively simple, representing simple percepts or motor acts. At the top cognits are wider, and represent complex and abstract information of perceptual or executive (motor) character, or both. Perceptual and motor networks are associated with each other in cortico-cortical

connections. These connections support the dynamics of the perception-action cycle in sequential behavior, speech, and reasoning (Fuster, 2006).

Inspired by the cognit model for cortical representation LIDA is moving towards (McCall & Franklin, 2013) a network view of mental representation. That is to say, we can (and should) view LIDA's perceptual and motor representations as vast, distributed network-based representations. One implication is that all but the most trivial of LIDA's representations, will be stored as distributed patterns across a hierarchy generated by an individual memory with each pattern at a different level of abstraction, from features to objects to events and beyond (see Chapters 4 and 5). Another implication is that there is clearly no room for copies (in the computational sense) of long-term representations. By this I mean that complex memories (e.g., events), which involve simpler memories, should "point back" to their constituent parts. These stances are in accordance with situated and embodied cognition to which LIDA has long adhered (Franklin et al., 2012).

Scheme Representation in LIDA. LIDA's Procedural Memory is a critical bridge from the motivational primitives introduced in this chapter to action selection. Procedural Memory provides long-term storage of ideas of what to do in various circumstances. Each such datum is called a *scheme*, inspired by Drescher's schema mechanism (1991). A scheme in LIDA consists of a "context" and a "result," both perceptual symbols, along with an "action," which connects to "an appropriate network in sensory-motor memory" (D'Mello, Ramamurthy, Negatu, & Franklin, 2006). Under this account, a scheme also maintains a base-level activation, and, when instantiated as a behavior, a current activation as well. The current activation measures "the relevance of the scheme to the current situation (environmental conditions, goals, etc.)," while its base-level activation is

an estimate of "the likelihood of the result of the scheme occurring as a result of taking the action within its context" (D'Mello et al., 2006).

One point of clarification needed regarding this earlier description of schemes concerns the "action." This entity should not connect directly to Sensory-Motor Memory. Instead, the "action" should refer to the perceptual symbol that is based on the internal and external sensory stimuli that accompany the execution of the action (e.g., for the "grabbing" action, the internal stimulus of one's hand grabbing an object, and the external stimulus of seeing one's hand grabbing). Figure 4 depicts a scheme that includes an action event. To be clear, there can still be associations between a scheme and motor plans, however, these are considered to be a part of Sensory-Motor Memory, not a part of the scheme.

I agree with the earlier stance that the current activation of an instantiated scheme should derive from "environmental conditions." Specifically, each of the scheme's context, action, and result contributes to the instantiated scheme's activation based on their total activation. Later in this chapter I propose how "goals" can impact a scheme's current relevance. However, this impact will come in the form of *incentive salience* (defined below), a separate quantity from current activation.

The final attribute of schemes to discuss is base-level activation. How might scheme base-level activation be implemented? One possible answer comes by noting that the context, action, and result events of a scheme will likely also be the events of an event sequence in PAM, that is, a recognition memory of a temporal pattern of events connected by temporal links. If the temporal links of such a sequence encode the likelihood, for a given context, that the link's source event leads to its sink event, then the

scheme's base-level activation could be based directly on the base-level activation of these temporal links. The final advantage of this view of schemes as action sequences in PAM is parsimony. The same event sequence is useful for recognition (i.e., it's in PAM), and, if the sequence involves a context, action, and result, it also comprises procedural memory, without the need to store the memory twice in two different memory systems.

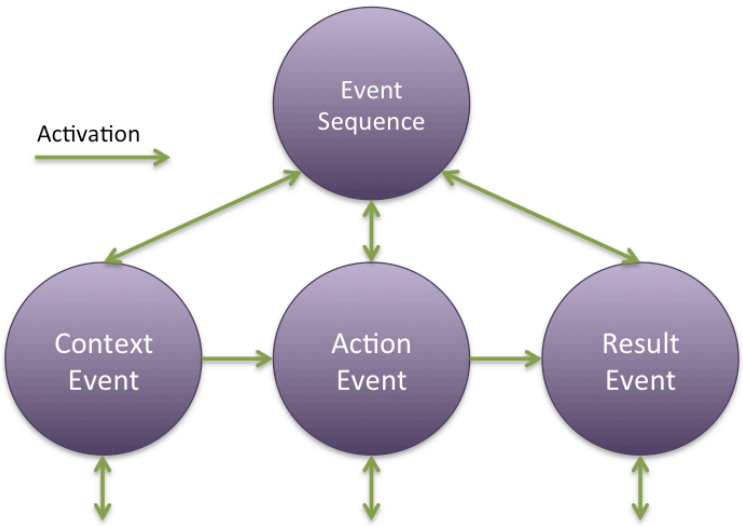


Figure 4. A depiction of a single scheme as proposed in this chapter. Each of the purple nodes can be thought of as perceptual symbols in LIDA's PAM.

Motivational Concepts

This section gives several definitions of motivational concepts from other researchers. I explain how each concept is realized within the LIDA model, and in doing so, hope to bridge LIDA to existing concepts facilitating comparison and comprehension. I do not intend to offer LIDA as a comprehensive model of all motivational and emotional phenomena, but rather as a useful tool for thinking about motivation and emotion, and as a possible architecture for implementing them in software agents.

Homeostatic Receptor. Biological agents use *homeostatic control mechanisms* to regulate their internal environment, keeping internal variables within stable ranges (Cannon, 1929). More recently they have been used to implement drives in robots (Breazeal (Ferrell), 1998). Each homeostatic control mechanism has at least three interdependent components for the variable being regulated. The *receptor* is the sensing component that monitors and responds to changes in the internal environment. When the receptor senses a stimulus, it sends information to a "*control center*," the component attempting to keep the variable within a stable range. The control center then sends signals to *effectors*, which can be muscles, organs, or other structures, that receive signals from the control center (Marieb & Hoehn, 2007). Homeostatic receptors then are sensors, which detect the value of critical attributes in an agent's bodily state. Examples of these in humans include sensors for low levels of oxygen or glucose in the blood. The detection of low oxygen and/or high carbon dioxide in the body sends a signal to the brain (control center). From this point, the brain must interpret the signal. Such interpretation gives rise to a *feeling* of, for example, the need to breathe more rapidly. A sensor may be implemented in many ways, for instance, with distributed inputs from all over the body or by taking inputs a localized bodily region.

In LIDA, homeostatic sensors are implemented as sensors, and are considered to be a part of Sensory Memory (see Fig. 1). A particular feature detector begins the control process detecting an incoming internal (or external) stimulus, and activating an associated feeling node in Perceptual Associative Memory. The control center concept is effectively described throughout the rest of this chapter including the aforementioned mechanisms of LIDA's cognitive cycle.

Feeling. Damasio (1999) views feelings as somatic markers, the perception of a certain state of the body along with the perception of a certain mode of thinking, and thoughts with certain themes (Damasio, 2003). Franklin and Ramamurthy (2006) also adopted this view writing:

> Feelings in humans include hunger, thirst, various sorts of pain, hot or cold, the urge to urinate, tiredness, depression, etc… One feels feelings in the body. Implemented biologically as somatic markers, feelings typically attach to response options and, so, bias the agent's choice of action… Feelings, including emotions, are nature's means of implementing motivations for actions in humans and other animals. They have evolved so as to adapt us to regularities in our environments. These general preferences derived evolutionarily from regularities can be viewed as values. Thus feelings become implementations of values in biological agents, providing a common currency for quick and flexible action selection.

I adopt this notion of feelings for this work. Feelings are implemented as PAM nodes in LIDA. They always have exactly one valence sign, either positive or negative (see below). Given that feelings are implemented as PAM nodes, it is also critical to distinguish two types of feeling nodes, which will prove useful later on:

1) ***Drive feeling node****, a feeling node representing a drive of the agent.*

2) ***Interpretive feeling nodes****, a feeling node representing a perceptual interpretation of a stimulus.*

The former is a representation of the current internal state of the agent's body, independent of the current external environment, while the latter is the agent's evaluation of a stimulus regardless of whether it is external or internal to the agent. For example, a "hungry" feeling node represents a drive, and has negative valence sign, while a "satiated" feeling node also represents a drive but with positive valence sign. A "sweet" feeling node represents a perceptual interpretation of some stimulus with positive valence

sign, while a "bitter" feeling node is similar, but with negative valence sign. Intuitively, one may be hungry or satiated in a variety of external situations, and one may interpret a variety of different stimuli as sweet or bitter. Again, the importance of this distinction becomes evident later in the chapter.

Affective Valence. Affective valence is the basic hedonic niceness or nastiness, "liking" or "disliking," of reward or aversion that is essential to emotions (Berridge & Kringelbach, 2008).

*For LIDA, I propose that each feeling node has a **valence sign**, having either a positive or negative value. With a positive value, the sign stands as the agent's basic representation of niceness or liking, while a negative value represents basic nastiness or disliking. For example, the feeling node "thirst" would have a negative valence, while the feeling node "relief from thirst" would have a positive valence. A feeling node's valence sign plus its current activation specifies its **affective valence**, with the activation representing the magnitude of "liking" or "disliking." Thus, a strongly activated feeling node having a positive sign represents strong "liking," while a weakly activated one represents weak "liking."*

*Like all PAM nodes, feeling nodes must be activated from the sensory stimulus to be instantiated in the Workspace. Said in another way, feature detectors in the feeling node's receptive field must detect features of the sensory stimulus. Either by direct recognition, due to earlier perceptual learning, or by close temporal association, feeling nodes may become part of preconscious event representation(s) in the Workspace. Such events will be referred to as **feeling events**, a designation likely true of most all events. The feeling node can be aptly thought of as a feature of the event. This suggests that*

recalled or imagined feeling events may also involve some degree of affective valence. By having different feeling nodes carrying positive or negative affective valence, it is also possible to both "like" and "dislike" the same event structure. All that would be required is for two feeling nodes with positive and negative affective valence respectively to be a part of the same active event structure.

Liking vs. Wanting and Disliking vs. Dreading. Kringelbach and Berridge (2009) distinguish "wanting" from pleasure — "liking." By analogy, "dreading" should be distinguished from displeasure — "disliking." "Wanting" is an objective hedonic reaction, measured behaviorally or neurally, whether or not accompanied by conscious pleasure. Core "liking" reactions result from activity in identifiable brain systems that paint hedonic value on a sensation such as sweetness (Berridge & Kringelbach, 2008). There is evidence of separate neural representations for wanting and liking (Smith, Berridge, & Aldridge, 2011). Dread appears to be the opposite of wanting or desire, as neurotransmitters in the nucleus accumbens have been shown to modulate whether the area generates desire (wanting) or dread (Richard & Berridge, 2011).

For LIDA, as previously mentioned, a node's affective valence is the product of its valence sign and current activation, and is either positive (representing liking) or negative (representing disliking). This is clearly distinguished from a node's ***incentive salience*** *attribute (see below), a scalar ranging from negative (dreading) to positive (wanting). Both the affective valence and incentive salience must be further distinguished from current activation, which measures the current salience of a node, and base-level activation, which measures the learned usefulness of a node in general. Note that having positive incentive salience refers to a potential or proposed goal, not necessarily the*

agent's current goal. The relationship between wanting and goals is discussed in the section below on options.

Incentive Salience. Incentive salience is a motivational "wanting" attribute given by the brain to stimuli transforming the representation of a stimulus into an object of attraction. This "wanting" is unlike "liking" in that liking is produced by a pleasure immediately gained from consumption or other contact with stimuli, while the "wanting" of incentive salience is the motivational magnet quality of a stimulus that makes it a desirable and attractive goal, transforming it from a mere sensory experience into something that commands attention, induces approach, and increases the likelihood of its being sought out (Berridge & Robinson, 1998).

Roughly, stimuli have incentive salience, i.e., are "wanted," only if they have been previously "liked." However, it has also been demonstrated in biological agents that physiological signals, e.g., hunger, are *integrated* with a stimulus' incentive salience (Tindell, Smith, Berridge, & Aldridge, 2009). Thus, overall incentive salience is adaptable as its learned component is modulated by current physiological signals, a view originally articulated by Bindra (1978) and Toates (1986), and later referenced in the incentive salience model (Berridge & Robinson, 1998). Such a combined measure contributes to an event's *decision utility* (Berridge & Aldridge, 2008), the valuation of an outcome that determines choice (Kahneman, Wakker, & Sarin, 1997). Zhang et al. developed a "neurocomputational" model of incentive salience, which combines "learned reward components" with "current physiological state" to model salt appetite and addiction (Zhang, Berridge, Tindell, Smith, & Aldridge, 2009). In the LIDA model,

activation and incentive salience implement this so-called decision utility as they alone determine which actions will be selected.

Inspired by all of this, I implement incentive salience as a distinct attribute of event nodes having both a base level[1] and a current component. This attribute is similar to, but separate from, node activation. Activation represents a node's salience, which may involve relevance, importance, urgency, insistence, novelty, unexpectedness, loudness, brightness, motion, etc. While activation represents an event's situational salience, incentive salience represents the value the agent places on an event actually occurring. Incentive salience may take on a range of values from positive (e.g., 1) where it represents the degree that an event is "wanted," to negative (e.g., -1) representing the degree of aversion to the event. Just as a node's total activation is a function of its base-level activation (learned usefulness) and current activation, a node's total incentive salience combines its base-level incentive salience and current incentive salience. Base-level incentive salience is modified in perceptual learning (see Base-Level Incentive Salience Learning below); events gain incentive salience by being associated with feelings with positive affective valence. Likewise, their incentive salience decreases when it is associated with feelings having negative affective valence. Current incentive salience comes from activated drive feeling nodes (representing physiological drives), connected by an incentive salience link (see Incentive Salience Link Learning below) to an event. The details of such a link including how it is learned are discussed later in the Perceptual Associative Memory Learning section. Both the total activation and the total incentive

[1] PAM is recognition memory, and a PAM node having some base-level incentive salience is not sufficient for recognition. However, base-level incentive salience is a fundamental attribute of nodes, and, as such, is considered a part of PAM.

salience of an event contribute to its salience, which makes the event more likely to come to consciousness.

Option. An *option for action (option)* (Keller & Ho, 1988; Klein, Wolf, Militello, & Zsambok, 1995; Raab, de Oliveira, & Heinen, 2009; Ward, Suss, Eccles, Williams, & Harris, 2011) is an action representation an agent can select for execution. It is a candidate for action or a "choice alternative" (Kalis, Kaiser, & Mojzisch, 2013).

So far, I have discussed feeling events; those events with one or more associated feeling nodes. Feeling events have an overall affective valence, which is based on the aggregate affective valence(s) of the feeling node(s) associated with (linked to) the event. In the last subsection, I proposed that events also have incentive salience. In fact, I hypothesize that most every event will involve some feelings and some nonzero amount of incentive salience. Event memories in PAM can have base-level incentive salience; however, they do not have current incentive salience.

Given this, I define a ***virtual event*** *as an event instantiated despite the absence of the event's corresponding sensory signature with respect to the agent's sensors. For example, recalled episodic memories, plans, imaginations, options, and hallucinations are all virtual events that may occur in absentia. Based on this, I define an* ***option*** *as an instantiated virtual event in the Workspace with some nonzero total incentive salience. Thus, options are unique in that their salience is based on an event's total incentive salience, and, possibly, the event's activation (e.g., from stimuli) as well. Options can be thought of as possible "choices" that have not yet been made. When an option helps instantiate a scheme likely to bring it about, that is, a behavior with the option as its result, and if such a behavior is selected, then the option becomes an* ***immediate goal****. A*

behavior stream is a stream of behaviors that can be thought of as a partial plan of action. Given this, I define a ***pending goal*** *as an option that is 1) a part of a behavior stream selected by a volitional decision-making process (ideomotor theory) and 2) cannot be immediately achieved.*

Emotion. The usage of the term "emotion" has been imprecise and, for many researchers, the concept is not easily definable (Alvarado, Adams, & Burbeck, 2002; Thompson & Madigan, 2007). Various definitions of emotion have been given, and very important functions have been attributed to emotion; however, there is currently no consensus for one definition. Some well-known work on emotions includes the six basic emotions: surprise, fear, disgust, anger, happiness, and sadness (Ekman, Sorenson, & Friesen, 1969). They are particular to individuals (Picard, 2003), and influence cognition directly (Phelps, 2006; Squire & Kandel, 2000). More recently some researchers deny that there is anything "basic" about these six (Camras, 2011). Westen (1999) describes emotions as a mix of several biochemical, sociocultural, and neurological factors. Purves et al. (2008) divides emotions into three processes: 1) a behavioral action (e.g., escape), 2) a conscious experience of an event or situation (e.g., regret or shame), 3) a physiological expression (e.g., paleness, blushing, palpitations, uneasiness). It is not specified how these three processes are related. Diener (1999) suggested that a theory of emotions should account for several components: 1) subjective experience (how it feels to be in an emotional state), 2) physiological processes (neural, neurochemical and physiological mechanisms that facilitate emotion), and 3) expressive behavior (facial and bodily expression, movement patterns, the modulation of voice and breath etc.), and 4) cognitive evaluations.

The LIDA model does not address subjective experience; rather its consciousness is only postulated to be functional[2] (Franklin, 2003). While "cognitive evaluations" are addressed in the next section, the physiological and expressive aspects of emotions are beyond the scope of this current work, though they could, in principle, be added.

In LIDA, emotions exist only within the context of a being, or an agent. It is a feeling towards ***something*** *or* ***someone*** *that is an emotion. For example, the emotion of* ***feeling*** *sad because one has suffered a loss, or the emotion of* ***feeling*** *angry at someone because they interfered with something you want to do. I adopt such a definition of emotion for the LIDA model. This stance was previously described by Franklin and Ramamurthy (2006) who wrote: "Emotions, such as fear, anger, joy, sadness, shame, embarrassment, resentment, regret, guilt, etc., are taken to be feelings with cognitive content (Johnston, 1999). One cannot simply feel shame, but shame at having done something. The something done constitutes the cognitive content." A feeling node in PAM would represent shame. A currently active association of an event node, the cognitive content, with the shame node within the Workspace would constitute an instance of the emotion shame. Once again, such an emotion may, or may not, come to consciousness. Shame, of course, may come in multiple varieties, each represented by its own node. Note that every emotion is a feeling, but there are feelings, e.g., thirst, that need not be emotions. Thus emotions constitute a subset of feelings.*

Appraisal. Appraisal theory is described as providing a cognitive interpretation of what we sense or perceive (Lazarus, 1991). Moreover, this theory explains our evaluation of

[2] Global Workspace Theory, which LIDA implements and fleshes out, is a high-level theory of the role of consciousness in cognition. It assumes that phenomenal consciousness occurs in conjunction with each conscious broadcast. Wishing to apply our LIDA model to both biological and artificial agents, the model remains agnostic on the question of phenomenal consciousness.

specific external (environmental) or internal (within ourselves) stimuli that give rise to emotions. Roseman and Smith (2001) explained how human motives and goals play an important role for the evaluation of a specific situation. Lazarus (1991) identified two appraisal methods: 1) primary appraisal, for the organism to determine the importance or meaning of an event, and 2) secondary appraisal, for the organism to determine whether or not it can manage or handle the event and its consequences. Lazarus also discusses two types of appraisal models: the structural appraisal model and the process model.

The structural model has three components: 1) a relational aspect, involving the person-environment interactions, 2) a motivational aspect, involving the assessment of a current situation's relevance in relation to one's current goals, and 3) a cognitive aspect involving one's appraisal of the situation (Lazarus, 1991). Accordingly, one's evaluation of the situation leads to specific emotions.

The process model suggests that person-environment interactions might go through an appraisal-coping-reappraisal cycle (Lazarus, 1991). For example, imagine a father becomes angry (appraisal) after hearing of his daughter's disobedience. He then has a heated argument with her (coping). Later on, he may feel guilty about his behavior during the argument (reappraisal), which, in turn, may lead to an apology (coping). This, finally, leads to a feeling of satisfaction at having apologized (reappraisal). However, the process model of appraisal fails to account for automatic emotional response, a fast and direct pathway for action in critical situations. I will discuss how this occurs in the LIDA model in detail in the section below on *alarms*.

In LIDA model terms, appraisal is an initially unconscious process occurring automatically with perception, and may also involve episodic and semantic recall,

imagination, and consciousness. I consider the appraisal process as one that evaluates a situation (event) along dimensions such as novelty, goal-alignment, agency, coping potential, and availability of a plan, thereby determining the activation level of relevant feeling nodes. In this sense the appraisal process culminates in the attachment of an activated feeling node to the cognitive content, typically an event, producing the corresponding emotion. This is typically accomplished in LIDA's Workspace by the preconscious association of an event and the emotion(s) activated during the event's appraisal. Such association is performed by structure building codelets concerned with particular appraisal dimensions. The attachment of the feeling node modulates the current activation of the event. If this newly appraised structure, including the feeling node, wins the competition for consciousness, this emotion, this feeling event, is learned into PAM (long-term recognition memory). Once learned, emotions, like other feelings can be recognized in PAM without additional appraisal. Often times the appraisal of emotion is a multi-cyclic affair requiring at least several cognitive cycles. However, I hypothesize that the initial appraisal of emotion can also occur during a single LIDA cognitive cycle.

Reinforcement Learning. Reinforcement learning typically refers to learning what to choose to do — how to map situations to actions so as to maximize a numerical reward signal (Sutton & Barto, 1998). The learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics — trial-and-error search and delayed reward — are the most

important distinguishing features of reinforcement learning (Sutton & Barto, 1998).

In LIDA terms, the learning involved in reinforcement learning is most closely related to procedural learning. A potential qualm LIDA might have with the standard form of reinforcement learning is the numerical reward signal aspect. For LIDA, and for autonomous agents in general, what reinforcement learning calls reward must come from the agent's perception of its current state, including its bodily state. If such perception is appraised with positive affective valence, i.e., if it is liked, it can then be viewed as a "reward." Note that such assignment of reward, being preconscious, necessarily precedes conscious learning in the LIDA model. In fact, if the reward lacks salience, it may not reach consciousness, and thus will not be learned. Another possible point of departure is LIDA's procedural learning, which adds new schemes to its Procedural Memory based on the consequence of actions in various contexts. Such temporal knowledge can later be used for reasoning and planning (see Deliberation and Model-Based Control below).

As denoted by the orange arrows in Figure 1, the LIDA model supports multiple, simultaneous learning modes, including perceptual, spatial, episodic, attentional, procedural, and sensory-motor. Each of the various modes of learning in the model follows the ***principle of profligacy****. This principle suggests that new representations are added to the various memories profligately, at the slightest justification, that is, whenever they come to consciousness, and left to survive or not. The principle of profligacy is often referred to as "generate and test" because multiple representations are generated but very few survive. Repeated conscious learning of a given representation over time can increase the base-level activation of the representation allowing it to survive. However,*

all memories and active representations decay incrementally during each cognitive cycle and, are removed from memory if their base-level activation drops to zero.

Model-Free Learning. Lucantonio, Stalnaker, Shaham, Niv, & Schoenbaum (2012) succinctly describe model-free learning (Daw, Niv, & Dayan, 2005) as a set of reinforcement learning methods that use prediction errors to estimate and store scalar cue or action values from experience. In reinforcement learning, these stored values represent the "predicted total future reward if an action or cue is pursued." Agents may use such stored values to guide action selection. The name "model-free" is based on the fact that choices are made only based on this stored value, and not on a model incorporating the details of the current situation or possible future consequences. Such a decision strategy is "simple but inflexible" because the values are merely scalar numbers, "separated from the identity of the expected future outcomes themselves or the specific events that will ensue en route to obtaining the outcomes." In particular, such cached values do not immediately change if the outcomes are revalued, e.g., a cue, typically followed by a reward, suddenly begins to lead to a punishment. Instead, such a change to stored values requires repeated experience and learning of new cached values by means of prediction errors. As a result of this inflexibility, it has been postulated that decision-making using cached values underlies habitual behavior (Lucantonio et al., 2012). Model-free approaches have been invoked to account for the activity of dopamine neurons and their (notably dorsolateral) striatal projections (Houk, Adams, & Barto, 1995; Schultz, Dayan, & Montague, 1997).

In the motivation extension proposed for LIDA, the cached values updated in model-free learning directly correspond to the base-level incentive salience of event

nodes, which is modified (learned) during conscious learning. Model-free control (action selection) occurs when a LIDA agent selects an option for execution only based on its total activation and base-level incentive salience, ignoring the consequences. This type of decision-making in LIDA is referred to as ***consciously mediated action selected*** *since conscious access is required for content to affect action selection. For example, an agent might have the event node, "open-the-refrigerator-door," with a strong base-level incentive salience due to its relation with food. This leads to agent to habitually open the refrigerator door, even when the power is out and the agent desires to keep the refrigerator closed to preserve its temperature.*

Model-Based Learning. Lucantonio et al. (2012, p. 360) also describe model-based learning (Daw et al., 2005) briefly as:

> A set of reinforcement learning methods in which an internal model of the environment is learned and used to evaluate available actions or cues on the basis of their potential outcomes. In these methods, values are not learned incrementally through prediction errors and stored for future use, but rather are computed 'on the fly' when needed, by mental simulation of sequences of events and outcomes using the internal world model. All that is required is a world model that includes predictions about the immediate consequences of each action or state in a sequence. This approach to action selection is computationally expensive owing to the need to mentally simulate and 'test' alternative courses of action, but it allows one to flexibly adapt behavior in a changing environment, and specifically, to adapt immediately to changes in the current value of the outcome.

Model-based methods are identified neurally with a prefrontal cortex system (Daw et al., 2005).

In LIDA's Workspace, the Current Situational Model (CSM) contains the agent's current model of the environment including possible future events. I propose that such anticipated events may be part of the content of a coalition brought to the Global Workspace by attention codelets and, that, as a part of a coalition, their total activation

and total incentive salience determine the activation of the coalition and thus its likelihood of winning the competition for consciousness. Over multiple cognitive cycles, LIDA may deliberate over various coalitions of possible actions and their expected outcomes based on the coalitions' activation. This type of decision-making is referred to as ***volitional (deliberative) action selection*** *in LIDA. How LIDA implements model-based control is the subject of further discussion in the sections "Learning Event Sequences" and "Deliberation and Model-Based Control" below.*

Summary. Table 1 summarizes the major concepts introduced in this Motivational Concepts section and describes their suggested implementation in the LIDA model.

Table 1. Motivational concepts introduced in this section and their implementation in the proposed motivation extension to the LIDA Model.

Concept	**Brief Description**	**LIDA's implementation**
Homeostatic receptor	A sensing component that monitors and responds to particular changes in the internal environment.	An agent-dependent sensor within Sensory Memory.
Affective valence	An attribute representing the basic hedonic niceness or nastiness of stimuli.	A scalar attribute of all feeling nodes determined by the product of the feeling node's valence sign and current activation.
Feeling	A representation of a stimulus (internal or external) having a valence, which serves to implement one or more of the agent's values.	A particular type of node in PAM, called a feeling node, each with a valence sign, either positive or negative.
Virtual Event	An active event representation in absence of its external counterpart.	An instantiated event node in the absence of its corresponding sensory signature.
Option	A candidate for action or a choice alternative.	An instantiated virtual event in the Workspace with nonzero total incentive salience.
Immediate Goal	The current actionable goal of the agent.	An option that is the expected result of a selected behavior.

Table 1. (Continued).

Concept	Brief Description	LIDA's implementation
Incentive Salience	The motivational magnet quality of a perceptual representation making it a desirable and attractive goal, transforming it into something that commands attention, induces approach, and causes it to be sought out.	An attribute of event nodes, distinct from activation, that contributes to a node's overall salience. PAM nodes have a base-level incentive salience while their instantiations additionally have a current incentive salience.
Liking vs. Wanting	The distinction between "wanting," an objective hedonic reaction and "liking," the hedonic value of a sensation	The use of two distinct node attributes: affective valence and incentive salience.
Emotion	A feeling with cognitive content.	A feeling node associated with an event node.
Appraisal process	The process that evaluates a situation (event) along particular dimensions thereby determining the activation levels of relevant emotions.	The dynamic association, in LIDA's Workspace, of an event with the emotion(s) activated by the event and the agent's current mental state.
Reinforcement learning	Learning how to map situations to actions so as to maximize a numerical reward signal.	Conscious learning, in several modes, based on not only the conscious contents, but also their overall affective valence, and incentive salience, which modifies both base-level activation and base-level incentive salience.
Model-free learning	A set of reinforcement learning methods using prediction errors in "value" to update, from experience, a stored scalar quantity measuring the value (expected future reward) of pursuing an option.	The update of the base-level incentive salience attribute of event nodes during conscious learning.
Model-based learning	A set of reinforcement learning methods in which an internal model of the environment is learned and used to evaluate available actions or cues on the basis of their potential outcomes.	The internal model is learned via LIDA's conscious learning. Over multiple cognitive cycles, an active option in the Current Situational Model of the Workspace can instantiate relevant parts of the learned model incorporating the various expected results of pursuing the option.

Single-Cycle Motivational Processes in LIDA

The LIDA model implements several motivation mechanisms, which are discussed in this section. Most mechanisms make a trade off between speed and the amount of memory brought to bear on the situation. This makes each best suited for particular situations; however, if implemented together in an integrated architecture they might provide a flexible set of faculties well suited for a range of situations.

To help frame this section, I first review two distinctions previously used in LIDA. The first is a distinction between levels of conscious access. LIDA identifies three levels of conscious access of mental processes or representations: 1) *never conscious*, unable to ever become conscious, 2) *preconscious*, potentially, but not currently, conscious, and 3) *conscious*, part of the content of a conscious broadcast (Dehaene, Changeux, Naccache, Sackur, & Sergent, 2006; Franklin & Baars, 2010). LIDA also distinguishes between four types of action selection: *volitional*, *consciously mediated*, *automatized*, and *alarm*. In *volitional (deliberative) action selection (decision-making)* much of the selection process itself is conscious, that is, a part of successive contents of consciousness (Franklin, 2000). *Consciously mediated action selection* may occur in each of LIDA's cognitive cycles, when a never-conscious selection process consults the current contents of conscious during its choosing of the next action (Franklin et al., 2012; Franklin & Baars, 2010). *Automatized action selection* occurs when one action calls the next via a never-conscious process, and without use of the current contents of consciousness (Franklin et al., 2012; Negatu, 2006). Lastly, action selection during an *alarm* is the topic of the next section.

Alarms. In dangerous situations, agents must react as quickly as possible. The notion of "alarms" was previously articulated, in the context of agent architectures, by Sloman

(1998) and enjoys neuroscientific support (e.g., LeDoux, 2006; Liddell et al., 2005). In terms of the LIDA model, an *alarm situation (alarm)* is one where an agent instantiates a scheme, selects its action, and executes an associated motor plan in rapid succession in response to a dangerous situation prior to, or simultaneous with, consciousness of the situation. This would correspond to an action selection and execution before (or while) the situation is brought to consciousness by attention codelets. For example, an alarm occurs with the unconscious turning of a car's steering wheel in response to suddenly being cut off by another vehicle. During an alarm, some part(s) of the Current Situational Model in the Workspace directly instantiates a scheme from Procedural Memory. The ensuing behavior is sufficiently activated for the immediate selection of its action and, in close succession, the execution of an associated motor plan. All of this occurs before, or simultaneously with, consciousness. Note that alarms require the existence of some learned or built-in scheme. They also require that a scheme be strongly "activated." Here, I suggest the activation *and the incentive salience* both play a role in the rapid recruitment of scheme(s) during alarms. Note that every alarm directly instantiates a scheme before it can come to consciousness. However, later, an agent can become conscious of the situation causing the alarm, and of the action it took in direct response to the alarm. After an action is taken during an alarm situation, the result will likely come to consciousness and become involved in the procedural learning for the scheme selected and executed during the alarm.

Feeling Nodes. The presence of activated feeling nodes, be they drive feelings or interpretive feelings, in a LIDA agent's Workspace has several motivational effects. Feelings and emotions are part of the cue to Transient Episodic and Declarative Memory

(Franklin & Ramamurthy, 2006) as well as to LIDA's nascent spatial memory (Madl et al., 2013). The resultant local associations can contain records of the agent's past feelings and emotions in associated situations. Present and past feelings and emotions influence the competition for consciousness in each cognitive cycle, by affecting the activation of structures in the Current Situational Model, as well as the activation of coalitions containing such structures strengthening a coalition's chances of winning the competition for consciousness. This implies conscious content can include affective portions, and play a role in conscious learning (see *Perceptual Associative Memory Learning* below). If modeling human performance, then affect should modulate learning following an inverted U curve according to the Yerkes-Dodson law (1908): increasing affect causes increased learning up to a point, after which more affect results in less learning.

Incentive Salience of Nodes. I have proposed that event nodes in PAM have a base-level incentive salience while their current instantiations additionally have a current incentive salience. The total incentive salience of events contributes to the salience of events, and as a result, can play several roles within a single cognitive cycle. A coalition involving events with high incentive salience is more likely to win the competition for consciousness, and become the object of conscious learning. If broadcasted, events having a large magnitude of incentive salience better recruit schemes from Procedural Memory, and can instantiate more salient behaviors, increasing their likelihood of selection. This influence of incentive salience on action selection is akin to the influence of "goals" on behaviors in Maes' (1989) behavior network model of action selection.

Action Selection and Consciousness. The LIDA model distinguishes action selection from motor plan selection. The former is the selection of a behavior (an instantiated

scheme) from Procedural Memory, while the latter is the selection of an instantiated motor plan template from Sensory-Motor Memory in the service of action execution. As mentioned in the previous section, Procedural Memory is built using structures originating in Perceptual Associative Memory, i.e., it is a grounded representation. This implies that it may be primed preconsciously, allowing for preconscious priming of actions. Does this suggest that action selection occurs without consciousness? The answer is that there can be several possibilities: The unconscious selection of alarms can occur with a sudden, strong instantiation of a learned scheme that leads to a motor plan execution before consciousness of the event can occur. However, in the vast majority of situations, action selection follows the conscious broadcast (consciously mediated action selection). Consciously mediated action selection can be viewed as a "fast" decision-making using cached base-level activation and base-level (as well as current) incentive salience values to select options. Such a mechanism has also been referred to as model-free control (Daw et al., 2005) as well as "System 1" (Kahneman, 2003, 2011). Finally, the action selection process can itself be conscious, as is the case in volitional decision-making (see "Volitional Decision-making in LIDA" below). Volitional decision-making has also been referred to as model-based control (Daw et al., 2005) and Kahneman's "System 2" (2011).

Conscious Learning in LIDA

LIDA's conscious learning involves several memories. In Perceptual Associative Memory (or recognition memory), new concepts, and associations between them, are learned while existing ones are reinforced. In Spatial Memory, new cognitive maps are stored and existing ones updated. In Transient Episodic Memory broadcasted events

containing what, where, and when are encoded. *Schemes*, procedural memories of what to do when, are learned into Procedural Memory. It is also possible for attention codelets to be learned or reinforced.

Since affective valence plays a role in determining saliency, its presence among the content of a broadcast affects conscious learning, in addition to influencing scheme recruitment and action selection. Following Tindell et al. (2009), affective valence is hypothesized to strongly influence learning based on feelings. I present the details of this below and then discuss how such learning may lead to appetitive or aversive behavior.

Perceptual Associative Memory Learning. Whenever an event is part of a conscious broadcast the nodes and links of the event are learned into PAM. If the node or link is not currently present in PAM, then it is added to PAM, and given an initial base-level activation according to some function of the node's activation. If the node or link is already present in PAM, then its base-level activation is updated according to the same function.

Base-Level Incentive Salience Learning. Previous work on feelings as motivators (Franklin & Ramamurthy, 2006), stipulated that instantiated feeling nodes play an important role in LIDA's conscious learning. They add to their coalition's activation, which, in turn, modulates the learned base-level activation of coalition nodes. Earlier I proposed that feeling nodes should have a valence sign, positive or negative. I next specify how affective valence *additionally* affects perceptual learning. I propose that for feeling events having affective valence, an additional kind of perceptual learning occurs. The base-level incentive salience of broadcasted event nodes is positively updated as a function of the overall affective valence of the event, that is, the sum of the affective

valences of all feeling nodes currently associated with the event. For example, if an event were associated with a positive feeling having a total activation of 0.6 and a negative feeling with total activation 0.5, then the overall affective valence of the event would be +0.1. If the event has an overall positive affective valence, then base-level incentive salience is increased, while if it has an overall negative affective valence its base-level incentive salience is decreased. The magnitude of the overall affective valence determines the strength of this base-level incentive salience learning.

In summary, perceptual learning updates the base-level incentive salience of event nodes. After such learning, events can be directly recognized as "desirable" or not. In the later section "Incentive Salience Link Learning," I discuss a learning sub-type affecting how event nodes obtain *current* incentive salience.

Temporal Difference Learning. Temporal difference (TD) learning (Sutton, 1988) is an approach to learning how to predict a quantity, which depends on future values of a given signal (Barto, 2007). TD is a reinforcement learning technique, which considers not just reward, but expected reward as well. The name TD derives from its use of changes, or differences, in predictions over successive time steps to drive the learning process. In general, the process mandates that the prediction of a quantity (e.g., base-level incentive salience as a prediction of affective valence) at any given time step be updated to bring it closer to the next time step's prediction of the same quantity (See Equation 3). TD algorithms are often used in reinforcement learning to predict a measure of the total amount of reward expected over the future, which, in the context of this chapter, corresponds to base-level incentive salience (Barto, 2007). A number of neuroscientific findings (e.g., Hollerman & Schultz, 1998; Montague, Dayan, & Sejnowski, 1996;

O'Doherty, Dayan, Friston, Critchley, & Dolan, 2003) support the hypothesis that the brain performs TD learning. The mechanisms underlying the changes in valuation appear to occur during learning itself, and "bias decisions without effortful retrieval at the time of decision" (without deliberation in LIDA) (Wimmer & Shohamy, 2012).

Let us denote the t^{th} conscious event (or state) in LIDA by, s_t, and the event's accompanying overall affective valence by r_t. Also, let the *discount rate*, γ, be a real number from the closed interval, [0,1]. This rate defines the degree to which the rewards (overall affective valence) of future events are discounted or de-emphasized, with a value of 1.0 corresponding to no discounting and 0.0 corresponding to complete discounting. Given these definitions, the *complete return*[3], R_t, for conscious event s_t is defined as:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \ldots) \qquad [1]$$

$$= r_t + \gamma R_{t+1}$$

It is then assumed that the long-term value of a state (here, conscious event), $V(s)$, is directly equivalent to the expected return under an unspecified action policy, π:

$$V(s) = E_\pi(R_t \mid s_t = s) \qquad [2]$$

In the context of this work, I denote the base-level incentive salience of the event memory underlying the i^{th} conscious event, s_i at the time t, $V(s_i, t)$. Given a learning rate, α, the temporal difference update for the base-level incentive salience of event s_t at the time $t+1$ is given by:

$$V(s_t, t+1) = V(s_t, t) + \alpha[r_t + \gamma V(s_{t+1}, t) - V(s_t, t)] \qquad [3]$$

[3] Since LIDA considers the overall affective valence (reward) as coming with (not preceding) a given conscious event (state), the complete return is defined starting at t, the time of the event, instead of $t+1$ as is done in the reinforcement learning literature.

This equation says that the update for the base-level incentive salience of event s_t is a function of the difference between the event's previous base-level incentive salience, $V_t(s_t)$, and a new estimate of the complete return of s_t; namely, $r_t + \gamma V(s_{t+1}, t)$. Notice that the r_t term corresponds to the affective valence-based learning described in the previous subsection. Thus, temporal difference learning additionally suggests using the discounted base-level incentive salience of the next encountered event when learning base-level incentive salience of an event.

Event Sequence Learning. Sequences are a large and critical part of Perceptual Associative Memory that associate patterns having temporal regularity. Consider the multitude of melodies, word phrases, and action sequences that one may recognize immediately; these are all event sequences in PAM. As mentioned in the earlier section Scheme Representation in LIDA, event sequences are integral to schemes (Figure 5). Such temporal connections may be initially "hypothesized" by structure building codelets as temporary temporal links in LIDA's preconscious Workspace. If two events and their temporal link win the competition for consciousness then such a link between them would be learned in PAM. The role of event sequences in volitional decision-making will be explored in further detail later on in the section "Multi-Cyclic Motivational Processes in LIDA."

Incentive Salience Link Learning. Drive feeling events and their affective valence play another role in perceptual learning. This learning is critical in determining which event nodes (whether instantiated or not) receive *current* incentive salience from active *drive* feelings. In the proposed model, current incentive salience is transmitted from drive

feelings by *incentive salience links*, that is, PAM links that transmit only incentive salience, not activation.

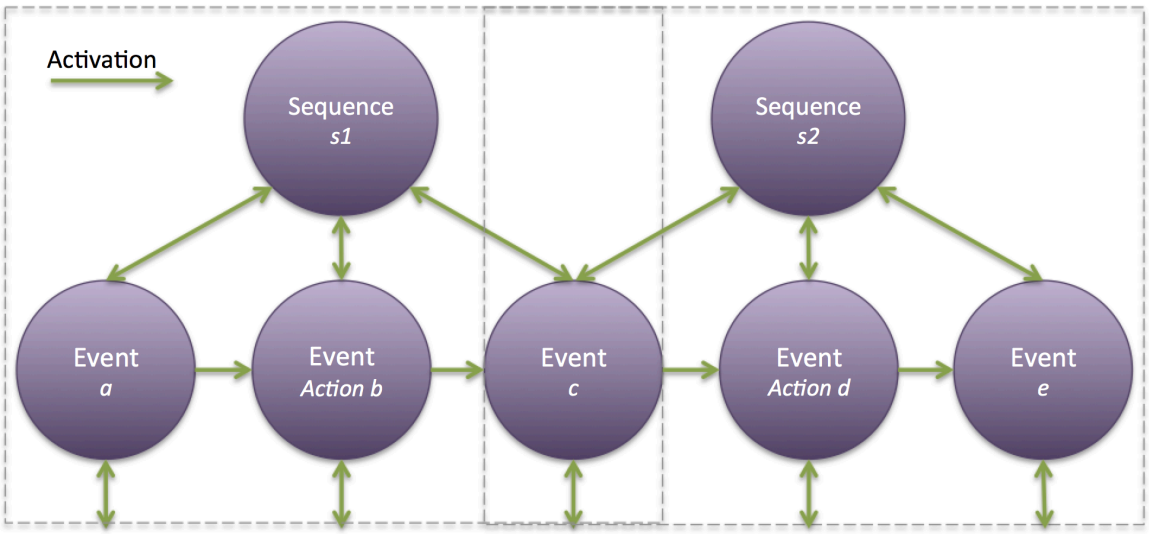


Figure 5. Two event sequences, *s1* and *s2*, underlying two different schemes. The green links transmit activation and bidirectional links suggest that activation may be passed in both directions. The context of one scheme may serve double-duty as the result of another.

Let's first consider an example of learning an incentive salience link. Suppose an agent becomes conscious of an active "thirst" drive feeling node having a negative affective valence. Despite this awareness, the agent takes a long walk, and returns home with the "thirst" feeling node, now strongly activated, carrying a large negative affective valence. Due to a preconscious temporal association by a structure building codelet in the Workspace, the agent becomes conscious of the fact that the walk led to the increase in the thirst node's activation. Consequently, the agent learns an incentive salience link with a *negative* weight from the "thirsty" node to the "walking" event. (Note that this incentive salience link learning would occur in addition to the learning of other links, such as a temporal link running from the walking event node to the strong feeling of thirst

event.) Having learned this incentive salience link, whenever the "thirsty" node becomes activated, "walking" (and any other similarly implicated events) receive a negative amount of current incentive salience. In contrast, suppose that the agent had instead drunk water in response to being conscious of the feeling "thirst," and consequently reduced the drive feeling's activation. If it becomes conscious of this temporal relationship, an incentive salience link with a *positive* weight would be learned, which would promote drinking when thirsty.

How can a LIDA agent learn from experiences involving a change in a drive feeling? In general, these situations involve a temporal link in the Workspace from one event to another event involving *change in activation*, across the two events, of a *drive feeling node*. (Note that such preconscious association could occur between instantiated representations from various places in LIDA's memory.) In such situations, a structure building codelet can introduce an incentive salience link that has the *feeling node as its source* and the event implicated in the feeling node's change as its sink. Supposing that the entire structure, consisting of both events connected by the temporal link, wins the competition for consciousness, then the proposed incentive salience link would be learned into PAM. Links learned in this manner are assigned a weight proportional to the *change* in the drive feeling's activation, a, multiplied by the feeling's valence sign, v_s. Equation 4 expresses how this link weight, w, is proposed to be calculated:

$$w = v_s \cdot f(\Delta a) \qquad [4]$$

After such links are learned they transmit some amount of *current incentive salience* to their sink events proportional to 1) the activation of the source (the drive feeling node) and 2) the link's weight. This, by itself, may lead to the instantiation of the sink.

Intuitively, the idea is to send positive incentive salience to events leading to positive changes to homeostasis, be they a decrease in a negative affective valence or increases in positive affective valence. Similarly, negative incentive salience is sent to events leading to negative changes to homeostasis under this approach. Figure 6 illustrates the example

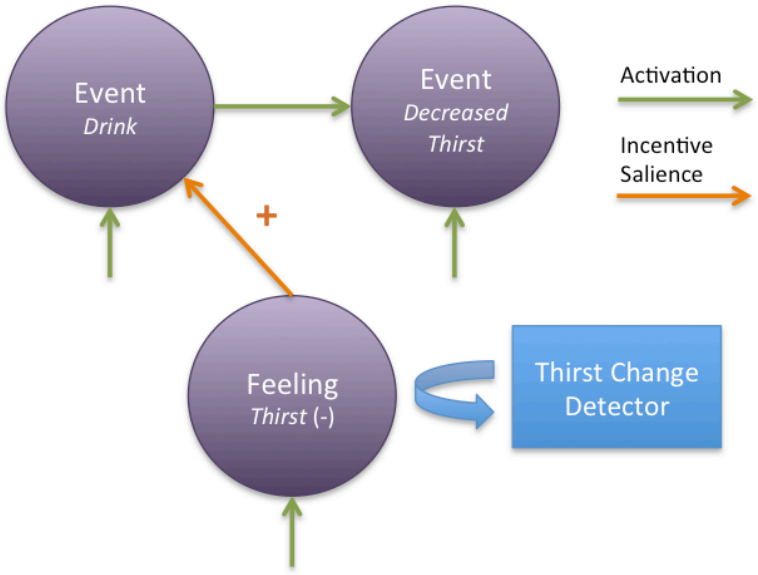


Figure 6. A learned incentive salience link in PAM. The orange-colored link depicts an incentive salience link having a positive weight in (long-term) PAM. The link's source is the *thirst* drive feeling having a negative valence sign. The link's sink is the event implicated in the *decreased thirst* event, here, the *drink* node. The thirst-change detector can recognize decreased thirst by monitoring changes in activation of the *thirst* feeling node. The learning process updates the weight of the incentive salience link according to Equation 4. In this example, a feeling, having a negative valence sign, decreased in activation, necessitating a positive link.

of drinking to reduce thirst. After an incentive salience link with positive weight is learned, the link transmits positive current incentive salience, based on the activation of the "thirst" node, to the "drinking water" event.

Interim Summary

Let's review the motivation extension for LIDA presented thus far. Central to this model are feeling nodes, which serve as a common motivational currency. All feeling nodes

have a *valence sign* attribute, either positive or negative. The valence sign combined with the node's current activation gives its *affective valence*. *Homeostatic sensors* in Sensory Memory detect fluctuations in the state of the agent's body exciting *drive feeling nodes* in Perceptual Associative Memory representing a deviation from homeostasis. Alternatively, *interpretive feelings nodes* represent an evaluation or interpretation of some stimulus, e.g., the sugar in any number of desserts may activate a "sweet" feeling node having positive valence. This interpretive feeling node may be associated, e.g., in the Workspace, with the event "eating ice cream," thereby adding positive affective valence to the event. Such preconscious feeling events in the Workspace have one or more associated feeling node(s) providing the event with valence. The appraisal process in LIDA operates similarly, where events are evaluated along dimensions such as novelty, goal-alignment, agency, coping potential, and availability of a plan, thereby determining the activation level of relevant feeling nodes. The result of the appraisal process is the attachment of an activated feeling node to the cognitive content, typically an event, producing the corresponding emotion. Such dynamic preconscious association of a preconscious event with those feeling node(s) activated by that event also occurs within LIDA's Workspace.

Whenever such a feeling event is part of a coalition that wins the competition for consciousness two main things happen: 1) the broadcast, including the event and feeling node(s), helps recruit schemes for possible action selection; 2) learning based on the broadcast's content occurs. Affective valence plays a special role in this learning. In perceptual learning, broadcast events that have an overall positive affective valence have their *base-level incentive salience* increased, while broadcast events having an overall negative affective valence have their base-level incentive salience decreased. In

accordance with temporal difference learning, the discounted base-level incentive of the next following conscious event also affects base-level incentive salience learning of an event. Broadcasts containing temporal links between two events are also critical for perceptual and procedural learning. If two events and a temporal link connecting them win the competition for consciousness, e.g., the event "push-button" linked to the event "receive candy," then a temporal link is learned. Such links are critical for sequence learning, prediction, and causal learning (not explored in this chapter). Note that temporal links connecting two events may form the context-action part or the action-result part of a scheme in Procedural Memory. For instance, in Figure 5, the link from Event *a* to Event *action b* forms the context-action part, while the link from Event *action b* to Event *c* forms the action-result part.

In the case where a broadcast contains a temporal link whose sink event represents a *change* in activation of a *drive feeling* node, an *incentive salience link* is be learned from the feeling node to the event implicated in the change of the feeling. The weight of the learned incentive salience link is proportional to the product of the drive feeling's affective valence and the change in its activation (see Figure 5).

Thus the motivational aspect of perceptual learning modifies base-level incentive salience and adds or reinforces incentive salience links, which transmit current incentive salience from activated drive feelings. Critically, incentive salience, in addition to activation, affects the saliency of coalitions, and the recruitment of schemes. The influence of the incentive salience on schemes constitutes one way the LIDA model can exhibit appetitive or aversive behavior.

Multi-Cyclic Motivational Processes in LIDA

So far I have discussed the role of motivational processes within a single cognitive cycle. However, many important aspects of cognition are implemented by, and occur over, multiple cognitive cycles. Recall the previous distinction between 1) an option, an instantiated virtual event having nonzero total incentive salience, 2) an immediate goal, an option that is the expected result of a selected behavior, and 3) a pending goal, a goal that is part of a behavior stream, selected by a volitional decision-making process, that cannot be immediately achieved. Now, suppose that an option comes to mind, that is, is a part of the content of a conscious broadcast. As a direct result, several schemes in Procedural Memory will typically be instantiated as behaviors with the option bound to their results. In some situations, an agent may immediately select an action to achieve this option in which case the option becomes the agent's immediate goal. In other situations, the agent enters into a volitional decision-making process. More generally, we can speak of *deliberation* (Sloman, 1999), a kind of cognitive process that builds upon reactive processes.

Volitional Decision-Making in LIDA. What conditions lead to volitional decision-making? I hypothesize that it may be initiated by a behavior with considerable activation, but not enough that it can be currently selected. There are a few main reasons why a behavior would fit this billing: 1) competition with other behaviors, 2) having an unsatisfied context, or 3) a result with low incentive salience. Recall that LIDA's action selection is an extension of Maes' behavior network (1989). In the behavior network, a competition situation arises between two behaviors when one undoes a precondition of the other. Alternatively, competition may arise when there is insufficient difference in

behavior activation to prefer one to the other. For instance, evidence from the basal ganglia suggests that action selection performance improves when competing alternatives inhibit each other requiring an action to have a "statistically significant" greater activation than competitors in order to be selected (Bogacz & Gurney, 2007; Bogacz, 2007). Alternatively, a behavior may have strong activation due to its underlying result and/or action event having positive incentive salience; however, its context might not be satisfied. Similarly, the context may be satisfied, but the result may have a high negative incentive salience. In each of these cases, a deliberative process may be initiated. What initiates such a process? In LIDA, with every instantiated behavior that was produced by an option or a drive a competing behavior with an *internal* action enters into Action Selection. If this special behavior is selected, its internal action starts an ideomotor theory volitional process (see next section).

Ideomotor Theory in LIDA. William James first introduced the ideomotor theory of volition (James, 1890; Shin, Proctor, & Capaldi, 2010). James postulated proposers, objectors, and supporters as actors in the drama of acting volitionally. He might have suggested the following scenario in the context of dealing with a feeling of thirst: The idea of drinking orange juice "pops into mind," that is, propelled to consciousness by a proposer motivated by the drive feeling, "thirst," and a "liking" for the sweetness of orange juice, it becomes the contents of consciousness. "No, it's too sweet," asserts an objector. "How about a beer?" says a different proposer. "Too early in the day," says another objector. "Orange juice is more nutritious," says a supporter. With no further objections, drinking orange juice is volitionally selected.

Baars incorporated ideomotor theory directly into his Global Workspace Theory (1988, Chapter 7). The LIDA model fleshes out volitional decision-making via ideomotor theory within Global Workspace Theory (Franklin, 2000) as follows. An idea "popping into mind" in the LIDA model is accomplished by the idea being part of the conscious broadcast of a cognitive cycle. Thus, the conscious broadcast implements the characters in James' scenario, with some broadcasts acting as proposers, others as objectors, and others as supporters, the content of each "popping into mind" if it wins the competition, and is broadcast.

But, how does the thought "Let's drink orange juice" come to consciousness? It must arise from a virtual "drink-orange-juice" event node with positive total incentive salience in the Workspace. According to the motivational extension to LIDA, an option may arise from an event having nonzero base-level incentive salience, e.g., seeing a glass of one's favorite orange juice. Alternatively, or additionally, an activate drive feeling, e.g., thirst, may provide sufficient current incentive to the drink-orange-juice event node in PAM to instantiate it in the Workspace as an option. Like every higher-order cognitive process in the LIDA model, volitional decision-making occurs over multiple cycles. In this example, "drinking orange juice" may come to consciousness for the reasons just described. If this proposal fails to immediately lead to an external action selection, another proposal, or an objection, could come to consciousness.

Specific implementations of internal actions may use a "timekeeper" (Franklin, 2000) or an "impatience" feeling node, which increases the likelihood of an action selection the longer the volitional process takes. Volitional decision-making can end when time runs out on some proposal (option), and an action is thus selected.

The LIDA model hypothesizes that neural correlates of volitional decision-making include the ventromedial prefrontal cortex, the striatum, the basal ganglia and the ventral anterior cingulate cortex (MacDonald, 2008; Pasquereau et al., 2007; Rowe, Hughes, Eckstein, & Owen, 2008) .

Proposers and Objectors from Mental Simulations. How might LIDA produce James' proposers and objectors? During volitional decision-making, instantiated events in the Workspace, including options and drives, repeatedly cue LIDA's Perceptual and Episodic Memories. This cueing retrieves relevant local associations, and, drawing upon knowledge stored by the temporal links of event sequences, includes potential future events. Such cueing and recall can be thought of as "internal" or virtual actions (Nakayama, Yamagata, Tanji, & Hoshi, 2008). Potential future events are grounded virtual representations of potential consequences of pursuing the options. Over multiple cognitive cycles, such virtual actions can produce an instantiated, "chain" of events. Such chains can become coalitions that may compete for consciousness. Critically, each event in the coalition contributes its total activation and total incentive salience. Recall that total incentive salience includes a base level component (updated by TD learning) and a current component (modulated by current drive feelings). Thus, the incentive salience of future events, proposers and objectors in ideomotor theory, would contribute to a coalition's overall strength or salience. The net effect is to bias those behaviors or behavior streams believed to lead to a "desirable" future set of events. For instance, considering going outside in the rain unprepared may lead you to imagine getting wet, which leads you to grab an umbrella, or imagining being robbed on a walk in the city may compel you to take pepper spray along. Thus, appetitive or aversive behavior may

also result from an anticipation, imagination, or recollection of an event with incentive salience.

Deliberation and Model-Based Control. I take the stance that deliberation in general should involve the kind of mental simulations described in the previous section. For each alternative, a mental simulation must occur, which accesses the total incentive salience of expected events. Huys et al. (2012) found that humans adopt a Pavlovian strategy for pruning decision trees. During mental evaluation of a sequence of choices, they curtailed any further evaluation of a sequence as soon as they encountered a large loss. In terms of the LIDA model high negative incentive salience may cut off one "branch" in a tree of possibilities. For example while planning a route to the airport you realize the plan will involve taking the interstate, which you view as dangerous. This leads you to abandon that specific plan, and come up with another route taking safer roads. Conversely, sufficient positive incentive salience may prompt the selection of a behavior stream, leading to the end of the deliberation altogether.

Recall that model-based learning refers to a set of reinforcement learning methods in which an internal model of the environment is learned and used to evaluate available actions or cues on the basis of their potential outcomes. I propose that volitional decision-making, modulated by the total incentive salience of anticipated or potential events in the Current Situational Model of the Workspace, underlies LIDA's implementation of model-based control. Such processes in LIDA typically require multiple cognitive cycles, in accordance with the slower, model-based control or "System 2" (Kahneman, 2011).

Experiment Replication

To validate the motivation extension to LIDA, I replicated a psychological experiment of

the reinforcer devaluation paradigm. In particular, I created a computer simulation of an experiment testing the effects of orbitofrontal lesions on the representation of incentive value in associative learning (Gallagher, McMahan, & Schoenbaum, 1999). The orbitofrontal cortex (OFC) is thought to contain representations of the motivational significance of cues (conditioned stimuli) and the incentive value of expected outcomes. The significance of the reinforcer devaluation task is that normal performance depends on the ability of a conditioned stimulus (CS) to gain access to the motivational properties of an upcoming unconditioned stimulus (US). Moreover, it illustrates the difference between the fast acting, slow-adapting model-free control (consciously mediated action selection in LIDA) and the slow-acting, fast-adapting model-based control (volitional decision-making in LIDA).

In the original study, the experimenters first divided rats into two groups: those in the first had their orbitofrontal cortex lesioned, while those in the second maintained an intact OFC. Rats were then trained in a *conditioning phase* in accordance with standard Pavlovian conditioning: Over a series of 40 trials, rats were presented with a 10-second light CS, which was paired with (immediately followed by) a food delivery, itself followed by a ten-minute period in which the rat was allowed to eat freely. After a series of conditioning trials, a conditioned response (food cup behavior) to the CS was established. The measure during these trials was the rat's appetitive behavior towards the food cup during the last 5 seconds of the 10-second cue.

After the conditioning phase, each rat underwent three trials in a different *US devaluation phase*. In each trial of this phase, there was no light cue, instead food was delivered first and the rat was given 10 minutes to eat. After this period, the rat

experienced an aversive event; namely, the injection of LiCl, producing temporary sickness. The US devaluation phase introduces two more experimental conditions: In the *paired injection condition* the experimenters injected the rats immediately after the eating period. In the *unpaired injection condition* the rats were injected six hours after eating. Combining these conditions with the earlier OFC lesion manipulation, there were a total of four experimental groups: lesioned-paired, lesioned-unpaired, intact-paired, and intact-unpaired. The measure during the US devaluation phase was the amount of food consumed during the eating period of each trial.

After the US devaluation phase, the experimenters performed a *devaluation test phase* that revisited the rats' conditioned responses (CRs) to the light CS. In this phase, each rat was presented with the light CS only, i.e., without any further experimental manipulations. As in the first phase, the measure for these trials was the rat's appetitive behavior towards the food cup during the last 5 seconds of the 10-second cue.

Although the light CS was absent during the devaluation phase, its previous association with the food US provides a basis for anticipating the US. The experimenters found that lesions of the OFC did not affect either 1) the initial acquisition of a conditioned response to the light CS in the initial conditioning phase, or 2) the learning of food aversion in the US devaluation phase. However, in the devaluation test phase, OFC lesioned rats exhibited no change in their conditioned responding to the light CS, i.e., they continued to exhibit appetitive food cup behavior. This outcome contrasts with the behavior of control rats: after the devaluation of the US, a significant decrease in the food cup approaches occurred in the devaluation test phase. The experimenters hypothesized

that, after OFC damage, the cue was unable to access the representational information about the incentive value of the associated US (Gallagher et al., 1999).

LIDA Account of Experimental Behavior. Recall that, in the first phase of the experiment, a light cue is paired with food delivery, that is, food is delivered immediately after the light signal terminates. The agent's appetitive behavior towards the food cup is recorded during the last 5 seconds of the 10-second cue. The measure of learning in Phase 1 was food cup behavior recorded as a percentage of total behavior recorded during the last 5-second observation interval of the 10-second CS presentation (the light). This was achieved by recording a single behavior for each 1.25-second interval, and then computing the percentage of behavior that was food cup behavior, i.e., the frequency of food cup behavior in an observation interval was divided by the total number of observations made in that interval. The results of Phase 1 of the original experiment are shown in the first graph in Figure 7, which shows no significant difference between the experimental groups in the acquisition of a conditioned response. Next, I describe how a LIDA agent models the events of this phase, and how it learns a conditioned response.

Let us first assume that a LIDA agent replicating this experiment would have a memory for the light cue in the form of a "light-cue" node in its Perceptual Associative Memory (PAM). Then, during this phase, the light-cue node would be instantiated while the light is on, and this node would come to consciousness. Later on, a "food" node is similarly recognized with, due to the short time passage, the light node still being active in the Conscious Contents Queue of the Workspace. A temporal structure building codelet then builds a new structure in the Current Situational Model of the Workspace, based on the food node and its temporal predecessor, the light node. If this structure of

light, link, and food is formed into a coalition by an attention codelet, and wins the competition for consciousness, then a new temporal link from light-cue to food would be learned into PAM.

The conscious broadcasting of this structure would also serve to recruit resources to deal with the situation. In this case, it is assumed that the broadcast instantiates one or more previously learned schemes from Procedural Memory having a context of "food," and an appetitive food cup action, "approach food." If such a scheme is instantiated and its resulting behavior is selected for execution, then we can expect the agent to additionally perform the "eat food" action. Note that when the "eat food" action event occurs, it will be is recognized and it will be likely accompanied by a built-in "food pleasure" feeling node having positive affective valence. If the "eat food" event comes to consciousness, two kinds of learning are performed: 1) a temporal link from "approach food" to "eat food," representing an affordance, is positively reinforced in PAM, and 2) due to the positive affective valence from the "food pleasure" feeling node, a positive update is made to the "eat food" node's base-level incentive salience.

Several repetitions of this first phase would lead to repeated conscious broadcasts of the "light cue, food delivery, approach food, eat food" event sequence. Temporal difference (TD) learning would occur each time a structure with a temporal link is present in the broadcast, and would update the base-level incentive salience of the link's source event. Working backwards in order of occurrence, TD learning would first update the base-level incentive salience of the "approach food" event based on the difference between the approach event's current base-level incentive salience and that of the following event, "eat food." Later, the two antecedent events of food delivery and light

cue would, incrementally over multiple cognitive cycles, gain base-level incentive salience. At first, this would only affect the approach node (immediate predecessor to eat food), but later the other nodes would receive some "credit" in predicting the food "reward." The upshot of this learning is that the node for the light cue gains a high base-level incentive salience, which later helps it to strongly activate the "food" node via learned temporal links. Once "food" is strongly activated, the selection of an appetitive food cup behavior is likely to occur, even in the absence of actual food.

For the second phase of the experiment, there were two conditions: 1) In the paired injection condition, rats were given food immediately followed by (paired with) an illness inducing LiCl injection, and 2) the unpaired injection condition also first provided food, but its LiCl injection occurred six hours later. The results from the original experiment are shown in the second graph in Figure 7. The experimental groups that received food paired with the injection are shown in white. These groups learned to greatly reduce their food consumption. The unpaired groups are shown in black. These groups attenuated their food consumption by significantly less. There was no significant decrease in food consumption across sessions for these unpaired groups. One explanation for the *apparent* decrease for the rats in the unpaired groups is that they might have performed some deliberative temporal association, followed by conscious learning, to actually form a memory of an association between the food and the injection.

For the paired experimental group, the mental events occurring in a LIDA-based agent are similar to those of the first experimental phase. The agent would, via conscious learning, add a temporal link from the food node to the injection event. Additionally, after the injection, the agent would recognize a "sickness" event and, via conscious.

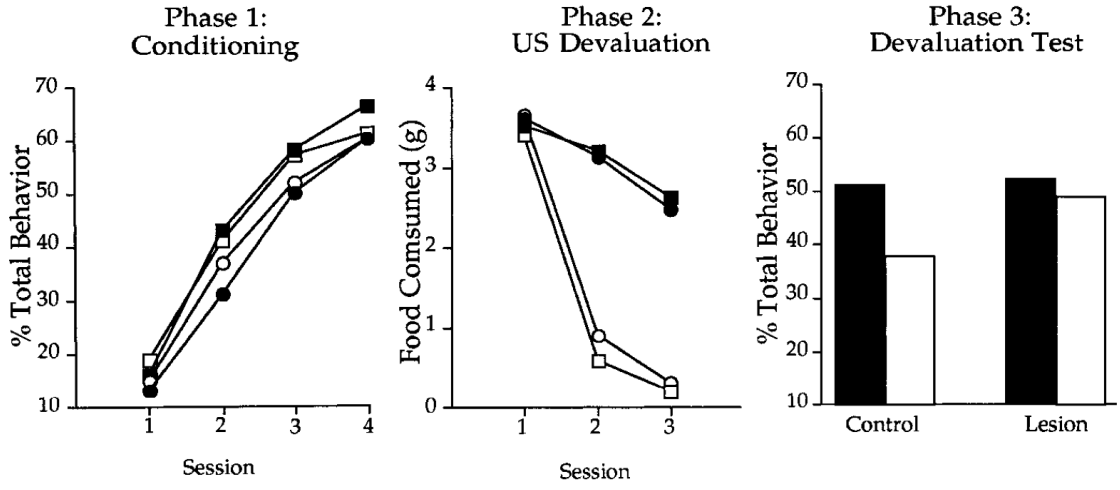


Figure 7. The results of the original reinforcer devaluation experiment. The Phase 1 graph shows that all groups acquired conditioned responses to the light cue, as evidenced by their increased food cup behavior as a percentage of total behavior during the latter half of the light cue presentation. The Phase 2 graph shows that the rats receiving paired LiCl injections significantly reduced their food consumption as compared to those rats receiving unpaired injections. There was neither a significant difference in food consumption (due to lesion) among the paired groups nor among the unpaired groups. Lastly, the Phase 3 graph shows that only intact rats receiving paired injections significantly reduced their food cup behavior during the devaluation test.

learning, add another temporal link from the injection event to the sickness event. The sickness event would come with an accompanying "sickness feeling" feeling node with negative affective valence. Conscious learning would then lead to the assignment of a low (possibly negative) base-level incentive salience to the sickness event because of this negative affective valence. Repeated conscious exposures of this "food delivery, approach food, eat food, injection, sickness" sequence would then, via temporal difference learning, "devalue" or decrease the base-level incentive salience, first of the injection event, then of the earlier events as well.

For an agent in the unpaired group, the processing and learning would be the same as for the paired group, except that, since the injection occurs six hours after the

food presentation, the "food" node would have long since decayed away from the Workspace during the aversive events. Thus, for such a simple agent, no temporal links are ever learned between the eating event and the injection or sickness. (In this second unpaired group, the injection and sickness events would still be learned with a temporal link between them, and both would be given low base-level incentive salience.) For the unpaired groups, the apparent decrease in food consumption across sessions may have been a result of deliberative association between the injection event and an earlier event (e.g., food consumption) recalled from Episodic Memory.

In the final phase of the experiment the agent receives several presentations of the light cue that are not followed by any additional experimental manipulations. As in Phase 1, the agent's appetitive food cup behavior is recorded during the last 5 seconds of the 10-second cue. The experimental findings for Phase 3 of the experiment are shown in the final graph in Figure 7. The white bars represent groups receiving paired injections, and the black bars represent unpaired groups. Only the paired control (having intact OFC) group (left white bar) significantly decreased its rate of conditioned responses from the other groups, whose CR rate were statistically equivalent.

To set up a LIDA-based explanation of Phase 3, I first note the LIDA agent can form new long-term memories based on temporal links. Such links afford the agent the ability to instantiate expected future event(s) into the Current Situational Model of the Workspace based on currently active nodes in the CSM. This process may repeat with subsequent future events each further ahead in time than the last. It is also hypothesized that expected events, their temporal links, and the original event can all form into a single coalition. This coalition competes based on the total activation and total incentive

salience of each of these events including the expected one(s). Keeping this in mind, if a LIDA agent replicating the results of the experiment is to be constructed, I must hypothesize a functional role for the OFC, and relate this role to a capacity of a typical LIDA agent that must be removed to simulate an OFC lesion. The OFC has been suggested as critical for "associative learning," and the representation of "associative information, particularly information about the value of expected outcomes" (Schoenbaum, Takahashi, Liu, & McDannald, 2011). The neural activity in the OFC "increases to cues and after responses that predict rewards." Finally, the authors suggest viewing OFC function as "constructing or implementing a model-based representation" (Schoenbaum et al., 2011).

Based on these ideas, I define a *lesioned OFC LIDA agent* as one that cannot use the total incentive salience of *expected* event(s) in determining the total incentive salience of a coalition involving those event(s). An *intact OFC LIDA agent* is one that can access the total incentive salience of expected event(s) and use it in computing the total incentive salience of any coalition containing the event(s). Note that for both agent types, the base-level incentive salience of event nodes is assumed to be intact. Likewise, the lesion does not affect existing memory in PAM or Procedural Memory. Lastly, both agent types, lesioned or intact, can perform temporal difference learning.

The results for the unpaired injection groups in Phase 3 can be explained simply: Since the injections were unpaired, the aversive injection event would not be active in LIDA's Workspace contemporaneously with the food event, and thus it cannot be associated by a structure building codelet in the Workspace. (This fact is independent of whether the agent is lesioned or not.) As a result, a potential temporal link never comes to

consciousness and no TD learning can occur that might devalue the base-level incentive salience of the temporally earlier events — the food delivery and the light cue. As a result, the light-cue event retains its high base-level incentive salience, originally learned from Phase 1, motivating the agent to approach the food cup when the light cue is later shown in Phase 3.

Now, let's consider the two paired injection groups. For the paired-lesioned OFC group, the lesioned OFC LIDA agent is only able to evaluate a stimulus' (light cue) value based on its base-level incentive salience. This would prevent the agent from integrating any expectation of future aversive events into a coalition with an instantiated light cue event node. Since the light cue occurred in the initial conditioning phase of the experiment, its base-level incentive salience was positively updated by TD learning. However, since the light cue did not occur in Phase 2, it could not have been altered by TD learning. Consequently, a coalition involving the light cue has an overall positive incentive salience, and, via a conscious broadcast, would prompt appetitive food cup behavior. The expected result is that an OFC-lesioned, paired-injection LIDA agent would exhibit a similar percentage of food cup behavior as both unpaired groups.

Why might an intact, paired-injection agent reduce its food cup behavior? This type of agent, given the instantiation of the light cue node, is able to instantiate the subsequent events it has learned to expect. It does this by repeatedly cueing its PAM based on instantiated events in the CSM of the Workspace. The initial light-cue event cues PAM instantiating the food-delivery event into the Workspace. Next, the eating event is instantiated, then injection, etc. An attention codelet then forms a coalition from this integrated sequence of events, and bring it to the Global Workspace. While the

earlier events in this sequence may have a fairly high base-level incentive salience, the later ones would surely have a low base-level incentive salience due to the devaluation trials. Such a coalition would then have less overall incentive salience and, consequently, less of a chance to win the competition for consciousness. Even if it does, it would have less of a chance to induce an appetitive action selection.

Agent Implementation. I reproduced the reinforcer devaluation experiment computationally to provide an environment for the motivationally extended LIDA agent. Using the LIDA framework, the simulated environment was implemented as a separate framework module from the agent. Only the sensing and acting processes coupled the two. A process of the environment module performed the manipulations of the experiment. In a sense, the experimenter was built into the simulated environment.

The experiment was simulated using a series of *stages*, with each stage representing a main event in the original experiment. The possible stages are *light-cue*, *food-delivery*, *LiCl-shot*, *nothing*. Only one stage may occur at a time, and each stage lasts for a fixed amount of simulated time, which was measured in *ticks*. For this work, one tick represents one millisecond of real time. The minimum length of a stage was 1,000 ticks (simulating 1 second) with longer stages lasting for some integer multiple of 1,000 ticks. I define a *stage sequence* as a specific sequence of stages. Each of the three main phases of the experiment was implemented as a stage sequence. Phase 1 was implemented with the stage sequence *light-cue*, *food-delivery*, *nothing*, *nothing* repeated 4 times. Phase 2 was implemented with the stage sequence *food-delivery*, *LiCl-shot*, *nothing*, *nothing* repeated 2 times. Phase 3 was implemented with the stage sequence *light-cue*, *nothing*, *nothing* repeated 2 times.

The replicated experimental environment maintains several variables representing the state of the experimental objects, the agent's internal state, and several other parameters. All quantities mentioned below (see Table 2) are scalars with possible values in the closed interval [0, 1]. The *food-amount* variable represents the current amount of food available to the agent in the food cup. The *food-delivery amount* parameter determines the amount of food delivered to the food cup. The *consumption-amount* parameter governs how much food is consumed when the agent eats. The *eating-action* variable implements the proprioceptive stimulus of performing the eating action, while the *eating-sensation amount* parameter specifies the value that *eating-action* takes immediately after the agent performs the eat action. The *eating-pleasure* variable implements the proprioceptive stimulus of "enjoying" the consumption of food, and the *eating-pleasure amount* parameter specifies the value *eating-pleasure* takes immediately after the agent eats. The *sickness level* variable represents the proprioceptive stimulus of sickness, and the *shot sickness amount* parameter specifies the value *sickness-level* is set to when the agent receives the LiCl injection. Each of these proprioceptive variables has an associated decay parameter (Table 2). The food-delivery amount and consumption amount values were chosen such that the agent could eat twice for each food delivery. The eating sensation parameter values were chosen such that it would take around 500 ticks for the eating node to decay way. If the eating sensation decays too quickly, it is not possible to temporally associate eating with future events. The eating pleasure parameter values were chosen such that eating "pleasure" would occur long enough to be a part of a conscious broadcast. Likewise, the shot sickness parameters were chosen so that the

sickness feeling decays away in about 200 ticks — long enough to be consciously recognized.

Table 2. The parameters of the simulated experimental environment and their values.

Parameter name	**Value**
Food-delivery amount	0.5
Consumption amount	0.25
Eating-sensation amount	0.5
Eating-sensation decay	0.001
Eating-pleasure amount	0.3
Eating-pleasure decay	0.005
Shot-sickness amount	1.0
Sickness-level decay	0.005

The agent is able to sense each of the proprioceptive stimuli, and can also sense two "visual" stimuli: one representing the current stage (e.g., light-cue stage), and another signifying whether food was present in the food cup. At any one instant (tick), it is possible for the agent to sense any or all of these stimuli.

The LIDA agent has a single motor plan, *eat-food*, which it can execute to affect the environment. When the agent executes the eat-food motor plan, several things occur in the simulated experimental environment: 1) The *eating-action* variable is set to *eating-sensation amount*, 2) The *food-amount* variable is decreased by *consumption-amount* or is set to 0.0, 3) The *eating-pleasure* variable has its value increased in proportion to the decrease in *food-amount*, and 4) If the current stage is the *light-cue* stage, a count of appetitive food cup behaviors is incremented. Independent of the agent's actions, the *food-amount* variable is set to *food-delivery amount* by the simulation whenever the food-delivery stage occurs. Similarly, the sickness variable is set to *shot-sickness amount* whenever the LiCl-shot stage occurs.

The LIDA agent's Sensory Memory acts as a simple buffer of sensor activity, which was refreshed every tick. The implementation of its Perceptual Associative Memory (PAM) is more complicated. The PAM has five built-in nodes of regular type: *light-cue*, *food-delivery*, *eating-action*, *injection*, and *sickness,* and two interpretive feeling nodes: *eating-pleasure*, and *sickness-feeling*. There are feature detectors processes for each node, which check for the presence of sensory activity corresponding to a PAM node. All PAM nodes begin each simulation with zero base-level activation.

New to this implementation of PAM is the addition of temporal difference learning, and base-level incentive salience learning to the existing algorithm for conscious learning. Two parameters modulate this learning: *learning rate*, α, and *discount rate*, γ. The former is a multiplicative factor modulating the magnitude of all updates to base-level activation and base-level incentive salience in PAM. The latter modulates the degree to which the incentive salience of a predecessor event is "credited" to its immediate successor in temporal difference learning. (For example, see Equation 3 in Temporal Difference Learning). In addition to propagating activation, PAM also gains the ability to propagate current incentive salience along incentive salience links. Activation passing in PAM is multiplicatively modulated by a third parameter, *upscale*. The effects of different values of these three parameters will be described later in the testing section.

A general addition to the LIDA framework with the motivational extension is the addition of a current incentive salience attribute to all nodes and, additionally, a base-level incentive salience attribute for long-term PAM nodes. Base-level, current, and total incentive salience may take on values in the closed interval [–1.0, 1.0] where negative

values represent negative incentive, and positive values represent positive incentive. All of these changes to the implementation are designed for inclusion in a future version of the LIDA software framework (see Chapter 6).

The agent's Workspace contains both the Current Situational Model (CSM) submodule and the Conscious Contents Queue (CCQ) submodule. The Workspace has a *temporal structure building codelet*; one that tries to builds a structure in the CSM consisting of the previous conscious event from the CCQ and an event that is a part of the incoming percept to the CSM. In particular, the codelet proposes (adds) a new temporal link from the previous CCQ event to the current incoming percept event. This codelet was not implemented as a periodically running task; rather, the algorithm ran each time a new node was added to the CSM from PAM. This is a more parsimonious approach as a link need only be built when a new event arrives from the percept. In practice, this approach also avoided the creation of cycles in the CSM graph structure. The pseudocode details of this codelet's algorithm can be found in Appendix B. Secondly, the Workspace also has a *feeling structure building codelet*, running as a separate task, which proposes links associating instantiated feeling nodes with the currently most active event in the CSM.

There is a single *default attention codelet*, which periodically forms coalitions from the most active node in the CSM and those active node(s) and link(s) connected with the most active node. A conscious broadcast is triggered every 100 ticks for any currently active coalition with nonzero activation.

The agent's Procedural Memory has one built-in scheme with the *food* node as the context, the *eat* node as the action, and an unspecified result. The Action Selection

module checks if an action should be selected every 75 ticks with a 200 tick refractory period after each action selection. Sensory-Motor Memory had a single motor plan template, which associated the selected action *eat* with a motor plan. Selected motor plans are run in a separate task after *process action task* ticks have passed to simulate the time to execute the action. The *decay rate* parameter involves multiple modules as it affects the decay of nodes, coalitions, schemes, behaviors, and incentive salience throughout several LIDA modules. Finally, in calculating the total activation of nodes, a weight on the base-level activation's contribution was used. Table 3 summarizes these main parameters for this motivational agent implementation.

Table 3. Main parameters of the replication agent, their associated modules, and their default parameter values.

Parameter Name	**Module**	**Default Value**
Sensing frequency	Sensory Memory	1 tick
Feature detector frequency	PAM	5 ticks per run
Learning rate	PAM	0.1
Discount rate	PAM	0.5
Upscale factor	PAM	0.1
Feeling structure building codelet frequency	Workspace	6 ticks per run
Attention codelet frequency	Attention Codelet Module	5 ticks per run
No broadcast occurring period for broadcast trigger	Global Workspace	100 ticks
Individual coalition threshold for broadcast trigger	Global Workspace	0.0 activation
Action selection frequency	Action Selection	75 ticks
Action selection refractory period	Action Selection	200 ticks
Process action task	Sensory-Motor Memory	150 ticks
Decay rate	Various	0.005
Base-level activation weight	Various	0.2

The OFC lesion manipulation was implemented in the Procedural Memory module. In the intact (non-lesioned) case, event nodes (options) from the current conscious broadcast are evaluated based on the average of their total incentive salience and the total incentive salience of all associated expected events. For the lesioned case, broadcasted event nodes (options) are evaluated only on their total incentive salience. In either case, if the average is positive, then the node is permitted to recruit and instantiate schemes, otherwise it is not.

Simulation Results

Fixed Parameters. I first evaluated the motivational agent by having it participate in multiple trials of the simulated reinforcer devaluation experiment. First, using a fixed set of parameters summarized in Table 3, I performed 30 lesioned trials of the experiment and also 30 non-lesioned trials. In each trial a LIDA agent was generated with only the aforementioned PAM nodes built in. The agent was then exposed to the 3 stages sequences corresponding to the 3 phases of the experiment in which it had to learn from experience. The measurement in the original experiment for Phases 1 and 3 was the ratio of the number of appetitive food cup behaviors to total behaviors recorded during the latter half of the light cue presentation. For the simulation I simply recorded such behaviors during the entire light cue presentation. Figure 8 summarizes the synthetic results for the lesioned and non-lesioned cases in the first and third experimental phase. The results are consistent with the original experiment. There was no significant difference between conditions in terms of food cup behavior in Phase 1, but there was a significant difference between these conditions in Phase 3. In Phase 3 the non-lesioned

condition almost always avoided approaching the food cup, while the lesioned condition failed to halt its food cup behavior.

In the following sections, I study the effects of changing three parameter values that were fixed in the original test. By exploring a range of parameters, I can give a better sense of the stability of the original fixed parameter set.

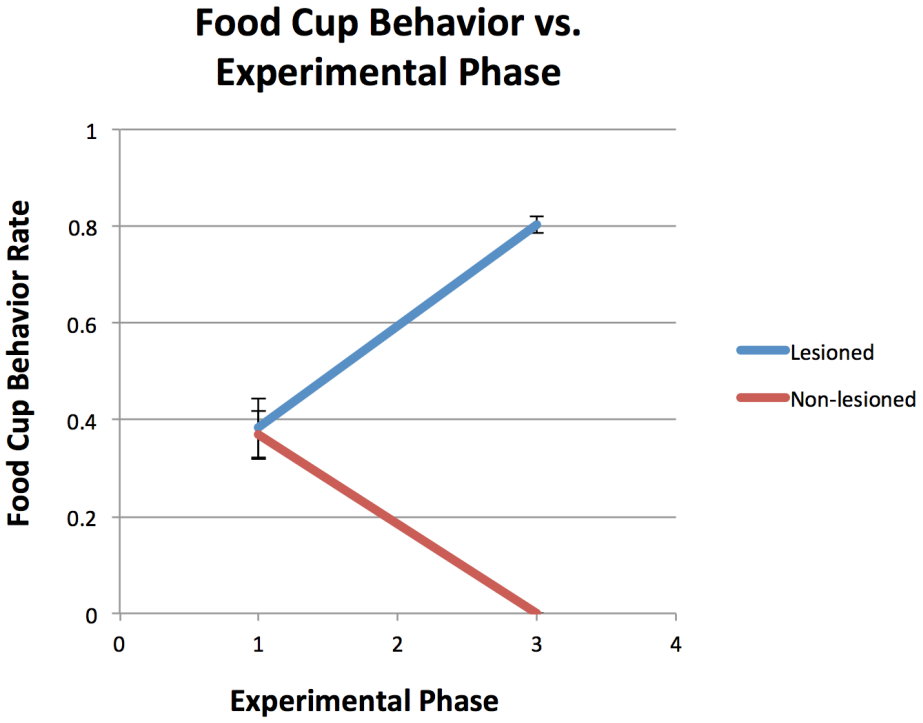


Figure 8. The results of the reinforcer devaluation experiment replication for a fixed parameter set. The error bars represent the standard deviation.

Varying Learning Rate Parameter. The first parameter for which I tested a range of values was the *learning rate*, a multiplicative factor modulating the magnitude of all updates to base-level activation and base-level incentive salience in PAM. In the fixed parameter test the learning rate was 0.1. Here I explored values of the learning rate from 0.0 to 0.5 for both lesion conditions. The results for Phase 1 of the experiment are given in Figure 9 and show a general increase in appetitive behavior as learning rate increases

until stability at and above 0.3. There was no significant difference between lesion conditions.

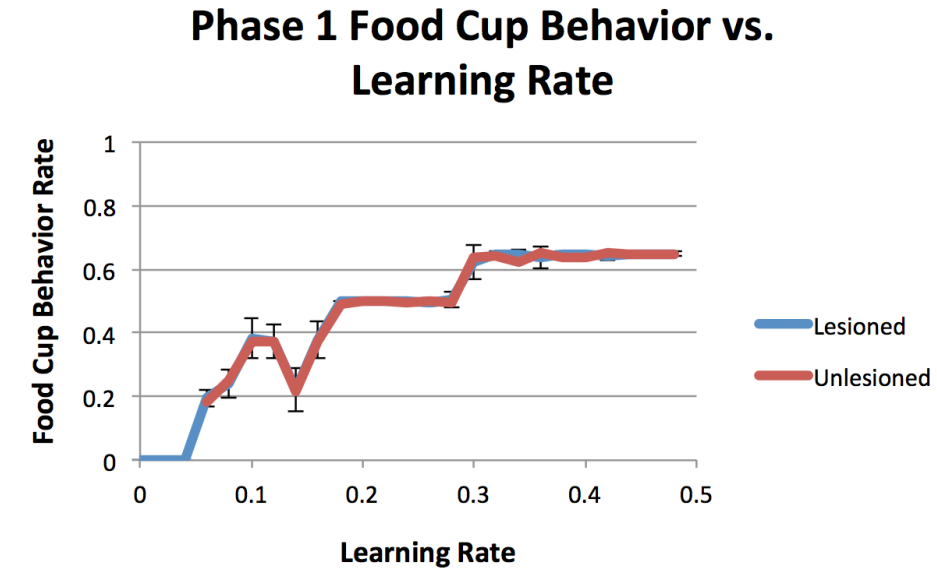


Figure 9. Phase 1 food cup behavior versus learning rate.

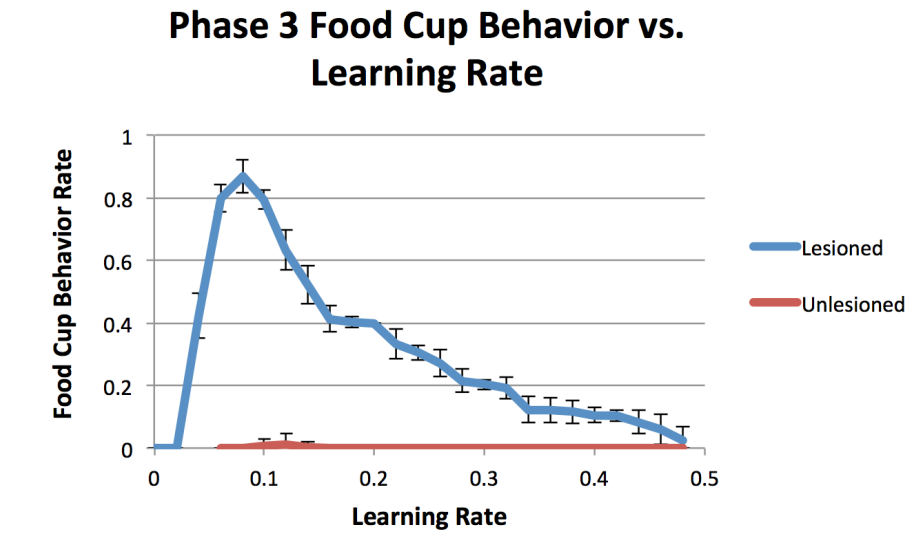


Figure 10. Phase 3 food cup behavior versus learning rate

For Phase 3, we should expect *lesioned* agents to maintain substantial food cup behavior and *non-lesioned* agents to avoid food cup behavior. Across learning rates there

was a significant difference between the lesioned and non-lesioned cases with non-lesioned agents largely abstaining from food cup behavior (Figure 10). Lesioned agents' behavior was more complex for various learning rates. Around from 0.0 to 0.1 increases in learning rate serves to bolster food cup behavior; however, beyond 0.1, increases in learning rate decrease this behavior, likely due to the increase in the rate of temporal difference learning. That is to say a high learning rate allows the lesioned agent to overcome its mental deficit via more rapid TD learning, which allows it to more quickly revise the base-level incentive salience of food.

Varying Discount Rate Parameter. The second parameter for which I explored a range of values is the discount rate, which modulates the degree to which the incentive salience of an event is "credited" to its immediate temporal predecessor in temporal difference learning. In the fixed parameter test the discount rate was 0.5. In Phase 1, there was no significant difference in food cup behavior across values of the discount rate from 0 to 1. Moreover, there was no significant difference due to the lesioning manipulation either (Figure 11).

In Phase 3, the discount rate's value had a significant effect on food cup behavior for the lesioned condition. In particular, starting at around 0.5, higher values of the parameter served to attenuate, but not completely extinguish, food cup behavior (Figure 12). Again this suggests that improving Temporal Difference learning helps the agent overcome its deficit to a degree for this particular task.

Varying Upscale Parameter. The final parameter for which I tested a range of values was the *upscale*, which multiplicatively modulates activation passing in PAM. In the

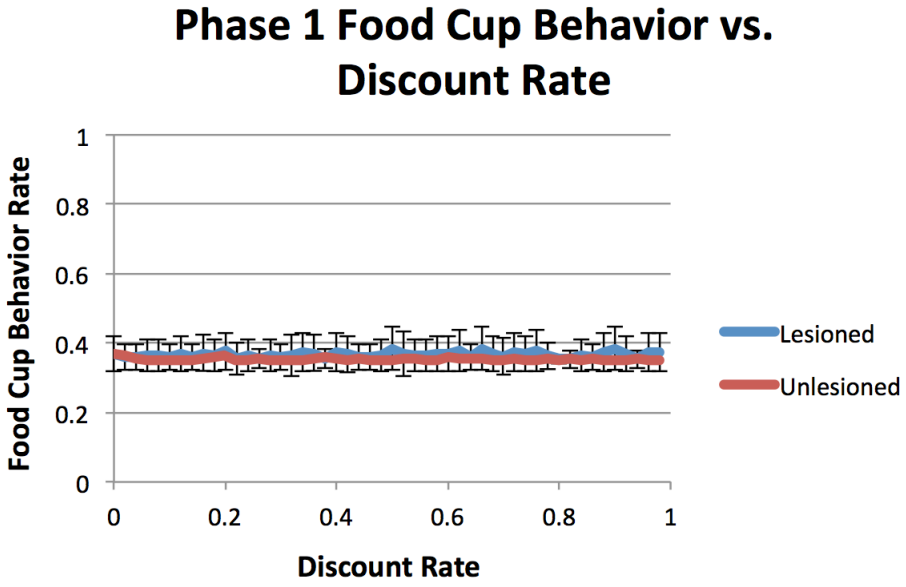


Figure 11. Phase 1 food cup behavior versus discount rate.

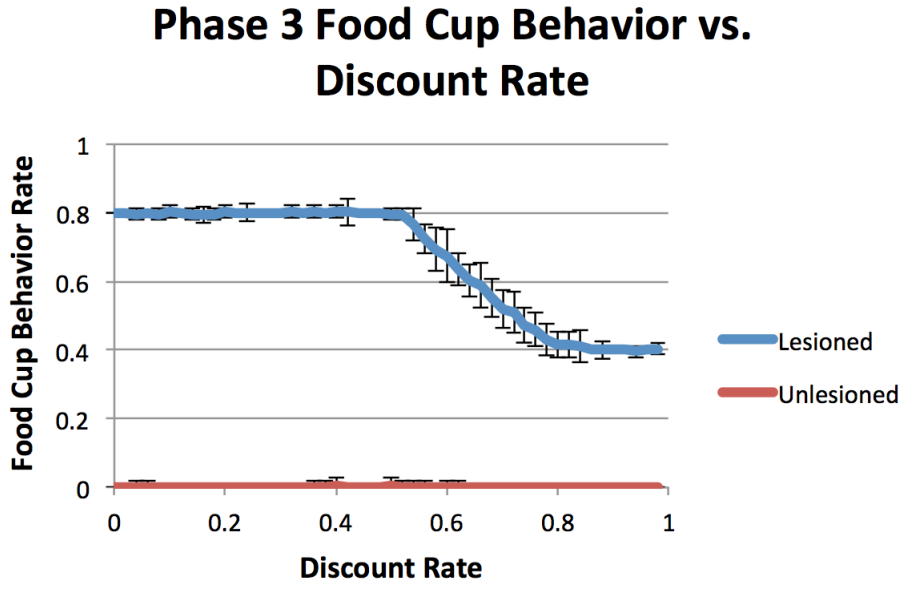


Figure 12. Phase 3 food cup behavior versus discount rate.

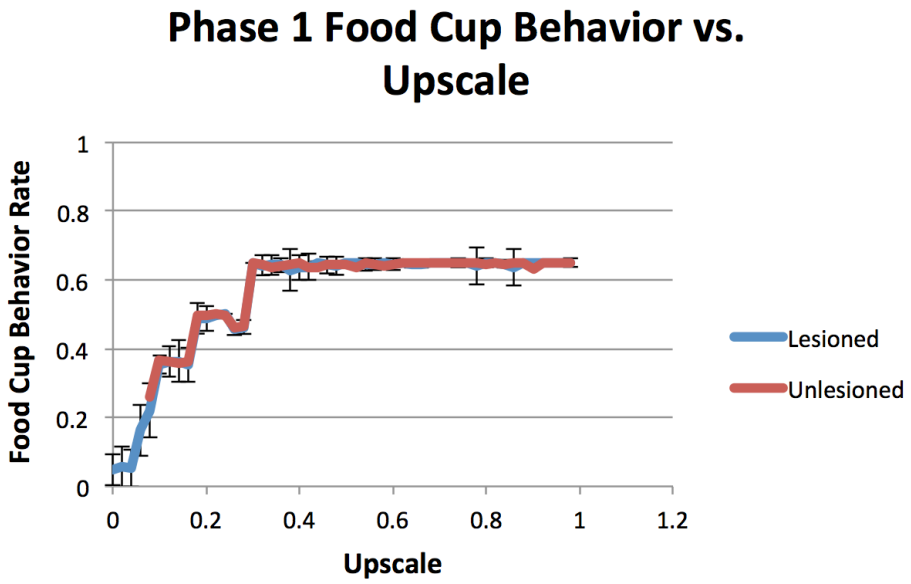
fixed parameter test the learning rate was 0.1. Here I explored values of the upscale from 0.0 to 1.0 for both lesion conditions. The results for Phase 1 of the experiment are given in Figure 13 and show a general increase in appetitive behavior as learning rate increases until stability at and above 0.3. There was no significant difference between lesion conditions. Increased upscale allows an agent to better anticipate future events. In Phase 1 this encourages the agent to eat the food since it would be more likely for the light cue to significantly activate the food node.

In Phase 3, upscale value had little effect on food cup behavior for either condition. There appears to be a small uptick in food cup behavior for the non-lesioned condition as upscale increases. It may be that increased upscale chiefly serves to bolster the relevance of the food node, but not the later event nodes with negative consequences. Similarly for the lesioned condition there may be a small decrease in food cup behavior with increased upscale (Figure 14).

In this section I have presented the results of the replication of a reinforcer devaluation experiment using a motivationally extended LIDA software agent. The agent successfully replicated the main result of the experiment. I also explored the effects of varying three critical internal parameters on the agent's behavior in the experiment.

Discussion

The Psi model (Bach, 2009) provides inspiration for some of the concepts in the motivation extension to LIDA. I discuss this relationship next, and outline it in Table 4. Psi distinguishes between demands, urges, motives, cognitive modulators, affect, and directed emotions. *Demands* are built-in drives that give rise to goals while *urges* are the



-

Figure 13. Phase 1 food cup behavior versus upscale.

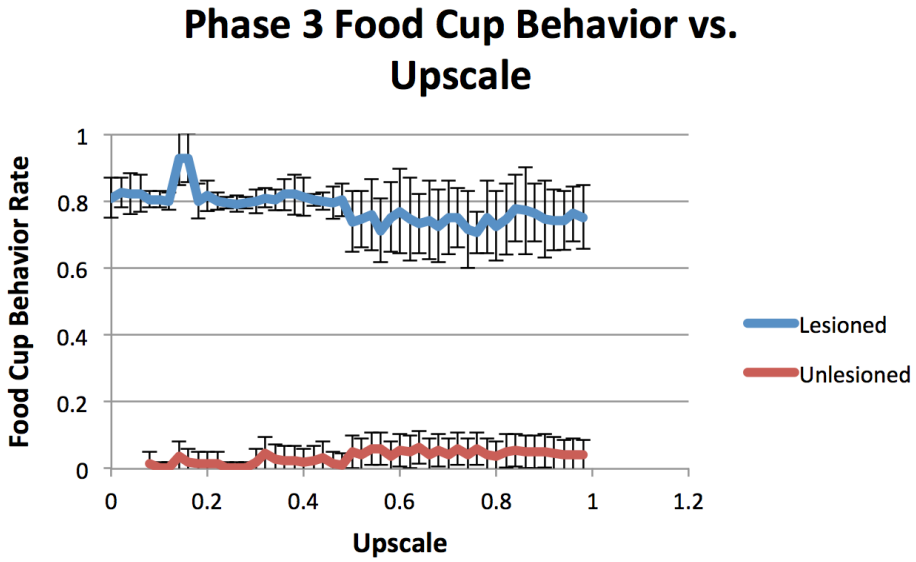


Figure 14. Phase 3 food cup behavior versus upscale.

Table 4. Approximate overlap of motivation concepts in the Psi and LIDA models.

Concept	Psi Term	LIDA Term
Built-in homeostatic mechanism	Demand (demand sensor)	Homeostatic sensor
Cognitive representation of a drive	Urge	Drive feeling node
Valence (Psychology)	Valence (kind of modulator)	Affective valence
Arousal	Arousal (kind of modulator)	Sum activation of feeling nodes
Desired (or aversive) event	Motive	Event with nonzero incentive salience
Emotion	Motives combined with perceptual representation	Feelings with cognitive content

signals that make demands apparent. In LIDA, *homeostatic sensors* and the sensors' associated *drive feeling nodes* implement drives. Psi theory suggests: "an abrupt increase of an urge corresponds to… a 'displeasure or distress signal'… while a decrease of an urge — its satisfaction — yields … 'pleasure signals'" (Bach, 2012). In Psi, a *motive* is defined as the combination of an urge and an event associated with the satisfaction of the urge. LIDA refers to motives as events with nonzero incentive salience. Psi theory asserts that goals depend on the nature of a given situation and the existing demands. Higher-level emotions such as jealousy and anger are said to arise in Psi from the combination of motives and the agent's perception of a situation. Fellous' work (2004) has been interpreted as providing a biological underpinning for the use of emotions in Psi (Lee-Johnson & Carnegie, 2009). Psi maintains that emotions should be distinguished from motivational phenomena: while hunger is cognitively represented (as an urge) and implies a valence and an effect on the allocation of mental and physical resources of an organism, it is not an emotion, but a *motivator*. Finally, Bach (2012) adopts the general

notion that motivation determines *what* has to be done, while emotion determines *how* it has to be done. In my view such an assertion is more problematic than elucidating.

One difference between the model introduced above and the Psi model is the way drive feeling nodes (urges in Psi) affect perception, action selection and action execution. In Psi urges interact with perception via modulators (Bach, 2003, Fig. 1; Bach, 2012, Fig. 6). In LIDA feelings can immediately modulate perception and action selection, given that conscious learning has previously occurred. Some feeling nodes are likely part of most coalitions, and affect learning if they win the competition for consciousness. Emotional valence is considered a representation of desirability (Bach, 2012), while in this model the two are distinguished, and play different functional roles (see sections "Affective Valence" and "Incentive Salience" above). Furthermore, it appears that in Psi, valence is a "reinforcement signal that emanates from changes in the perceived demands" (Bach, 2012). However, in this model, valence can also arise from intrinsically positive or negative stimuli, e.g., a sweet feeling node with positive valence. Lastly, the Psi model does not integrate cognitive neuroscience evidence for the "short route of emotion" (Cannon, 1927; Faghihi, Nkambou, Poirier, & Fournier-Viger, 2009; LeDoux, 2000), which have been referred to as *alarms*.

Future Work. In the section on the temporal difference learning of base-level incentive salience it was assumed that that the discount factor, γ, was fixed. However, changes in physiological state (e.g., stress) may serve to dynamically modulate this factor. Future work could, in general, explore the situations where emotions modulate the agent in a physiological way.

Another avenue of future research may involve the so-called, *eligibility traces*, which "trace" the credit assignment of temporal difference learning back further in time than the immediately previous event. Eligibility traces can be viewed as providing a short-term memory of multiple previous events so that several of these previous events may all be updated as each new observation arrives. Eligibility traces are usually implemented by an exponentially decaying memory trace, with decay parameter λ (Barto, 2007).

Conclusions

Here, I have explored how to implement the agenda of an autonomous agent in the systems-level LIDA cognitive architecture. To this end, I presented a motivation extension to LIDA covering a range of motivation-related concepts, including feelings, affective valence, incentive salience, emotion, appraisal, reinforcement learning, and model-free and model-based learning (McCall et al., in preparation). This extension adds new faculties to the LIDA model including drive feelings, affective valence, incentive salience, and temporal difference learning. I discussed the role of motivations in a single LIDA cognitive cycle, as well as over multiple cycles. To begin validating this motivational extension, I presented a computational implementation of a LIDA-based agent with some of these capabilities, which replicates the results of an existing "reinforcer devaluation" experiment, which tested the agent's ability to learn, and later update, the reward-predicting attributes of stimuli that drive its behavior. To better understand the working parameter set, I also explored the effects of varying the values of three key agent parameters on agent behavior.

4 Cortical Learning Algorithms with 2D Receptive Fields for a Systems-Level Cognitive Architecture

Introduction

In this chapter I consider how modern treatments of perception might be integrated into a broad systems-level cognitive architecture, LIDA. Having reviewed several inspirational models for this work including HMAX, Hierarchical Temporal Memory, predictive coding, Generalized Filtering, etc., I now strive to identify key principles across these various approaches to guide the research and development of an algorithm capable of generically assimilating a model of the data it samples. I argue that such an algorithm has applications to systems-level cognitive architectures. The organization of this chapter is as follows: I identify and justify guiding principles for perceptual representation, its related processes, and learning. Next, I perform a case study of Generalized Filtering, a biologically plausible application of the free-energy principle, to complex hierarchical dynamic statistical models. Generalized Filtering provides the mathematical and theoretical guide for this work that adapts the Cortical Learning Algorithms (CLA). I then describe the Cortical Learning Algorithms and my own 2D-CLA an extension adding 2D receptive fields to CLA (McCall & Franklin, 2013). The chapter finishes with reports of several tests of the CLA, which has seen relatively little in the way of published work. Figure 15 depicts where this work contributes to the LIDA architecture.

Perceptual Principles for Cognitive Architectures

Motivated by strong ideas within the models of perceptual analysis and learning reviewed in Chapter 2, this section identifies several key principles for perceptual representation, processing, and learning in cognitive architectures. To start, what evidence is there that

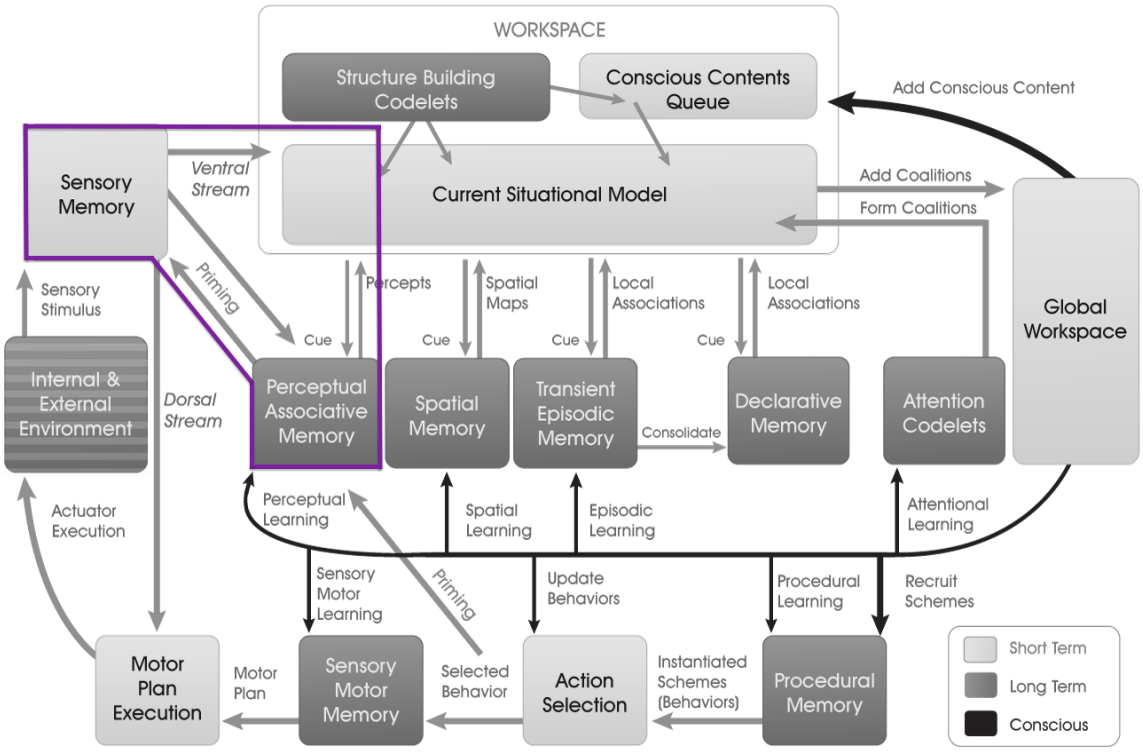


Figure 15. Contributions of 2D-CLA and PC-CLA Chapters to the LIDA model. The work in this chapter on 2D-CLA and that of the subsequent chapter on PC-CLA chiefly contribute to the LIDA model as implementations of the low-level Sensory Memory, Perceptual Associative Memory, and their associated current representations in a part of the Current Situational Model of LIDA's Workspace.

common perceptual principles might exist? Mountcastle (1978) proposed that there is a common function performed throughout the entire neocortex. The main idea is that a common cortical algorithm learns patterns of coincidence, and then sequences of coincidence patterns in sensory data (Hawkins & Blakeslee, 2004). Being grounded in time, such an algorithm appears to be generally useful across the various sensory modalities. Some early experimental work in support of this idea came from Métin and Frost (1989). They studied hamsters whose retinal ganglion cells were surgically adjusted at a young age to project to somatosensory areas. As adults, the hamsters' somatosensory neurons responded to visual stimulation in resemblance to normal visual cortical neurons.

For both experimental animals and controls, the same functional categories of neurons occurred in similar proportions, and the neurons' selectivity for orientation or movement direction were comparable. This suggests that the visual and somatosensory pathways perform similar transformations on their inputs. Other experiments using ferrets similarly rerouted visual inputs to the cortical areas typically involved in auditory processing. For instance, Roe, Pallas, Kwon, and Sur (1992) concluded that the sensory neocortex is not uniquely programmed to process inputs of a particular modality. Rather the audio cortex can process visual input, and the visual transformations performed by auditory cortex appear very similar to those of a normal visual cortex. Von Melcher, Pallas, and Sur (2000) also performed a similar rerouting experiment, reporting that their ferrets had visually responsive cells in the auditory thalamus and auditory cortex forming a retinotopic[1] map. Cells in the auditory cortex had receptive field properties typical of cells in visual cortex. The ferrets additionally exhibited behavior in response to a visual light stimulus.

In a similar vein, neuroscience research suggests that the neo-cortex may be implementing a large number of similar, hierarchically arranged "cortical circuits" (Douglas & Martin, 2004; Felleman & Van Essen, 1991; Phillips & Singer, 1997; Rockel, Hiorns, & Powell, 1980). While there is debate over the uniformity of such circuits (e.g., Meyer et al., 2013), variations might be understood as parametric refinements of common structure. Recently, Bedny, Pascual-Leone, Dodell-Feder, Fedorenko, and Saxe (2011) found that congenitally blind humans perform language processing in their visual cortices during some verbal tasks. More specifically, they found

[1] Retinotopy is the mapping of visual input from the retina to neurons, particularly those neurons within the visual stream.

that the left visual cortex behaved similarly to the classic language regions of the left frontal and temporal cortex. The authors concluded that brain regions thought to have evolved for vision could take on language processing as a result of early experience.

In a paper with similarities to this "Perceptual Principles" section, O'Reilly (1998) identified principles for "biologically based computational models of cortical cognition." He suggested the following advantages: "…integrating these principles allows one to demonstrate the consistency of different models, capitalize on synergies between different principles, organize and consolidate existing findings, and generate novel insights into the nature of cognition." His principles were: biological realism, distributed representations, inhibitory competition, bidirectional activation propagation, error-driven task learning, and Hebbian model learning. I directly discuss the first two of his principles, and mention the third under the heading "Sparse Distributed Representation." "Bidirectional activation propagation" comes up under the heading of the more specific hierarchical predictive coding, a consequence (under certain assumptions) of the free-energy principle, which I identify in this section. Similarly Hebbian-style, prediction-error-driven learning also stems from the free-energy principle (see the later section "Parameter Optimization via Associative Plasticity").

Converging Evidence Approach. While machine learning approaches emphasize efficient, high-performance algorithms, cognitive architectures require perception algorithms to integrate within the architecture. Biologically inspired cognitive models add the further constraint of biological plausibility. O'Reilly (1998) succinctly qualified biological plausibility arguing that: "Although the issue of biological realism is easy to state, it can be difficult to apply, because the known biology often does not provide

sufficient constraints. Thus, biological realism often reduces to plausibility arguments, which depend on things like how simple and local the mechanism in question is, and that it is not inconsistent with known biology." He goes on to suggest a converging evidence approach where multiple constraints from biology, computation, and cognition converge to support a given principle. In this spirit, I present my own set of principles below, justified by converging evidence from disciplines including biology, information theory, and mathematics.

Autonomy and Agency. Biologically plausible, autonomous, learning agents demand that learning occur online without stoppages. This still allows for the possibility of a developmental period, not unlike human development, during which an agent learns rapidly in a natural setting. Secondly, agents must be able to learn autonomously without the aid of labeled data, i.e., to perform unsupervised learning. Another critical aspect of agency is motivations, which are beyond the scope of this chapter; nonetheless there are ideas on how to implement motivation in hierarchical Bayesian networks (e.g., Friston et al., 2009).

Hierarchical decomposition. Hierarchical decomposition is an idea that "pervades almost all attempts to manage complexity" (Russell & Norvig, 2009). Decomposition using hierarchical structure allows, at the non-root levels of a hierarchy, a computational task "to be reduced to a small number of activities at the next lower level so the computational cost of finding the correct way to arrange those activities [...] is small" (Russell & Norvig, 2009). In the context of internal representations for agents, many patterns (at non-root levels) can be constructed from patterns in the level immediately below, achieving substantial pattern reuse. For example, consider a hierarchically

decomposed memory of sentences where the 1st level stores letter patterns, the 2nd level, word patterns, and the 3rd, sentence patterns. Given several stored words patterns at the 2nd level, it is more efficient to construct and store sentence patterns (at the 3rd level) using patterns from the 2nd level (words) than out of 1st level patterns (letters). Note that patterns can be hierarchically decomposed using co-occurring features in the input space. For example, the representation of a face can be based on a set of co-occurring features including the eye, nose, mouth, etc. Likewise, patterns can be decomposed temporally, using sequences of features. Returning to the previous example with sentences, we can have the 3rd level storing and representing a single pattern representing an entire sentence based on temporal sequences of word patterns in the 2nd level.

There is considerable evidence that brains use hierarchical decomposition dating back to Mumford (1992). Fuster's "cognits" model of cortical representation (Fuster, 2006, 2007) also includes hierarchy. According to Fuster, *perceptual cognits* are cognitive networks initially made of neurons associated by information acquired through the senses, while *executive cognits* are made of neurons associated by information related to action. In both cases, cognits are hierarchically organized. At the bottom of the hierarchy, cognits are small and relatively simple, representing simple percepts or motor acts. At the top, cognits are wider and represent complex and abstract information of perceptual and/or executive (motor) character. Perceptual and motor networks are associated with each other in cortico-cortical connections. These connections support the dynamics of the action-perception cycle (Dijkstra et al., 1994; Freeman, 2002; Neisser, 1976) in sequential behavior, speech, and reasoning (Fuster, 2006).

Sparse Distributed Representation. Snaider (2012, pp. 53–57) reviewed distributed representations in connectionist systems. In localist representation, each unit represents a single object, concept, or element of the system, and the represented elements have a one-to-one correspondence to the system's units (Franklin, 1995, p. 132). The main advantage of a localist approach is the explicit representation of data. Currently, the Perceptual Associative Memory module in the LIDA architecture (Ramamurthy & Franklin, 2011) follows this paradigm. In localist representation, passing activation among units can explicitly implement constraint rules, e.g., is-a, if A then B, and if A not B. Also, the excitation of a unit based on the activation of other connected units can model similarity and composition of elements. Despite these advantages, this type of representation is inefficient since the one-to-one correspondence between items and units in the system requires *n* units to represent *n* items (Snaider, 2012).

With distributed representations several units represent each pattern, and each unit can participate in the representation of many patterns (Franklin, 1995, p. 132; Hinton, McClelland, & Rumelhart, 1986). Each unit in a distributed representation can be thought of as representing a single *microfeature* (Hinton et al., 1986), with the information being encoded by particular combinations of such features and not by a single bit or feature. With distributed representation, explicitness in representation is lost; however, there are several advantages. Firstly, it is more efficient than localist representation, e.g., for binary vectors, only units are required to represent *n* patterns, whereas localist representation would require *n* units. Furthermore, similar patterns have similar representations, because they may share several features. This can lead to automatic generalization, since similar items will activate a similar pattern of units (Franklin, 1995, pp. 132–133). Distributed

representations can implement what Plate (2003, p. 13) calls *explicit similarity*; i.e., similar patterns have similar representations. Electrophysiological recordings demonstrate that distributed representations are widely used in the cortex (e.g., Rao, Rainer, & Miller, 1997).

The idea of sparse representation can be traced to the infomax principle (Linsker, 1990), which suggests the brain maximizes the mutual Shannon information between sensations and representations. Infomax also suggests that internal representation should efficiently and parsimoniously encode sensory data, and provides an information-theoretic view of the sparse coding hypothesis of Olshausen and Field (1996), which suggests that sensory patterns are represented by the strong activation of a relatively small set of neurons. How can a code be made sparse? One biologically plausible answer is inhibitory competition, which arises when mutual inhibition among a set of units (e.g., via inhibitory interneurons), prevents all but a sparse subset of them from becoming active at a give moment (O'Reilly, 1998). Inhibitory competition allows only the most strongly activated representations to win. Approximately 20% of the neurons in the cortex are believed to be inhibitory interneurons (Gabbot & Somogyi, 1986). More recently, Rinkus (2010), proposed that cortical macrocolumns are functional detectors of, and repositories for, sparse distributed representations of inputs, and that the "generic function of the minicolumn is to enforce macrocolumnar code sparseness."

What is the value of sparse coding? One answer is that sparse representations are highly unique and thus robust. Kanerva (1988) considered high-dimensional binary spaces where each point, x, in the space has exactly one opposite x'. He showed for a space N, having dimensionality n, that for an arbitrary point, x, the average Hamming

distance[2] to any of the other points in space is roughly half the distance between x and x'. More specifically, the distribution of the other points as a function of Hamming distance is approximately Gaussian with a mean distance to x of $n/2$ and standard deviation of $\sqrt{n/2}$ (Snaider, 2012). Thus, for sufficiently large n, say 1000 and greater, the distribution becomes quite concentrated around $n/2$. Suppose a pattern y has been stored in a memory, then this property implies that there are few other patterns close to y, and, if a similar pattern y^* is encountered, it is reasonable to consider y^* as a version of y.

Use of Prediction. William James (1890) once noted: "Enough has now been said to prove the general law of perception, which is this, that whilst part of what we perceive comes through our senses from the object before us, another part (and it may be the large part) always comes out of our own head." James' law is consistent with modern views of perception that see the brain as a constructive or predictive organ actively generating predictions of its sensory inputs using an internal or generative model[3] (Friston, 2012). Such a view can actually be traced back to Helmholtz's original writings on unconscious inference, e.g., "Objects are always imagined as being present in the field of vision as would have to be there in order to produce the same impression on the nervous mechanism" (von Helmholtz, 2005).

More recently, Bar (2009) suggested a framework portraying a proactive human brain that continuously generates predictions anticipating the relevant future. In this framework, associations (from the cueing of memory) bias representations of "what is most likely to occur and be encountered next." Such associations vary in scale and in the

[2] The Hamming distance between two strings (here vectors) measures the minimum number of substitutions required to change one string into the other.

[3] In probability and statistics, a generative model is a model for probabilistically generating observable data from some distribution, typically given some hidden parameters.

automaticity of their activation from automatic, simple and unique, similar to a basic Hebbian association, to more deliberative associations. Associations are formed through experience, based on similarity and frequent co-occurrence in space and time. The framework is supported by demonstrations showing a striking overlap between the cortical network that mediates contextual associative processing (Bar, Aminoff, Mason, & Fenske, 2007), and the cortical network termed the "default network," (Raichle et al., 2001) suggesting that the brain continuously cues its memory "by default" as does the LIDA model.

What are the advantages of continual predictions and associations? According to Bar, one advantage is that seemingly random thoughts and aimless mental simulations are actually helping to create memories. Agents can benefit from such memories without the external events ever taking place. Another benefit may be that the predicted or expected aspects of an experience can be used to compute a prediction error, which is deemed salient and, therefore, comes to consciousness and is learned. The predictable aspect of the experience is often not salient and not learned. Then, accurate predictions can help obviate the need to allocate attention (and learning) towards predictable aspects of the environment. This frees resources to explore the environment for novelties from which we can learn, and surprises we should avoid. Generating predictability based on our experience is, therefore, a powerful tool for detecting the unexpected (Bar, 2009).

Predictive coding theory suggests that local transformations can provide the benefit of reducing the dynamic range (the ratio between the largest and smallest possible values of a changeable quantity) required to encode potentially highly variable stimuli, e.g., natural image signals (Huang & Rao, 2011). In addition to these considerations,

HTM theory (Hawkins et al., 2011) suggests additional advantages of predictions. Predictions can be used to produce representations with invariance over time. For instance, suppose that a network learns a temporal pattern "A, B, C, D" and can predict expected patterns in advance. Later, if the pattern A is first input then the network would build a *single* representation not only of A but, due to predictions, expected future inputs (e.g., B, C) as well. If B appears next then the network will form a single representation of B, and via predictions, C and D as well. Since these two representations include aspects of B and C, the representation exhibits some invariance over time. Furthermore, for distributed representation, temporal predictions can maintain an internal representation, over time, in the face of incomplete, corrupted, and/or noisy input.

At this point it is important to distinguish two different kinds of prediction. Within the context of hierarchical Bayesian inference, I distinguish both 1) *top-down predictions* (structural priors in probabilistic terminology), which emanate "top-down" from one hierarchical level into the immediately subordinate level, and 2) *temporal predictions* (dynamical priors), encoding the expected temporal dynamics using "lateral" or recurrent links. In relation to this, it is also important to qualify that, for this work, "prediction" does not specifically refer to the high-level psychological phenomenon that is akin to a deliberative judgment. Rather, it is more aptly viewed as a process within a hierarchical, dynamic (recurrent) network producing 1) dynamic expectations of the current state based on the previous state(s) at that level, and 2) top-down expectations for a given hierarchical level stemming from a current belief at the next higher hierarchical level.

Finally, hierarchical predictive coding (Rao & Ballard, 1999) involves top-down predictions whereas in generalized predictive coding (Friston et al., 2010), structural priors dynamically reshape the trajectory of a low-level attractor, and, additionally, the low-level attractor itself also contains an abundance of dynamical priors (predictions) that encode expected higher-order dynamics. Friston and Kiebel (2009; 2011) describe a simulation demonstrating the necessity of structural and dynamical priors for accurate perception. They produced artificial "birdsongs" using two time-varying control parameters governing the frequency and amplitude of vibrations emanating from an artificial syrinx. The syrinx was driven with two states of a Lorenz attractor, one controlling the frequency and the other controlling the amplitude or volume. The parameters of the Lorenz attractor were chosen to generate a short sequence of chirps every second or so. To endow the songs with a hierarchical structure, they placed a second Lorenz attractor; whose dynamics were an order of magnitude slower than the first. That is to say the states of the slower attractor entered as control parameters to control the dynamics of the first. They decoded the resulting sonogram with a "synthetic bird," a data analysis algorithm (dynamic expectation maximization) that infers the hierarchical and dynamic structure of its input. Then, they tested the effects of selectively destroying these two types of priors (structural and dynamic) by lesioning the structural connections to remove structural priors, or by cutting the intrinsic connections that mediate dynamical priors. In both situations, the synthetic bird cannot recognize the sequence, and exhibits an inflation of prediction error. Interestingly, the removal of structural priors has a less marked effect on recognition than removing the dynamical priors. Without dynamical priors, there is a failure to segment the sensory stream and,

although there is a preservation of frequency tracking, the dynamics per se have lost their sequential structure. Although it is interesting to compare and contrast the relative roles of structural and dynamics priors, the important message here is that both are necessary for veridical perception, and that destruction of either leads to suboptimal inference. Both of these empirical priors prescribe dynamics, which enable the synthetic bird to predict what will be heard next (Friston & Kiebel 2009, 2011).

Approximate Bayesian Inference. Kording and Wolpert (2004) reviewed studies showing a close correspondence between human decision-making and those judgments made by Bayesian decision theory, which defines optimal behavior in uncertain environments where signals in sensory and motor systems are corrupted by variability or noise. Other work along these lines has suggested that the nervous system combines visual and haptic information in a fashion that is similar to a maximum-likelihood integrator (Ernst & Banks, 2002), while Lee and Mumford (2003) have suggested that the visual cortex performs hierarchical Bayesian inference, and that the "recurrent feedforward/feedback loops in the cortex serve to integrate top-down contextual priors and bottom-up observations so as to implement concurrent probabilistic inference along the visual hierarchy." Knill and Pouget (2004) popularized the "Bayesian coding hypothesis," which states that the brain represents information probabilistically, by coding and computing with probability density functions or approximations to them. Most recently, Tenenbaum et al. (2011) presented an approach to understanding cognition in terms of Bayesian inference over richly structured, hierarchical generative models. They argued this provides a "unifying mathematical language for framing

cognition as the solution to inductive problems and building principled quantitative models of thought with a minimum of free parameters and ad hoc assumptions."

The Free-Energy Principle. The free-energy principle (Friston, 2005, 2009, 2010) is a recent theory of brain function that accounts for action, perception and learning. The principle unifies several existing theories including predictive coding, the Bayesian brain hypothesis[4] (Knill & Pouget, 2004), infomax (Linsker, 1990) and sparse coding (Olshausen & Field, 1996), associative plasticity, optimal control, and value learning (see Friston (2010) for a review). A common thread among these all of these theories is optimization. Some of these theories consider the optimization of value (e.g., expected reward, expected utility). Others consider the optimization of a complement of value; namely, self-information[5] or "Shannon surprise" (e.g., prediction error). The inverse relationship between value and surprise can be seen if we consider defining those entities that are valued to also be ones that are predicted or expected. Then, for example, a prediction error is a failure to optimize value. In the context of this work on perception, I discuss the matter as the optimization of prediction error. Also, it is worth noting, that the free-energy principle accounts for the earlier stated principles "Sparse or Efficient Coding" and "Approximate Bayesian Inference" (Friston, 2010).

"Free energy" in this context refers to the *variational free energy*, F, originally used in statistical physics, which can be defined as follows:

$$F = \int q(\vartheta) \ln \frac{p(s, \vartheta)}{q(\vartheta)} d\vartheta \qquad [5]$$

[4] The hypothesis that the brain updates its beliefs with incoming sensory information in an approximately Bayes-optimal way, using internal probabilistic generative models.

[5] The negative log-probability of an outcome, which implies that improbable outcomes are "surprising."

where s is some data, ϑ is a set of parameters, $p(s, \vartheta)$ is the true joint distribution, and $q(\vartheta)$ is an approximating posterior distribution modeling the data. Variational free energy can be understood as providing a bound on the natural log of the probability of sampling the data. Given a probabilistic generative model, m, of the data, free energy can be expressed in terms of this log-probability and a non-symmetric measure of the difference between two probability distributions called the Kullback-Leibler (KL) divergence (Equation 6). The KL-divergence between two probability distributions, a and b, is denoted by $KL[a \| b]$.

$$F = \ln p(s \mid m) - KL[q(\vartheta) \| p(\vartheta \mid s, m)] \qquad [6]$$

Since KL is always positive and log-probabilities are always negative or zero, F provides a lower bound on the model evidence, $\ln p(s \mid m)$. That is, F is less than or equal to the model evidence. Equivalently, F can be viewed as an upper bound on self-information. If the approximate posterior $q(\vartheta)$ equals the true posterior $p(\vartheta|s)$, then KL is zero and F equals the model evidence (Figure 16).

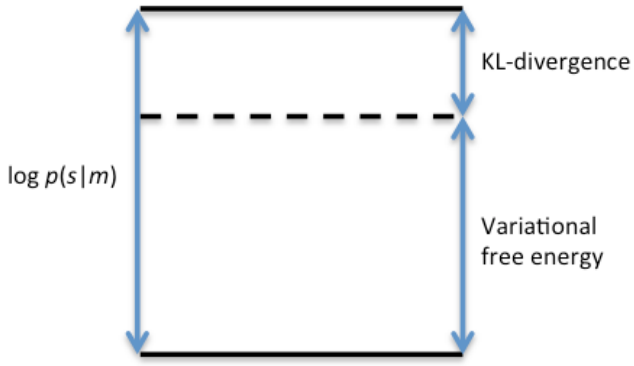


Figure 16. Decomposition of the log-model evidence. The log-model evidence can be expressed in terms of two quantities: the always-positive KL divergence and the variational free energy, F, which lower bounds the model evidence and upper bounds Shannon surprise.

Succinctly, the free-energy principle suggests that the brain changes to minimize its free energy, which bounds (by being greater than) the Shannon surprise on sampling some data, given a generative model (Friston, 2010). How can an agent minimize free energy? It cannot know whether its sensations are "surprising" in the Shannon sense, since this would require it to know the true probability of every outcome. This is where free energy comes in as an upper bound on surprise, which means that if an agent minimizes free energy, it implicitly minimizes surprise. Unlike surprise, free energy can be evaluated because it can be expressed as a function of two things to which the agent has access: its sensory states and a *recognition density* encoded by its internal states (e.g., neuronal activity and connection strengths) (Friston, 2010). The recognition density is an approximate conditional probability distribution of the causes of data (here, sensory input), and is updated by probabilistic inference or "inverting a generative model" in the statistical parlance. There are several approaches in the literature to approximate probability distributions, for instance, free-form particle-based approaches, as well as parametric approaches that assume fixed forms (see Friston 2009 for a discussion).

In the next section, I provide a high-level overview of an application of the free-energy principle to hierarchical dynamic state-space models showcasing an approach termed Generalized Filtering (GF). Among the appealing characteristics of GF include several of the principles identified in this section. For starters, GF was constructed with neurobiological plausibility in mind (Feldman & Friston, 2010; Friston et al., 2010, p. 28), and conforms to the free-energy principle, thus taking what we have identified as a Converging Evidence Approach. The scheme operates online and unsupervised, in keeping with the objectives stated in the Autonomy and Agency section. The internal

models considered in Generalized Filtering have hierarchical form, and are generative models, which predict their inputs. Finally, the approach involves the estimation of statistical uncertainty, and can be seen as implementing a form of hierarchical Bayesian inference (Efron & Morris, 1972; Kass & Steffey, 1989). Incorporating many of the previously stated perceptual principles, I adopt Generalized Filtering as a theoretical guide for the proposed PC-CLA network, detailed later in this dissertation.

Applying the Free-Energy Principle

Friston et al. (2010) applied the free-energy principle to derive a Bayesian filtering[6] scheme called Generalized Filtering (GF). This method seeks equations for the approximate Bayesian inversion (recognition) of statistical models, which are quite sophisticated and general in their form. The approach requires arguably reasonable assumptions, and produces relatively simple, biologically plausible, update equations for model parameters[7]. More specifically, GF considers quite general hierarchical, dynamic, generative statistical models consisting of hidden state variables, the dynamics of which can be influenced by an appropriate nonlinear function (equation of motion). Furthermore, at each level, hidden state variables may be subject to (random) stochastic fluctuations (uncertainties or inverse precisions) representing these variables' statistical uncertainty.

[6] Also called recursive Bayesian estimation, this is a general probabilistic approach for estimating an unknown probability density function, recursively over time, using incoming measurements and a mathematical process model.

[7] In this context, parameters are not assumed to be fixed; rather they are continually estimated and updated.

I will discuss uncertainty in terms of *precision*, the multiplicative inverse of the variance[8]. High statistical precision implies a probability distribution that is highly concentrated about its mean suggesting that the mean is known with a high degree of precision in the ordinary sense. In terms of statistical model inference, precision is a hyperparameter to be estimated from the data. I will refer to such hyperparameters that parameterize estimations of precision as *precisions*. Precisions in GF's form of generative model may themselves undergo state-dependent changes in amplitude, and can have arbitrary autocorrelation[9] functions influencing their time evolution. An important aspect of this model is its hierarchical form, which induces top-down priors (predictions) into subordinate levels. Figure 17 illustrates these features at one hierarchical level.

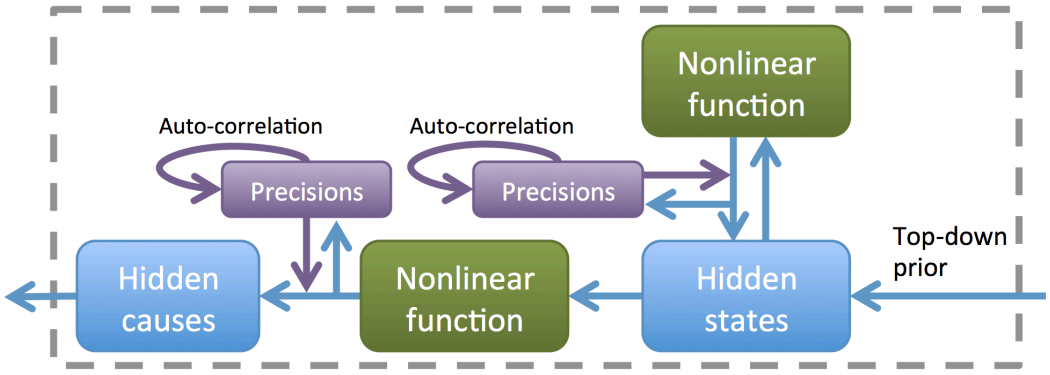


Figure 17. A single hierarchical level in Generalized Filtering. The dashed lines bound one level with the lower level to the left and the higher level to the right. The model's state variables are depicted by the blue boxes. Two nonlinear functions and their parameters (green) govern both top-down and dynamical priors (or predictions). Two corresponding sets of precisions (purple) affect the hidden states and vice versa. The dynamics of precisions are governed by some autocorrelation function.

[8] For a multivariate distribution, the precision takes the form of a precision matrix, which is the matrix inverse of the covariance matrix, if such an inverse exists.

[9] Autocorrelation is the correlation between a signal and itself at some offset in time as a function of the offset.

The next key feature of Generalized Filtering is the representation of internal states and hidden parameters in a generalized coordinate system. That is, one where the coordinates describe the configuration of the system relative to some reference configuration with the constraint that the set of coordinates must *uniquely* define the configuration of the system relative to the reference configuration. For example, the location of the end of a pendulum can be defined in terms of a coordinate representing the angle of the pendulum arm from its stationary position, which simplifies calculations about the dynamics of the pendulum.

For GF, the generalized coordinates include the variable values and the infinite set of the variables' higher order derivatives. In such a coordinate system, each point includes a variable's current value and the higher-order derivatives of its equations of motion. Such a point can be viewed as encoding the variable's current trajectory since future variable values can be extrapolated using the higher order terms. To illustrate, if x represents a set of state variables, then this coordinate system considers $\tilde{x} = \{x, x', x'', ...\}$. With generalized coordinates of motion, instead of simply estimating the conditional density of hidden states (and parameters), one optimizes the conditional density on their generalized motion to an arbitrarily high order. In practice, the representation in generalized coordinates can be truncated at relatively low order (Friston et al., 2008, p. 861).

The third main feature of GF is its use of the Laplace assumption, which entails a fixed-form, Gaussian, approximation to the agent's internal probabilistic representations of the causes of sensory data. While the Gaussian distribution has several advantages (see Friston, 2009, pp. 297–298 for a discussion), a critical one is that, when free energy is

minimized, the conditional covariance of hidden states becomes an analytic function of the conditional mean. This implies that only the mean has to be optimized since the covariance can be derived from it. This simplifies the mathematics of the model optimization to a single ordinary differential equation describing the motion of the conditional mean of states (Friston et al., 2010).

In summary, given the above form of the generative models, treatment of variables in generalized coordinates of motion, and Gaussian assumptions on the recognition density (approximate conditional probability distribution of the causes of data), GF prescribes a set of ordinary differential equations governing recognition dynamics, which update the conditional density on a model's states (e.g.,hidden state variables), parameters (e.g., weighted links), and hyperparameters (e.g., uncertainties or precisions). For additional details, including the mathematical derivation, please see Friston et al. (2010) and Feldman and Friston (2010). In the next subsections I discuss GF's update equations for perception, learning, and attention in detail, but in a conceptual manner.

State Optimization via Prediction Error Minimization. The update equation for the model's hidden state suggests that instances of two kinds of computational units send and receive messages at each hierarchical level (Figure 18). *State units* (blue) encode the mean vector of the multivariate Gaussian distribution over unknown states generating sensory data, while *error units* (red) encode prediction error. The activity of each error unit is a function of state units (or the input), while state unit activity is a function of error units. State units provide top-down and temporal predictions to same-level and lower-level error units, respectively. Hierarchical inference requires only the prediction error

from the lower level, which drives the conditional expectations toward a better prediction, so as to minimize the prediction error in the level below. This recurrent message passing between hierarchical levels and its processing optimizes free

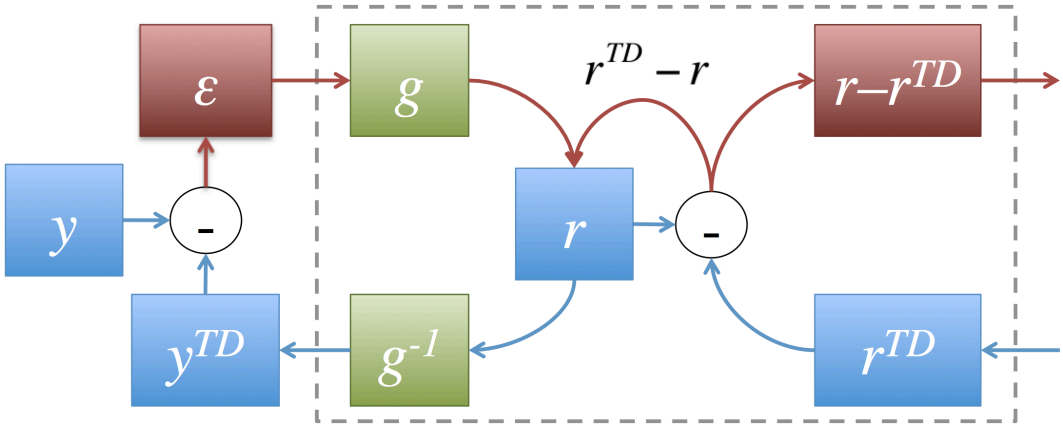


Figure 18. Hierarchical predictive coding. Each hierarchical level (dashed box) receives top-down predictions, r^{TD}, of the level's current representation, r, of the input signal, y. Feedforward pathways carry errors (e.g., $r–r^{TD}$) between the predictions and the actual activity. Errors ($r^{TD}–r$) are used to correct the current representation, r, and, hopefully, future top-down predictions, y^{TD}. g represents a set of feedforward connection weights and g^{-1} a set of feedback connection weights. Redrawn from Rao and Ballard (1999).

energy by minimizing prediction error, and constitutes perceptual inference (Friston & Kiebel, 2009). It is also known as (hierarchical) predictive coding (Rao & Ballard, 1999). Note that prediction errors can be based on top-down predictions (r^{TD} in Figure 18) or on temporal predictions (not shown). Temporal prediction error would be based on a temporal prediction and the actual future state.

Parameter Optimization via Associative Plasticity. Associative (or Hebbian) plasticity is another way in which a model may reduce free energy, in this case by updating the model's parameters. For example, such parameters modulate the effects of the processes, g and g^{-1}, in Figure 18. GF furnishes parameter update equations quite similar to models

of associative plasticity based on correlated pre- and post-synaptic activity. In terms of a computational network, a parameter value corresponds to the weight of a connection between two computational units. For example, the top-down process, g^{-1}, might use a connection between two units to produce the top-down prediction. The connection would constitute a model parameter, and its current weight would correspond to the parameter's current value.

Parameter optimization comes in two forms in hierarchical dynamic models: structural and dynamic. The structural parameter update is illustrated in Figure 19a, and considers a currently active hidden variable at a given level (e.g., Level 2) as the connection's source, and a currently active bottom-up prediction error variable as the sink. Such an update changes future top-down predictions, to hopefully better minimize future bottom-up prediction error. Figure 19b illustrates a dynamical update. Initially, there was a state variable, *p*, active at time *t* from bottom-up prediction error. This update takes a state variable, *q*, at the same level whose activity preceded *q*'s at time $t - 1$, and updates a dynamical connection from *q* to *p*. At a future time $t + n - 1$, the re-activation of *q* produces a temporal prediction of *p* (shaded), to hopefully minimize future prediction error. In either case, the parameter value is updated as a function of 1) the product of the q^{th} pre-synaptic input and post-synaptic response of the p^{th} error unit, 2) the product of the parameter value and its function's associated precision, and 3) a value-dependent decay, e.g., sigmoidal decay (Friston & Feldman, 2010, p. 8).

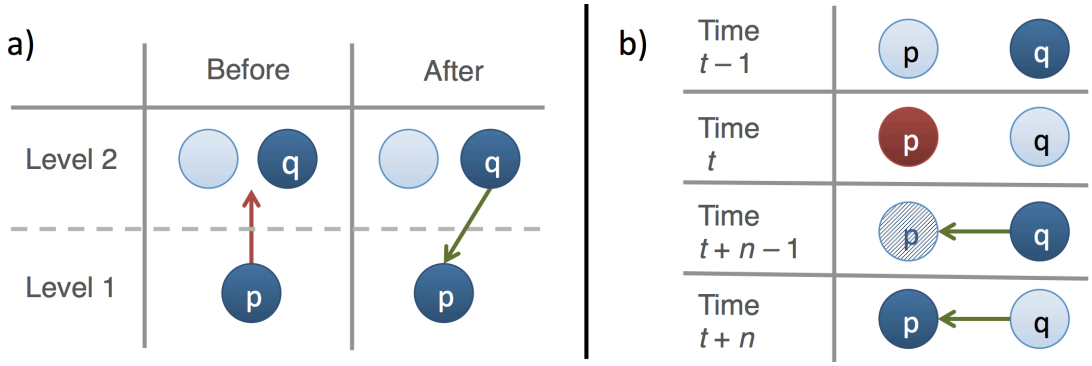


Figure 19. Structural and dynamical parameter updates. Both updates are performed in response to bottom-up prediction errors (red). a) A structural update modifies (or adds) a structural connection (parameter), from a higher to a lower level, which changes future top-down predictions, to (hopefully) minimize future bottom-up prediction error. b) A dynamical update modifies (or adds) a dynamic connection between two hidden state variables in the same level changing future temporal predictions to minimize future bottom-up prediction error.

Hyperparameter Optimization via Gain Control. Uncertainty plays an important role in GF, where it can be viewed as a *hyperparameter*, a variable that parameterizes a distribution of another parameter in a model (Bishop & Nasrabadi, 2006). How does GF suggest that the optimization of uncertainty occurs in a hierarchical, dynamic, predictive coding network? At each level in the hierarchy, the uncertainty weights the relative influence of the prediction error from the level below on the level in question. Recall that the bottom-up error is calculated based on the previous top-down prediction and the actual state occurring in the level below. Similarly, for the level in question, top-down prediction error is calculated based the received top-down prediction and the level's current hidden state (see Fig. 3). In GF, the uncertainty is expressed in the terms of a precision (inverse covariance) that weights prediction error. The larger the variance (i.e. the lower the precision) in the source of information, the less weight is given to that source, as is consistent (Rao, 1999) with Kalman filtering (Kalman & Bucy, 1961).

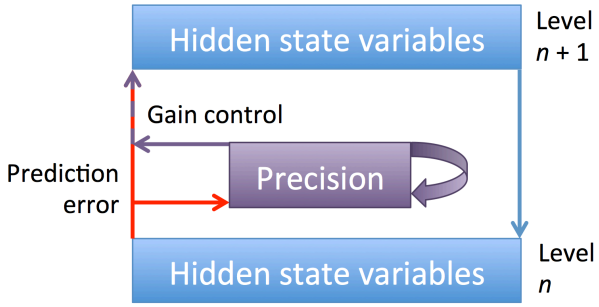


Figure 20. A single precision hyperparameter in a hierarchical predictive coding network. It is updated based on the sum of squared bottom-up prediction (red) and a value-dependent decay (purple arrow). It modulates the magnitude of the same prediction error signal from which it is estimated. Precisions can similarly exist for temporal prediction errors at each hierarchical level.

More specifically, for the form of models considered Generalized Filtering, precision parameters are state-dependent, and govern the gain control of error units. *Gain control* refers to the control of synaptic gain or post-synaptic responsiveness. As with parameter optimization, precisions change as a function of 1) the precision-weighted sum of a squared prediction error term and 2) a decay in proportion to the current precision value (Figure 20).

Synchrony and neurotransmitters have been suggested as the neurobiological implementation of gain control. This selective modulation of inputs is equivalent to the gain-control mechanisms invoked for attention (Martinez-Trujillo & Treue, 2004).

In this section I have discussed the application of the Free-Energy principle to complicated hierarchical dynamic stochastic models including the update equations required to perform inference on model states, parameters and hyperparameters. In this chapter, I next introduce the HTM Cortical Learning Algorithms (CLA) and an implementation with 2D receptive fields. In the next chapter I present a predictive coding extension to the CLA, termed PC-CLA, which is guided by the ideas in this section from Generalized Filtering. PC-CLA is proposed for use throughout LIDA providing

algorithms and data structures for LIDA's current representations, memory, and attention and learning processes.

CLA Implementation

Here I describe my implementation of the HTM Cortical Learning Algorithms (CLA) (Hawkins et al., 2011). The HTM Cortical Learning Algorithms is a recent approach to generic pattern recognition inspired by the organization of the cerebral cortex. While earlier iterations of the HTM theory have enjoyed publications by its creators (George & Hawkins, 2009; Hawkins & Blakeslee, 2004; Hawkins, George, & Niemasik, 2009), the CLA has not; however, some other intrepid researchers have made efforts to study the algorithm (Thornton, Main, & Srbic, 2012; Thornton & Srbic, 2013; Thornton, Srbic, Main, & Chitsaz, 2011). Here I present an implementation of the HTM Cortical Learning Algorithms, called 2D-CLA, which modifies the receptive fields of the algorithm to have limited overlapping 2D receptive fields. Existing neural network models employ similar receptive fields (see e.g., LeCun & Bengio, 1995). Such an extension allows 2D-CLA to deal with real-world input (i.e., visual or auditory stimuli). The tests in this chapter explored some of the basic properties of the spatial pooling portion of 2D-CLA on artificial 2D patterns.

The CLA takes Boolean vectors as input. It is worth mentioning that, for best performance, input patterns should, on average, be as uniformly distributed throughout the input's dimensions as possible. This constraint arises because the CLA is limited in its ability to produce representations that are more distributed than those it receives, and insufficiently distributed representations do not enjoy the advantages of high-dimensional spaces. "Natural" auditory and visual stimuli fit this distribution requirement quite easily

since they are composed of many pieces, which are apprehended by a distributed array of sensors. In contrast, for human-generated data, e.g., a time series of computer mouse pointer coordinates, a preprocessing step may be required to produce distributed inputs.

From here I first describe the original CLA algorithm in terms of data structures and algorithms. Next I describe the 2D receptive field modification. Then I describe several experiments exploring some of the basic properties of the implementation, including the effects of varying its parameters, which are, to my knowledge, undescribed elsewhere in this way.

Cortical Region. The main data structure of the Cortical Learning Algorithms is called the *cortical region* (Figure 21). A cortical region consists of a set of *columns*, which are based on the functionality of cortical minicolumns (Buxhoeveden & Casanova, 2002). In this implementation, I focus on a square 2D column arrangement, although other arrangements are possible.

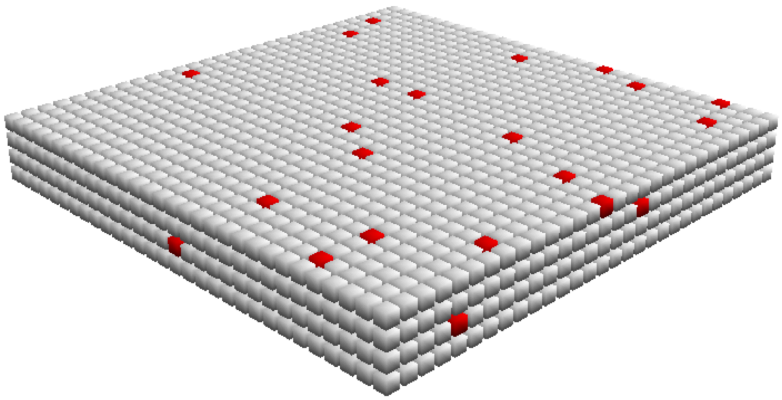


Figure 21. A depiction of a single cortical region. The region shown has 900 (30x30) columns, arranged in a 2D fashion, with each column having 4 cells. It is typical for implementations to have 1024–2048 columns.

Each column is comprised of several *cells* (explained below), and has one *proximal dendrite segment*, a functional approximation to a neuronal dendrite segment. The proximal dendrite segments integrate the activity of artificial *proximal synapses* (described later), implementing connections in close *proximity* to the column. Proximal synapses respond to active inputs received by the region. Figure 22 depicts a single column, its proximal dendrite segment and the segment's proximal synapses. The activity of the proximal synapses on a proximal dendrite segment determine its column's *overlap score*[10], a scalar measure of column activity from the bottom-up input. Each column also has a *boost* attribute, which weights the *overlap score*, and is based on the column's recent history of activity. *Boost* is used to make under-utilized columns more salient. Finally, each column has a *column activity state* which may be *inactive*, not sufficiently active to compete with other columns, *competitive*, sufficiently active to compete with other columns, or *active*, competitive and having won the competition with its neighboring columns to determine the most salient columns. A column's activity state is based on its boosted *overlap score* and *predicted column overlap*, an additional amount of overlap given to columns temporally predicted in the previous time step.

Next, a *cell* (gray circles in Figure 23) is a representational unit belonging to exactly one column. Cell activity provides a current representation of the temporal context in which their column is currently active, if it is. Long-term information about context is encoded in CLA using a combination of *distal synapses* and *distal dendrite segments*. Each *distal synapse* is a synapse that has a cell as its source, and has, as its sink,

[10] The original CLA white paper refers to both "overlap score" and "boosted overlap score" as "overlap." Here I distinguish between the two.

a distal dendrite segment, which is connected to another cell, with the constraint that these two cells must be from different columns (Figure 23).

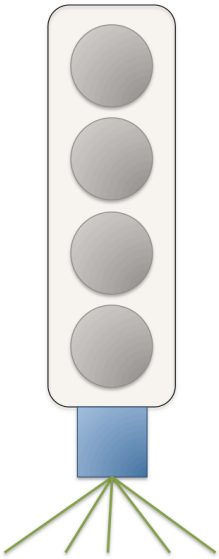


Figure 22. A single column having four cells. Each column has a single proximal dendrite segment (blue), which integrates the activity of several proximal synapses (green).

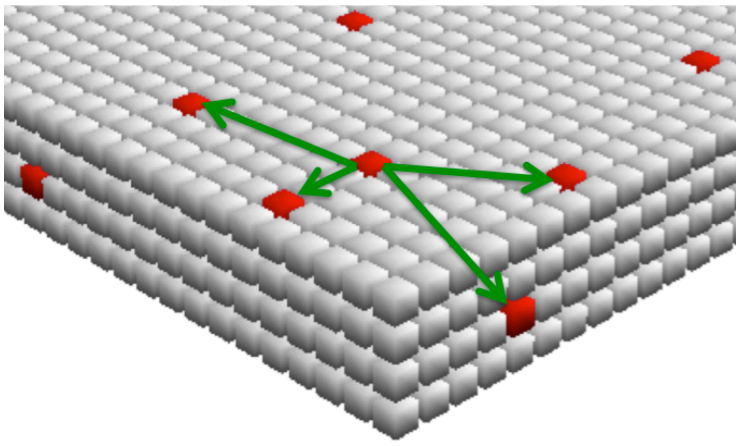


Figure 23. Distal synapse connection in a cortical region. Each distal synapse has a distal dendrite segment source that is associated with a particular cell. A distal synapse's sink is some cell in the neighborhood of the associated cell of its source.

Distal dendrite segments maintain a set of distal synapses, and integrate the synapses' activity. Distal dendrite segments also have an activation threshold, measured in the number of *active synapses* (explained shortly). If the number of active synapses is above this threshold the segment is considered active. Finally, each cell has several distal dendrite segments (Figure 24) whose activity, in part, determines the cell's state. Specifically, dendrite segments determine whether cell state is *predicted* to become active in the future. Additionally the cell state may be *active* or *inactive.*

Each synapse, proximal or distal, has a source, a sink, a binary weight, and a *permanence* (defined below). A *potential synapse* is one with a weight of 0, while a *connected synapse* is one with a weight of 1. A synapse is *active* if, and only if, its source is active *and* it is connected. Synapse weight is determined by *permanence*, a scalar attribute of each synapse modified by learning. If the permanence is above the *synapse connection threshold*, then the synapse is *connected* with a weight of 1. If the permanence

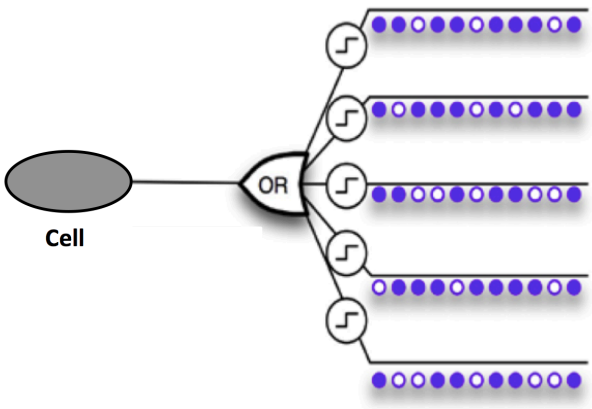


Figure 24. A single cell with five distal dendrite segments. Distal dendrite segments, depicted by the circles containing a step function, have several distal synapses, depicted as open and filled blue circles. Each of these circles represents a possible connection with a neighboring cell in the region. A cell becomes predictive whenever one or more of these segments become active, while a segment becomes active depending on sufficient synaptic activity (filled-in blue circles).

is below this threshold, but above 0, then the synapse is *potential* with weight 0. Finally, if a synapse's permanence drops to 0 or less, then it is removed. In review, the source of a *distal synapse* is some cell in the cortical region, and the sink is a nearby (relative to the source) distal dendrite segment. In contrast, the source of a *proximal synapse* is a bit in the cortical region's input, and the sink is the proximal dendrite segment of a column.

Cortical regions are designed such that they may be repeated hierarchically, without an intermediary between one region's output and the next higher region's input. It is also possible to have a tree-shaped hierarchy of regions, with multiple regions at same hierarchical level, each processing a portion of the network's input. In such a situation the next hierarchical level would pool together the output of these multiple regions. Here, I initially focus on a single 2D cortical region and a sensory input. In the PC-CLA chapter, I consider multiple hierarchical levels with each level consisting of a single region.

Cortical Region Process. In this implementation, each cortical region is driven by its own *cortical region process,* which runs a serial cycle updating the region's state, and performing learning. This cycle is depicted in Figure 25, and I first describe it at a high level just below.

The cortical region process can be roughly subdivided into two sub-processes, *spatial pooling* and *temporal pooling*, to reuse terminology from the original CLA description. Spatial pooling refers to the process of grouping similar inputs into the same (or nearly the same) sparse distributed set of active columns that represent the input. Spatial pooling approximately corresponds to Step 1 below in the detailed description. Temporal pooling (corresponding to Step 2) occurs sequentially after the spatial pooling

step. It takes the active columns representation produced by spatial pooling, and the current temporal context encoded by the cells predicted in the last cycle to become active this cycle, and produces the current active cell state. From the current active cell state the temporal pooler also produces a current predicted cell state, the context for the next cycle. Both the current active cell state and the current predicted cell state comprise the temporal pooler's output. The union of these two states represents at least two time steps, and should exhibit some temporal invariance, hence the name *temporal pooler*. With this high-level description in mind, I now summarize the main loop of the cortical region process (Figure 25) at an arbitrary cycle t as follows:

Step 1. Based on the current bottom-up Boolean input, y, compute the active columns of the cortical region, $L1$, for cycle t.

a) Perform process g taking in the bottom-up input, y, and the columns' proximal dendrites and associated proximal synapses, and outputting the columns' *overlap scores*.

b) For columns with a boosted *overlap score* greater than a threshold, perform a local k-winners-take-all procedure to determine the active columns, $L1$. The constraint, k, limits the number of possible active columns within a given area ensuring that the active columns are distributed.

Step 2. Compute the active cells at cycle t, $L2$, the current cells predicted to be active at *some* future cycle, $PL2_t$, and their union, U.

a) Based on the active columns, $L1$ and the currently predicted cells, $PL2_{t-1}$ (computed in Step 2b of cycle $t - 1$), compute the current active cells, $L2$.

b) Based on the active cells, $L2$, and the region's distal dendrites and synapses, perform process, f, producing the region's current predicted (for some future cycle) cells, $PL2_t$.

c) Compute the union, *U*, of the active cells, *L2*, and the current predicted cells, $PL2_t$.

Step 3. Perform the learning processes.

a) Perform spatial learning, updating the permanence of proximal synapses based on bottom-up prediction error. Also update each column's *boost* attribute based on its activity history.

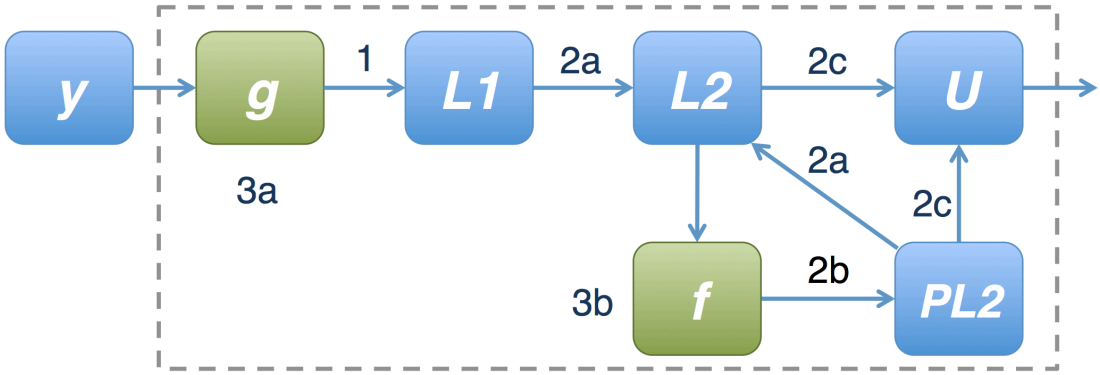


Figure 25. The cortical region process for a single hierarchical level with an input. All the components of single cortical region appear within the gray dashed box.

b) Perform temporal learning, which updates distal synapse permanence, and possibly adds new distal synapses. This learning is driven by both unpredicted columns and predicted columns that did not actually become active. I describe the details of these processes in the next section.

Learning. In this section, I give the additional details for the learning Step 3 of the cortical region process. In the original CLA, the spatial learning process iterates over each current active column, and updates the permanence values of the column's synapses based on the synapse's input. Synapses with active inputs have their permanence incremented, while those with inactive inputs have their permanence decremented. This has the effect of tuning columns to be more responsive to their current bottom-up inputs.

For boosting, there are two separate mechanisms to bias underrepresented columns. If a column's *active history*, the recent history of being an active column, is not a sufficient percentage of its neighbors' active history, its *boost* attribute is increased based on the difference. Increased *boost* makes it more likely for a column to win in the inhibition step, and become an active column. Define a column's *competition history* to be the recent history of whether its overall column activity was sufficient to enter the inhibition competition. If competition history is not a sufficient percentage of its neighbor's competition history, then the permanence of each of its proximal synapses is also increased or "boosted."

For the temporal pooler, learning updates distal dendrite segments. For computational reasons, here I introduce an attribute that is unique to distal dendrite segments called *prediction order*. The *prediction order* of a distal dendrite segment represents the number of cycles in the future for which the segment predicts when it is active. Initially, the *prediction order* of all distal dendrite segments is 1. Whenever the temporal learning process uses a segment in an attempt to add a temporal prediction with an order greater than 1, the segment's *prediction order* is changed to the new order. Given this, the temporal learning can be summarized in three types of update: 1) For each active column without any cells predicted to be active for the current time step, we wish to have, at least, a first-order temporal prediction. Based on the previously active cells, the distal dendrite segment that best predicted the column's activity is selected for an update later in that cycle. In this kind of segment update, the segment's synapses are positively updated, and new synapses, also predictive of the column's activity, are added to the segment. 2) For each active (predicting) distal dendrite segment (determined in step

3b of the cortical region process), a segment update data structure, recording information pertinent to a possible update, is stored for potential implementation until the cycle in which the prediction's validity can be verified, which is termed the segment's *verification time*. If this type of update is processed, new synapses are not added. 3) As with updates of the previous type (2), for each cell that is currently predicted by a distal dendrite segment, another distal dendrite segment update that predicts the cell's activity *one cycle earlier* is stored for a potential implementation. If performed, this type of update adds new synapses to bolster the new higher-order prediction.

Not all stored segment updates are actually performed, as we also wish to keep the distal synaptic connections, and the amount of predicted cells they produce, sparse. If a column is already well predicted by an existing first-order dendrite segment, it is not necessary, and likely detrimental, to update synapses so as to increase similar prediction. In order to enforce sparsity in the temporal pooler, one *learning cell* is always designated during each cycle for each active column. A cell is marked learning if it was just previously, and sufficiently, predicted by other learning cells or, otherwise, because it was the most strongly predicted by distal dendrite segments.

With all this in mind, I now discuss how segment updates are performed. Updates of type 1 are always performed in the cycle in which they are generated, and always positively update the segment's synapses. Stored updates of types 2 and 3 are processed at their *verification time* based on the observed state of the cell. If the cell is indeed active at the *verification time*, and the cell has been chosen to be the learning cell for its column during that cycle, then the update is performed positively. We want to positively reinforce correct predictions when they concern a learning cell. In this case, if the cell is

not learning, the stored segment update is not performed and is discarded (we want to keep learning minimal for each column). Finally, if the update involves a segment whose cell is inactive at the *verification time*, then the update negatively modifies the segment's synapses. This weakens a prediction that did not come true.

2D-CLA

The main difference between my 2D implementation of CLA and the original CLA model concerns the receptive fields of columns. The original CLA randomly chose the inputs from which to base its receptive field, from input bits near to the column, with closer bits being more likely to be chosen. 2D-CLA generates a deterministic receptive field based on a bivariate Gaussian distribution; however a smaller number of proximal synapses are randomly added to each column to prevent ties in column activity. I focus on 2D receptive fields and a 2D arrangement of the internal representational elements so that the algorithm is more applicable to "natural" data streams. There is evidence for retinotopic maps in the visual cortex involving the orderly mapping of receptive field position to coordinates in the brain (Swindale, 2008). More generally, neuroscience identifies *topographic maps* as neuronal connections arranged such that neighboring points in the periphery are represented by adjacent locations in the central nervous system. Such maps are found in the somatic sensory, visual, and motor systems. In other systems (e.g., the auditory and olfactory systems), there are also orderly representations of various stimulus attributes like frequency or receptor identity (Purves et al., 2001).

The next difference concerns the `getKthOverlappingColumn(Collection<Column>)` function. This function takes a collection of competing columns and a k value and returns the column with the k^{th}

highest activity score among the competitors or null if this is not possible. This function is used to implement part of the column inhibition step (Step 2c) of the cortical region process. This function was changed since I found that it was biased towards those columns out the outer edges of the cortical region's 2D array of columns. Without this correction, such outer columns are more likely than inner columns to be selected because their neighborhood is smaller. The modification offsets this bias by scaling *k* based on the number of columns that are passed to the method, and makes it harder for columns with small neighborhoods (those on the edge of the region) to win. This scaling factor is produced by dividing the number of columns by an estimate of the maximum number of competing neighbors a column could possibly have.

Initialization. The initialization of the 2D Cortical Learning Algorithms is as follows: based on user-specified parameters, the region's columns are generated having a 2D organization. The columns' proximal dendrite segments are initialized with a receptive field of proximal synapses modeled using a bivariate Gaussian distribution. Moreover, proximal synapses with randomly generated inputs are added to each column as well. The permanence of the proximal synapses is randomly initialized using a different univariate Gaussian distribution centered at the *synapse connection threshold*. Cells are given a fixed number of distal dendrite segments, which themselves, in turn, are initialized with some number of distal synapses chosen among local, neighboring cells. Distal synapse permanence is deliberately initialized to a value below the *synapse connection threshold* so that a few positive updates are required before a synapse can become connected. See Appendix B for the pseudocode details of the receptive field initialization for columns.

Parameters. The 2D Cortical Learning Algorithms has a significant number of parameters, some of which are introduced by this particular implementation. Their roles and values are discussed in detail in this section. To start, I designate the dimensionality of the input by *n.* Next, I define the *projection factor* parameter as the ratio of the number of columns in a cortical region to the number of bottom-up inputs to the region. As such the product of *n* and *projection factor* determines the total number of columns in the region.

For this implementation, the columns of a region have a 2D arrangement, and, accordingly, their receptive fields are modeled using a bivariate Gaussian distribution. That is to say, the input sources of the proximal synapses of each column are selected based on this distribution. More formally, I define the distribution over a 2D input space where a point in the space is denoted by (x, y). The distribution has equal variances in the x and y dimension, which I denote by, σ_{rf}, and I assume zero correlation between x and y. We want a function that outputs the number of proximal synapses that will be generated for a given input with position (x, y) as a function of the position's distance from the column at (x_c, y_c). Intuitively, this will produce a circular receptive field, with potentially more synapses near the center than on the periphery (depends on exact parameter values). To do this I take the bivariate Gaussian function just described, and add a multiplicative constant to obtain a function, f, which outputs the number of proximal synapses assigned to a given column, c, with position (x_c, y_c), as a function of the input position (x, y):

$$f(x,y) = \frac{s_{rf}}{2}\pi\sigma_{rf}^2 \exp\left(-\frac{(x-x_c)^2 + (y-y_c)^2}{2\sigma_{rf}^2}\right) \quad [7]$$

There are two parameters governing the size of this receptive field. The first, termed *receptive field sigma*, σ_{rf}, refers to the standard deviation for both dimensions (x

and y) of the bivariate Gaussian probability distribution. The second parameter, called *receptive field scaling*, s_{rf}, scales the possible output range of the bivariate Gaussian from [0,1) to [0, s_{rf}). The purpose of the scaling is so that we can assign a positive integer number of synapses to a position (x, y). For a given input location, $f(x, y)$ defines the number of proximal synapses that will be created, taking their source from that input location (Figure 26). In order to prevent ties in column activity, I assign a small number of proximal synapses with random input positions to each column. The parameter, *receptive field randomness*, specifies the number of random synapses as a ratio of the

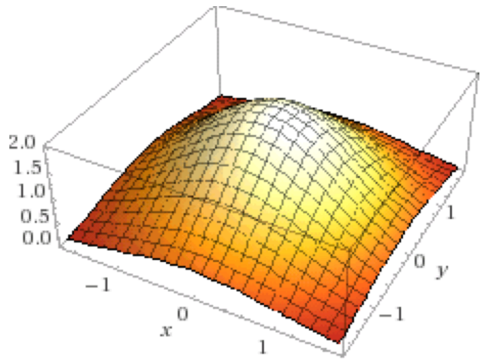


Figure 26. Rendering of the bivariate Gaussian receptive field model for a single column. Here the column is centered at (0, 0) in this example. The x- and y-axis represent the input space. For a given coordinate (x, y), this function outputs (vertical z-axis) the number of synapses (0 or more) to be generated for the column that will take their presynaptic input from coordinate (x, y) in the input space.

deterministic ones. A similar Gaussian function is used to generate the random input position for these synapses.

Given the *receptive field sigma* and *receptive field scaling*, it is possible to determine the maximum possible distance that a proximal synapse may be from its associated column. This maximum distance is defined to be the size of the column's *receptive field radius*, r_{rf}. I obtain an expression for this radius by taking Equation 7 and

setting the left hand side equal to ½, the lowest output value that still rounds to at least 1 synapse. Then, making the substitution $r_{rf}^2 = x^2 + y^2$ the *receptive field radius* is given by

$$r_{rf} = \sqrt{2\sigma_{rf}^2 \ln[\frac{s_{rf}}{\pi\sigma_{rf}^2}]} \quad [8]$$

Intuitively, twice the *receptive field radius* gives us an idea of the maximum possible distance between two inputs that feed into the same column. For this work, it will be assumed that r_{rf} is a parameter determined by the specific needs of the user. Then, given a user-specified radius, r', values for proximal synapse source sigma and proximal synapse scaling can be calculated. For instance, in this work they are calculated as follows:

$$\sigma_{rf} = r' \quad [9]$$

$$s_{rf} = \pi\sqrt{e}(r')^2 \quad [10]$$

At runtime, several additional parameters come into play. The *column competition threshold* parameter specifies a threshold on the overall column activity (Step 2b), which combines the bottom-up boosted *overlap score* and the *predicted column overlap*. Columns below this threshold are given an activity score of 0 for that cycle, and do not participate in the inter-column inhibition step. Next, the *local column activity* parameter determines how many columns can possibly win the inter-column inhibition within a given area (Step 2c). In detail, I define the inhibition neighborhood of the column by a circle with a radius, *inhibition radius*, the average of the distance from connected proximal synapses to their columns across all columns. Then, for each competing column, the column must have the *local-column-activity*th highest column activity score of all the columns in this neighborhood to become an active column.

The following parameters affect column boosting. *History size* defines the history length, in cycles, of each column's active history and competition history. The former records whether a column was an active column each cycle, while the latter records whether a column had a sufficient column activity score to compete in the inter-column inhibition step. Given these histories, the *minimum column activity* parameter refers to the minimum average activity (averaged over its active history) a column may have among its inhibition neighborhood before its *boost* value is increased. Recall that *boost* helps historically weaker columns become more likely to win. Thus, this parameter specifies the minimum average activity a column can have, as percentage of the neighbors with maximum average activity. Columns below this minimum are boosted. The *minimum column activity* parameter also determines the threshold at which insufficiently competitive columns have their proximal synapses positively updated or "boosted." Finally, the *boost function slope* controls the slope of a linear curve governing the amount of *boost* given to those columns that, recently, have been insufficiently active.

Several parameters are related to the cells of the cortical region. The *cells per column* parameter governs how many cells are generated for each column, *dendrites per cell*, how may distal dendrites segments feed into each cell, and *initial synapses per distal segment*, the initial number of synapses initially generated for each distal dendrite segment. Due to learning, distal dendrites may gain additional synapses (connected or potential) up to a maximum specified by *maximum new distal synapses per update*. Distal dendrite segments are judged active whenever they have at least *distal dendrite activation threshold* synapses active, and can be chosen to undergo learning if they have at least *segment learning threshold* synapses active. Finally, when a source of a new distal

synapse is chosen among neighboring cells, the source can be from a distance of at most *distal learning radius* away from the synapse's dendrite segment sink. Table 5 summarizes the parameters discussed in this section.

2D-CLA Testing

In this section, I describe several tests of the 2D-CLA implementation. Several earlier tests are focused on exploring several properties of the spatial pooling and temporal pooler sub-processes, as well as the effects of particular parameters on these processes. Many tests build upon the results of earlier ones. To my knowledge, these CLA properties have seen little or no formal testing.

Most tests use randomly generated binary vector patterns as input. A motivation for this choice of input is that the algorithm should be able to handle arbitrary input patterns; consequently one advantage of random patterns is that they introduce no bias to input patterns. However, a disadvantage is that random patterns may not introduce many patterns that are challenging to discriminate since random patterns are likely to be quite different from one another and easy to discriminate.

For some tests that compare two Boolean vector patterns, I will use the taxicab distance, d_1, (Krause, 1986), also known as Manhattan distance. This distance metric is defined for two vectors p, q, in an n-dimensional real vector space with a fixed Cartesian coordinate system, and can be defined by:

$$d_1(p,q) = \sum_{i=1}^{n} |p_i - q_i| \qquad [11]$$

For tests where I would like to evaluate the accuracy of retrieval (in this work's context accuracy of prediction), I use the *F-score* measure (van Rijsbergen, 1979). I consider the accuracy of both top-down and temporal predictions of cell activity with this

Table 5. The main parameters of the algorithm, their descriptions, and the default values used in this implementation.

Parameter	**Associated variable**	**Description**	**Default value**
Input size	n	The dimensionality of the input	256
Projection factor	p	Ratio of number of columns to the input size	4.0
Receptive field sigma	σ_{rf}	Sigma value for bivariate Gaussian modeling the receptive field of columns	4.0
Receptive field scaling	s_{rf}	Scaling factor for bivariate Gaussian modeling the receptive field of columns	83.0
Receptive field randomness	—	The number of randomly generated proximal synapses per column as a proportion of the deterministically generated ones.	0.05
Column competition threshold	—	Threshold on the bottom-up column activity determining the columns entering the inhibition step	0.02
Local column activity	*lca*	The maximum number of active columns allowed within a neighborhood around a competing column.	3.0
History size	—	The length of the history record of each column's activity.	1000
Minimum column activity	—	Minimum average activity a column may have, as a percentage of its neighbors' activity, before its overlap is boosted	0.01
Boost function slope	—	Slope of the linear function controlling *boost* updates.	0.05
Cells per column	*cells*	Number of unique cells in each column.	7
Dendrite segments per cell	*ds*	Number of unique distal dendrite segments associated with each cell.	5
Initial synapses per distal segment	—	Number of synapses initially generated per distal dendrite segment.	20
Maximum distal synapses added per segment update	—	Limitation on the number of new synapses added in a single segment update.	20

Table 5. (Continued).

Parameter	Associated variable	Description	Default value
Maximum synapses per distal segment	s_d	Absolute limit on number of synapses per distal dendrite segment.	32
Distal dendrite activation threshold	—	Number of active distal synapses for a distal dendrite segment to be active, and associated cell predictive.	1
Segment learning threshold	—	Number of active distal synapses required for a distal dendrite segment to be eligible for learning.	1
Distal learning radius	—	Radius of a circle about the column where new distal synapses are being adding. Only cells within this circle selected as such a synapse's source.	7.0

measure, as well as the accuracy in reproducing the active columns representation of a previously encountered input. Next, I introduce two prerequisite definitions in defining the F-score. Given an observation, representing the actuality, and a prediction from a classifier, denote the number of *true positives* (correct predictions) by *tp*, the number of *false positives* (incorrect predictions) by *fp*, and the number of *false negatives* (missing predictions) by *fn*. Given these, the *precision* can be defined as: $precision = tp / (tp + fp)$ and the *recall*: $recall = tp / (tp + fn)$. For either measure, if the denominator evaluates to zero, then the measure is defined to have value one, and, consequently, both measures can take on values in the closed interval [0, 1].

Precision and recall are particularly useful in evaluating the performance of classification algorithms involving skewed classes, a situation in which the likelihood of a particular class occurring greatly outstrips the others (Ng, 2013). In the context of predicting sparse Boolean vector patterns, I identify two such classes for pattern bits: 1)

true is the positive class and 2) a false is the negative class. With sparse patterns, the classes are then skewed with only 2% of the bits typically being of the positive class. In such situations, using the raw percentage of correct predictions to measure accuracy has a drawback, e.g., a naive classifier never predicting any positive results would typically obtain a high percentage of correct predictions, in this case, 98% correct. However, assuming that the patterns had a nonzero number of true bits, the same classifier would obtain the worst possible score in terms of recall: zero.

Typically, in statistics and machine learning, the precision and recall scores are combined into a single scalar measure of accuracy, the F-score, denoted F_β, and defined, for a positive real, β, as:

$$F_\beta = \frac{(1+\beta^2)\cdot precision \cdot recall}{\beta^2 \cdot precision + recall} \quad [12]$$

F_β measures the effectiveness of a classification (here, of true or false), given that we attach β times as much importance to recall as precision (van Rijsbergen, 1979). For example, F_1 weights the importance of precision and recall equally, and is a common single-valued accuracy metric. In the context of PC-CLA, we require a good recall score first and foremost; however, if we are evaluating temporal predictions, there is a tolerance for a less-than-perfect precision score, which I discuss next.

Precision measures the accuracy of predictions — low precision means positive predictions are incorrect overall, while high precision means positive predictions are mainly correct. For temporal predictions in PC-CLA, I suggest tolerating some inaccuracy in positive predictions. Such a tolerance for false positives can be explained by appealing to overlapping sequences, those sequences sharing common co-occurrence patterns for one or more instants that later diverge. Having learned overlapping sequences

in a cortical region can lead to situations where a particular prediction may be a false positive in some contexts, but a true positive in others. In general, the CLA can handle multiple predictions at the same time because each temporal prediction is sparse (typically less than 1%), thus combining predictions of multiple possible future patterns in a single set of predicted cells is unlikely to cause interference or break the sparsity of the representation. Furthermore, if those synapses producing false positive predictions were removed or considerably weakened, learned patterns could be lost without any good reason to believe that such patterns cease to exist.

Can we quantify a tolerance for false positives in terms of precision? The original CLA theory (Hawkins et al., 2011, p. 29) suggests that completing even 10% of the active columns representation of a pattern is sufficient to have high certainty that the full pattern is occurring. One main reason predictions are desirable is to maintain representations in the face of noisy or missing inputs. For instance, suppose there is a well-learned pattern A-B-C, and that A is input to a cortical region and, at the next moment, B is missing or severely corrupted. We'd like to predict B from A sufficiently well so that a representation corresponding to A-B-C is maintained; specifically we would like to correctly predict at least 10% of the active columns of B. Given that the active columns representations of input patterns is typically near to 2% of the total columns, it then follows that the number of active cells would be at most 2%, and is typically less (since temporal predictions are kept to a minimum). Since the active cells are so sparse, the predicted cells are typically sparse as well (< 2%). So we have an upper bound on the total number of predicted cells (2%), and a lower bound on true positive predictions (10%), which implies an upper bound on false positives of 90%. These

percentages correspond to a lower bound on precision: $0.1/(0.1+0.9)=0.1$. Given this, and the motivation for good recall, I suggest using a *beta* value of 10 for the F-score (F_{10}), which weights recall 10 times as important as precision, to evaluate temporal predictions in CLA. In contrast, later tests of multiple hierarchical levels suggest false positives are undesirable for *top-down* predictions, and, consequently, I will use the F_1 score to evaluate these.

Spatial Pooler. The central objective for the spatial pooling operation (see Step 1 of Cortical Region Process section) is the production of a sparse, distributed set of active columns. As mentioned previously, it has been suggested (Kanerva, 1988; Snaider, 2012) that sparse high-dimensional vectors have desirable uniqueness and robustness properties, e.g., binary vectors with 2% active bits in 1,000–10,000 dimensional spaces. Thus, a sparsity criterion of 2% column activity was adopted as the target activity for the spatial pooler's output.

The importance of sparse column activity compels the question: Which factors determine the amount of column activity produced by the spatial pooling operation? Figure 27 summarizes the causal relationships between the algorithm's various parameters and the amount of column activity. Starting at the bottom of the figure, we have the *inhibition radius* quantity, the average of the distance from connected proximal synapses to their columns across all columns. In this implementation, *inhibition radius* is a function of the parameters: *receptive field radius* and *projection factor*. Intuitively, a large *receptive field radius* generates columns having synapses farther from the column's center. This creates columns whose activity represents the presence of active inputs across a larger, more disparate range of inputs. Recall that the *projection factor* is defined

as the ratio of the number of columns in a cortical region to the number of bottom-up inputs to the region. The *projection factor* should be chosen to project the input into a sufficiently high-dimensional column representation space, e.g., 1024–2048. The values of these two quantities are assumed to be chosen based on the needs of a specific application of the algorithm.

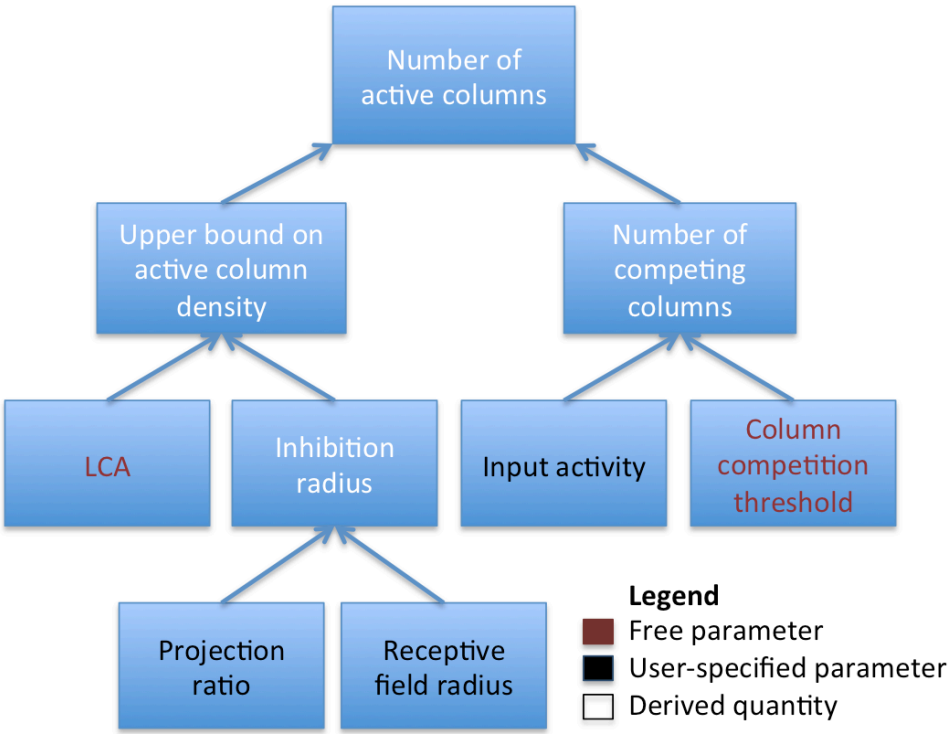


Figure 27. The factors affecting the number of active columns produced by the spatial pooler and the causal relationships between these quantities.

Moving upwards in the figure, we have both the *local column activity* (LCA) free parameter and the *inhibition radius* determining an upper bound on the density of active columns in a given neighborhood of columns. Choosing a good value for the LCA parameter is the subject of the second test below. Switching over to the right side of the diagrammed hierarchy, we have the number of competing columns being a function of

the amount of *input activity* and the free parameter, *column competition threshold*. I define *input activity* as the ratio of true bits to total bits in a Boolean input. Finding a good value for *column competition threshold* is the subject of the first test below. To wrap up the diagram, the inter-column inhibition step takes the active column density bound from the left side and a set of competing columns from the right, and outputs a set of active columns.

The Cortical Learning Algorithms are replete with parameters. To reduce redundancy, Table 6 gives the default parameter values used in each of the tests below unless specifically mentioned otherwise.

Table 6. The default cortical region parameters used in the following tests unless specifically noted otherwise.

Quantity name	**Value(s) used**
Input signal dimensionality	256
Projection factor	4.0
Receptive field sigma	4.0
Receptive field scaling	83.0
Receptive field randomness	0.05
Column competition threshold	0.01
Local column activity	3.5
Minimum column activity	0.01 (1%)
Developmental period (cortical region process cycles)	350 (or 100 consecutive stable cycles)

Sparsity Robustness Test. In the original CLA whitepaper, the authors suggested that HTM cortical regions are able to convert distributed representations into sparse distributed representations of active columns (Hawkins et al., 2011, pp. 11–12), and that the percentage of active bits in the input might vary significantly, yet the spatial pooler might produce a sparse set of active columns. Motivated by these ideas, this test was

developed to explore the robustness of the spatial pooling operation to varying degrees of true bits in Boolean inputs, which I refer to as *input activities*. For a range of *input activities*, a series of trials were performed to ascertain the average amount of column activity produced by the spatial pooler. In each trial, the following steps were performed: 1) a single cortical region was created performing the standard random initialization of its columns' proximal synapse weights, 2) an input with a particular percentage of true bits was randomly generated, 3) the spatial pooler was repeatedly run on this input for a developmental period of 350 cycles, during which the proximal synapses adjust to the pattern, and a stable active columns representation of the input forms, and lastly 4) the percentage of columns that were active at the end of the developmental period was recorded.

Recall that the parameter *column competition threshold* specifies the amount of overall activity that columns must have before they can enter the inter-column inhibition step. As such, this parameter affects sparsity robustness. Additional tests were run varying the value of *column competition threshold*. Table 7 gives the main parameters and their specific values for this test.

A plot of the results of the Sparsity Robustness Test is shown in Figure 28. The sparseness of the active columns representation, in terms of the percentage of total columns active, remains reasonably close to 2%, gradually increasing, until about 80% *input activity*. This demonstrates that the spatial pooling algorithm is robust in the production of sparse representation even to more "dense" input having higher *input activity*. The data also suggest that the effect of the *column competition threshold* is to

considerably decrease column activity, but only for lower amounts of *input activity*. For example, a threshold of 0.2 showed significant difference from the other two threshold

Table 7. The chief parameters and their values in the Sparsity Robustness Test.

Parameter name	**Value(s) used**
Input activity (%)	1–100, steps of 5
Column competition threshold	0.00–0.30, steps of 0.1
Trials per test	200
Local column activity	5.0

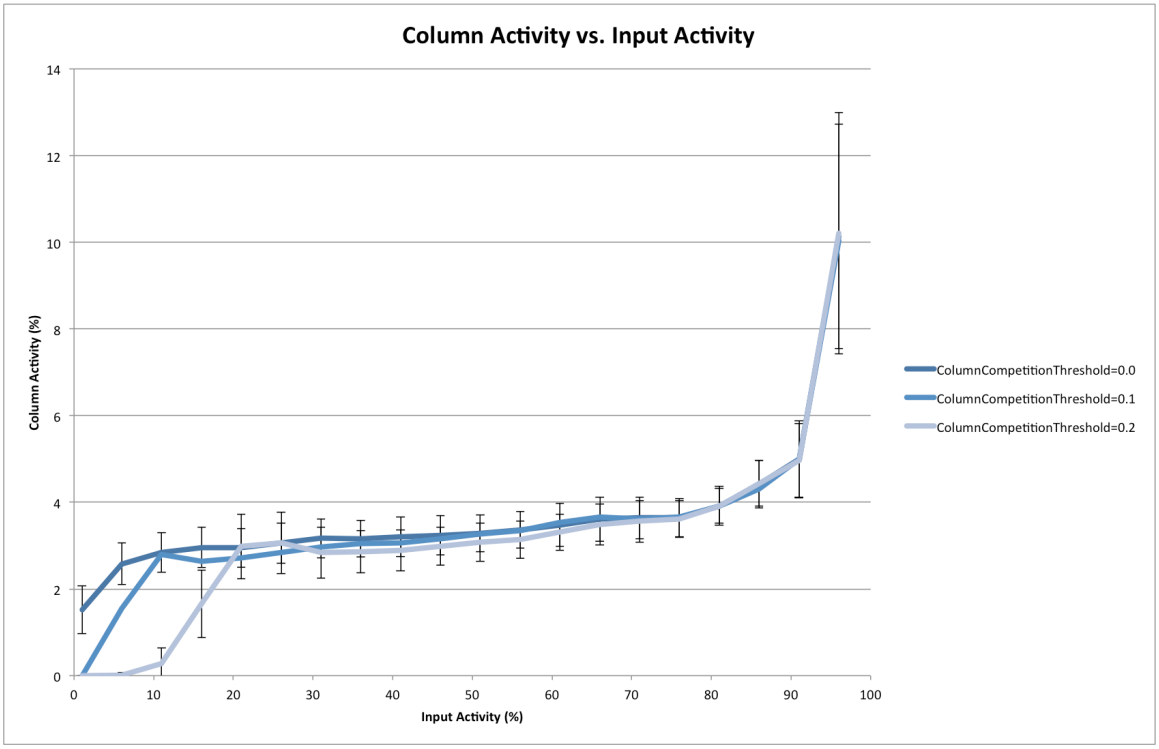


Figure 28. The results of the Sparsity Robustness Test. A plot of the average percentage of columns found active, after a developmental period, as a function of *input activity* percentage. The legend codes for difference values of the *column competition threshold* parameter.

conditions up to about 16% activity. This suggests keeping the threshold low (<= 0.1) so as to maintain sparsity for inputs with lower *input activity* (1–20%). A threshold of 0.0 is

discouraged in situations where noise in the input could excite at least one synapse in *every* column leading to a situation where *every* column must compete in the inhibition step (Ahmad, Personal Communication).

LCA Parameter Optimization Test. As discussed above, the *local column activity* (LCA) parameter affects column activity by determining the maximum number of columns that can possibly win the inter-column inhibition step within a given circular area with radius, *inhibition radius*. What value should this critical parameter take, so that spatial pooling produces approximately 2% column activity? This test seeks the answer. Based on the previous test, we know that the spatial pooler exhibits good sparsity robustness when the *column competition threshold* is low. So, for this test, I fixed this free parameter to 0.01. I also continued to fix the *projection factor* to 4.0 to have at least 1000 columns representing the data, since I am focusing on inputs of 256 dimensions. Given these assumptions, I seek a method of obtaining an LCA value producing about 2% *column activity* for a range of receptive field radii and a range of *input activity*. Thus, I varied the *input activity*, i, and the *receptive field radius*, r_{rf}. For each pair of these values (i, r_{rf}), I performed a series of trials, each using a randomly generated input. In each trial an input with activity i was randomly generated, and input to a cortical region with columns having a *receptive field radius* of r_{rf}. Then, the spatial pooling operation was performed on the input for a developmental period, at the end of which the percentage of active columns was recorded. After all of these trials were performed, the average activity percentage was compared with the 2% target: If it was sufficiently close to the 2% target, the current LCA value was recorded for condition (i, r_{rf}), and the process advanced to the next condition pair. Otherwise, LCA was adjusted, and the

process was repeated until a sufficiently close result was found. Table 8 summarizes the various experimental parameters of the test.

Table 8. Parameters used in the LCA Parameter Optimization Test.

Parameter name	**Value(s) used**
Input activity (%)	0.02–0.22, steps of 0.1
Receptive field radius	3–7, steps of 1
Random inputs per trial	60
Sufficient average column activity (%)	2 ± 0.3

The results of this effort are shown in Figure 29. The data suggest a linear relationship between *receptive field radius* and optimal LCA value, regardless of *input activity*. The *input activity* also has an effect, though it is apparently nonlinear, which makes choosing the LCA parameter difficult if the *input activity* varies considerably. If *input activity* can be stably constrained, then linear regression can be used to determine a function that outputs a good LCA value given a *receptive field radius*. For this reason, in the subsequent tests, I typically fix *input activity* at 2%, or employ a range of *input activities* close to 2%. Then, for a fixed *input activity*, I perform a linear fit on the data points shown in Figure 29 to obtain a function outputting a LCA parameter value as a function of the *receptive field radius*. If *input activity* is not stable, then one must dynamically update LCA based on the changing *input activity* as the algorithm runs. This could perhaps be achieved by using multiple regression to produce an LCA-outputting function with two inputs, *input activity* and receptive field.

Note in Figure 29 that I have not shown results for receptive field radii less than 3.0. For fixed *input activity* these curves are still roughly linear in this range of radii, although less so. The issue for small radii is that rounding effects occur since the

continuous, Gaussian-based receptive field model must be actualized using a set of discrete input locations. If a *receptive field radius* of less than 3.0 is desired, it may be necessary to optimize the LCA for the particular radius and/or *input activity*.

Noise Robustness Test. Recall that the spatial pooling operation attempts to produce a sparse distributed set of active columns representing the current bottom-up input. One purported benefit of sparse distributed representation is noise robustness. To determine *how* robust 2D-CLA's active columns representations are to noise, I tested the effect of varying amounts of noise (0% to 50%), added to input patterns, on the inputs' resulting sets of active columns, that is, its sparse distributed representation. I also varied the *input activity*, the percentage of bits in the input patterns that were true, from 1% to 5%. I used this range since the Sparsity Robustness Test had already suggested low *input activity* to be desirable.

Concretely, for each *input activity* condition, *i*, a *single* Boolean input, with 256 dimensions, was generated with *i* percent of its bits randomly set to true. Next, in a developmental period, the input was shown to a cortical region performing only the spatial pooling operation for 250 cycles. At the end of this period, the final set of active columns was recorded as the sparse distributed representation of that single input.

Next, for a range of noise conditions, the original input pattern was corrupted by an amount of noise. Noise was added uniformly to true bits and false bits alike, turning the former false and the latter true. For a particular *input activity* condition and noise condition, 500 trials were performed. In each, a noisy version of the single original input was generated, and was then exposed to the same cortical region for a developmental

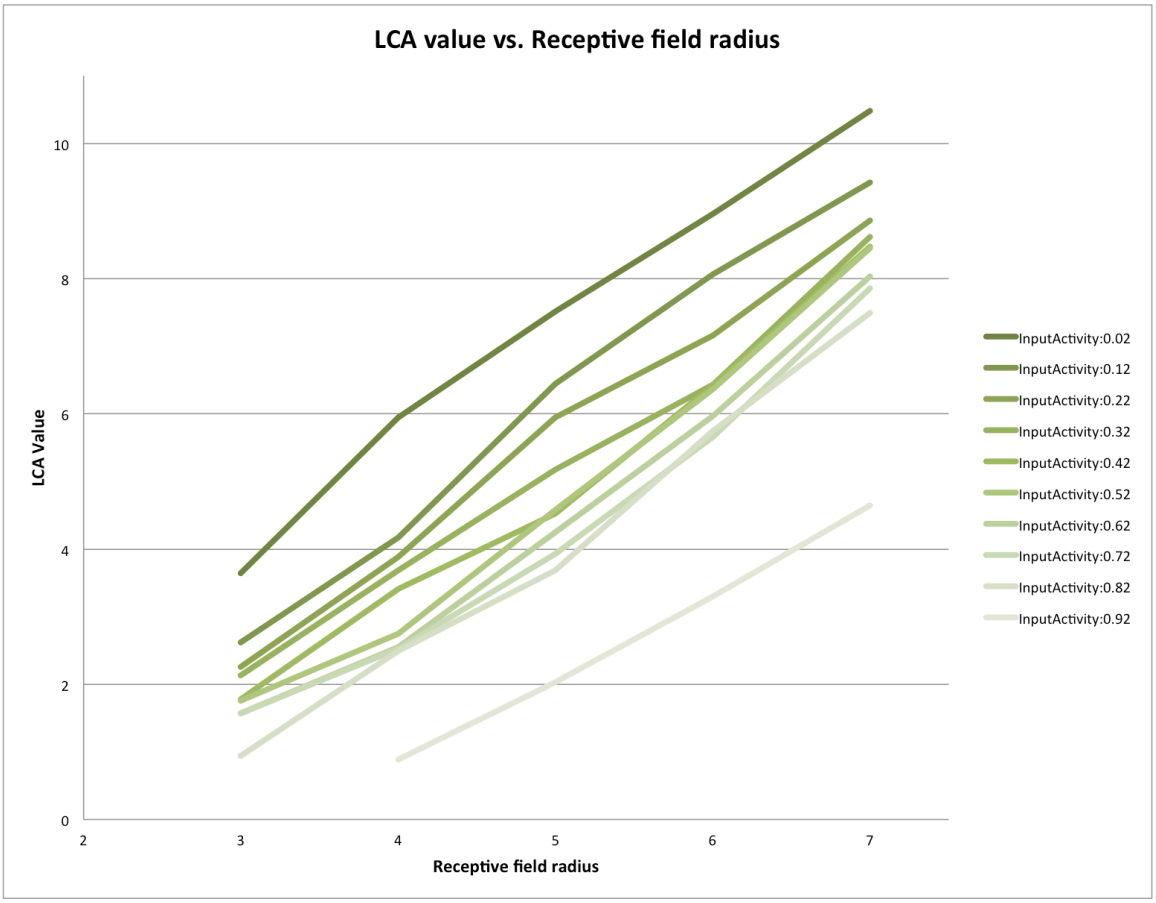


Figure 29. The results of the LCA Parameter Optimization Test. The LCA values producing near 2% column activity on average as a function of the column's *receptive field radius*. The legend additionally shows the range of *input activity* conditions explored.

period. At the end of this period the "noisy" set of active columns was compared with the single original set for that *input activity* condition.

To perform this comparison, I computed the normalized taxicab distance between the original set and a noisy set. Here, this distance is the total number of errors in the active columns representation, both false positives and false negatives, divided by total number of columns in the region. For a given *input activity* and noise amount, I averaged the scores across the 500 trials.

Figure 30 summarizes the results of this test. Increased amounts of noise (depicted in the legend) significantly increased the distance between an original active columns representation and noisy active columns representations. The data also suggest a benefit in keeping the *input activity* low; namely, that lower *input activities* have better noise robustness than higher *input activities* (e.g., the 50% noise curve changes signifcantly over the range of *input activities*).

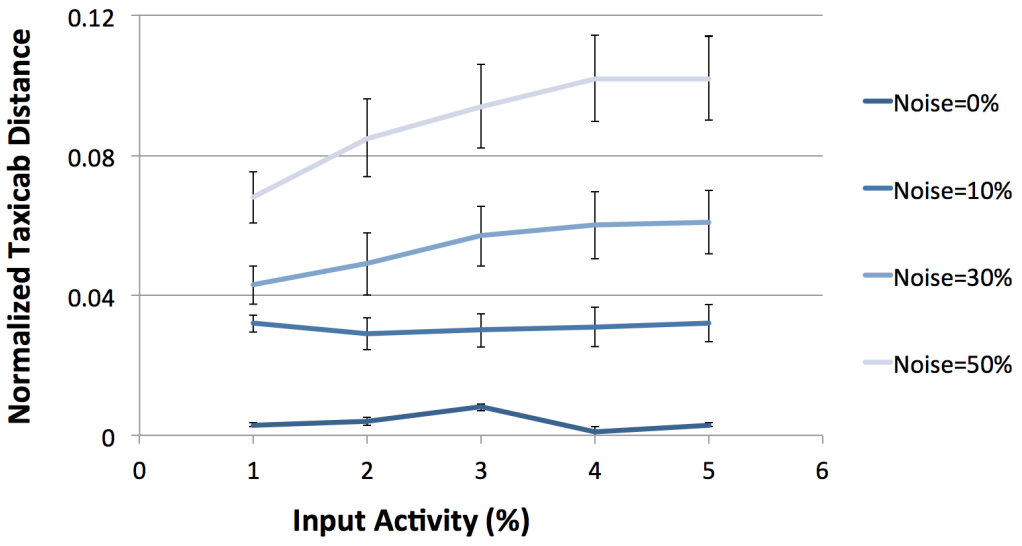


Figure 30. The results of the Noise Robustness Test.

Representation Distribution Test. So far we have looked at the *sparsity* of the sparse distributed representations produced by the spatial pooler, and their noise robustness. What about the degree of *distribution* of column usage in these representations? To this end, the spatial pooling operation includes a "boosting" component designed to encourage equal usage of columns in active columns representations over time. If active columns representations are inadequately distributed across all columns, then the algorithm is not taking advantage of the high dimensional column space and the benefits it confers. This test, then, explores the evenness of column usage in the active columns representations produced by the spatial pooler.

In each condition of this test, 10,000 inputs having 256 dimensions and a particular *input activity* were randomly generated. Each such input was shown to a cortical region running the spatial pooler for a developmental period of 200 cycles. After this period, I recorded the columns used in the active columns representation of the input. After all trials were performed, the average, and then the standard deviation, of each column's usage, across all trials, was computed. To summarize these usage standard deviations, they themselves were averaged to produce a singular measure.

Ideally, if the active columns representations are well distributed, then the variance in column usage should be high, approaching the variance of randomly chosen sets of columns; otherwise, the variance should be low. The aforementioned process was repeated for a range of *input activity* levels, and for a range of values of the parameter, *minimum column activity*. This parameter controls a threshold determining whether individual columns will be boosted based on their activity history. Another boosting parameter, *boost function slope*, governs the *magnitude* of the change to a column's *boost*

attribute when boosting is performed. Recall, the *boost* attribute weights competing columns in the inter-column inhibition step (Step 1b). For this test, I chose a fixed *boost function slope* that I found gave high variance scores.

To give the results of the test better context, I have also added a baseline data series. This baseline estimates the average variance of column usage if the distribution of column usage is (nearly) completely random. The baseline was obtained by first randomly generating, for each *input activity* condition, 10,000 Boolean patterns having the same dimensionality as the number of columns (1024) in the test's cortical region and having the particular *input activity*. Then, for each *input activity* condition, the average variance in the usage of each bit (representing a column) was computed across the 10,000 patterns.

The results of this test are shown in Figure 31. The data indicate that variance in column usage increases with *input activity*, which suggests keeping *input activity* low. While, this cannot be controlled by a parameter, it is possible to preprocess inputs to reduce *input activity*, e.g., performing subsampling or a predictive coding filter (Huang & Rao, 2011). All data series showed less variance on average than the baseline, implying that the column usage distribution was less than optimal. Moreover, it appears that this difference increased for higher *input activity* conditions. For the *input activities* depicted in Figure 31, the variance in column usage was not significantly affected by different values of the boosting parameter, *minimum column activity*. I did find greater differences between the boosting parameter conditions for high *input activity* (> 40%). If dealing with high *input activities*, this could be taken as suggestiing that this parameter be set very high. However, there is a drawback in doing so, which is the topic of the next test.

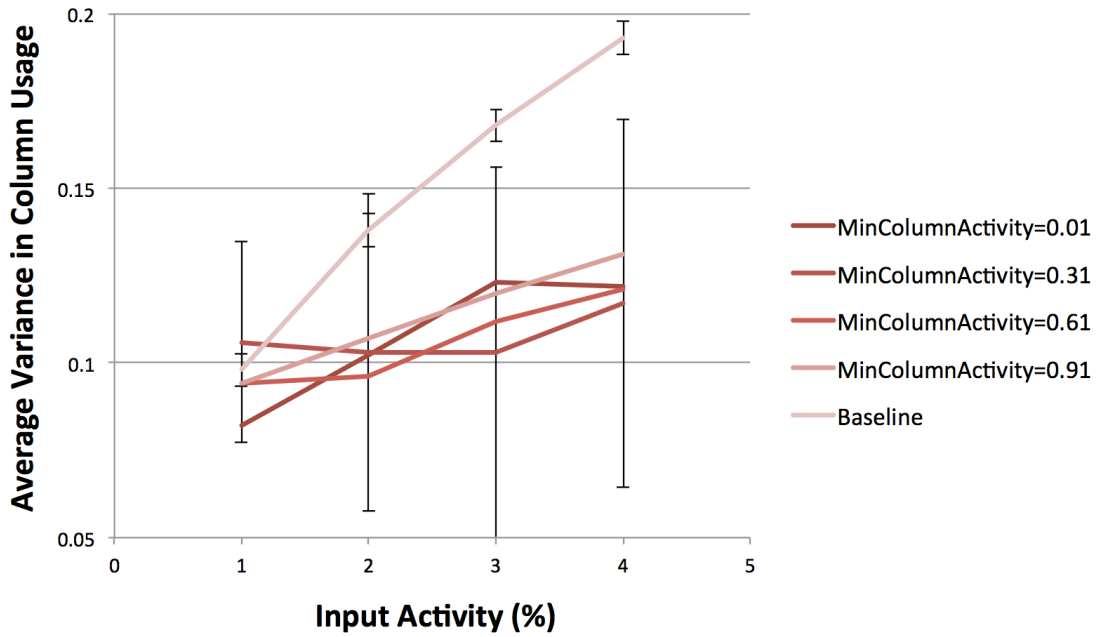


Figure 31. The results of the Representation Distribution Test. The graph depicts the average variance in column usage versus *input activity*. The legend illustrates different values of the boosting parameter, *minimum column activity*.

Representation Stability Test. The boosting component of the spatial pooler was motivated by the need for adequate distribution in column usage in the active columns representation of input. Recall that with boosting, columns recently having less activity (either in their competition score or winning the inter-column inhibition) are "boosted" or biased. In effect, little-used columns become more likely to be part of an active columns representation. A drawback of this boosting adjustment is that if it is implemented too strongly, then the active columns representation loses stability. For example, the representation of the exact same input might change with each presentation to the cortical region.

In this test, I explored this "representation stability" phenomenon. A series of trials were run using inputs with 2% of the bits set true. Such sparse inputs were chosen for earlier conceptual reasons and, additionally, some tests (discussed later) suggest benefits in using low *input activity* patterns. Another reason to try sparse inputs is that

cortical regions produce output representations with 2% sparsity (due to the sparsity of the active columns representation), and send their output to higher regions; thus most higher regions can expect to receive input with about 2% active bits. In each of 15,000 trials, a single input was generated and shown to a cortical region running the spatial pooler for 50 processing cycles. After each cycle, the difference between the current active columns set and the previous active columns set was calculated and recorded. Such trials were also run for different values of the boosting parameter, *minimum column*

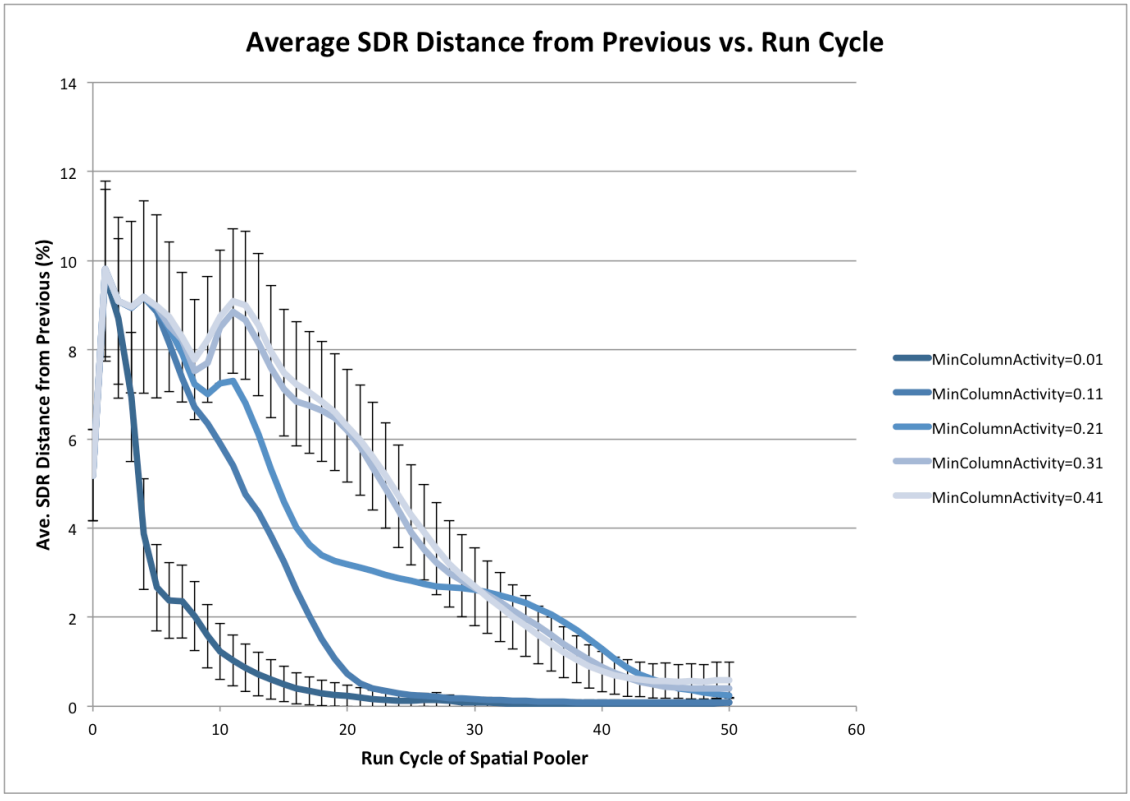


Figure 32. The results of the Representation Stability Test. This plot of the results of the Representation Stability Test shows the average distance in the current active columns representation from the previous representation versus the run cycle. The distance is the normalized taxicab distance from the previous cycle. Higher values of the *minimum column activity* parameter require more cycles for representational stability.

activity, which can take on a value from 0 to 1. The average results are shown in Figure 32 with their standard deviations for two of the parameter conditions.

Regardless of the *minimum column activity* parameter value, the active columns representation stabilized (< 1% change) after about 45 cycles of presentation. However, stability occurs much more quickly for a low *minimum column activity*, e.g., a value of 0.01 stabilizes in less than 20 cycles. This suggests that there is a drawback to setting *minimum column activity* high, and that it should not be set too high. Recall that the previous test suggests that *minimum column activity* should not be set too *low* since it increases the variance in column usage. In order to deal with this, I suggest keeping *input activity* low, e.g.,2%, which also has the effect of decreasing column usage variance. Then, *minimum column activity* can be kept at a low value, which will also maintain representation stability. An example of preprocessing that would reduce *input activity* is predictive coding filtering techniques (not to be confused with hierarchical predictive coding) (Huang & Rao, 2011). Such filters compute local prediction error, e.g., determining if a pixel's value is predictable based on its neighbors, which can have the effect of reducing the *input activity*, especially for signals with repetitive, redundant, portions.

Co-Occurrence Memory Degradation Test. The spatial pooler's learning process adjusts the permanence of the proximal synapses of a region's columns. These adjustments leave an imprint corresponding to a given input pattern, and can be viewed as memory for co-occurrence patterns. What happens when a cortical region receives a large number of different co-occurrence patterns? For instance, does the active columns representation of an input change because of all of the adjustments to the region's

proximal synapses? Surely, the more patterns that are "stored" the more learned patterns will be corrupted or degraded.

This experiment quantifies answers to these questions by storing an increasing number of co-occurrence patterns with 529 bits, 2% of which were true, into a single cortical region. After each set of patterns has been stored, the test then determines the degree to which each pattern's original active columns representation changes. Concretely, for a given number of patterns, p, a single cortical region is generated. Next, p patterns are randomly generated after which each is exposed to the cortical region for a developmental period of 200 cycles. At the end of this period, the active columns representation of the pattern is recorded. After all p patterns have seen development, and their active columns representation recorded, then each pattern is shown to the region for a single cycle, and the resulting "degraded" active columns representation is recorded and compared with the pattern's original active columns representation. The accuracy in pattern retrieval is computed in terms of the normalized taxicab distance and F_1 score. Figure 33 shows a plot of the average normalized taxicab distance, as a function of the number of patterns stored in a cortical region. The data indicate that there is not an increase in the taxicab distance with an increasing number of stored patterns. There is actually a slight decrease in the distance as more patterns are stored.

A similar pattern across the number-of-patterns condition was found using the F_1 score (Figure 34). That is to say the accuracy of retrieval seemed to improve slightly with larger numbers of stored patterns. In order to verify that there was good uniformity in column usage across the large number of active columns representations, I also computed the average variance in column usage (as defined in the previous Representation

Distribution Test) across both the original representations and the degraded ones as well (Figure 35). It appears that the average variance in column usage does not decrease due to a large number of stored patterns; in fact there seemed to be a general increase (an improvement) in column usage uniformity among the "degraded" patterns. For some context, randomly generated patterns of 2% true bits had an average variance of 0.139. This increase in usage variance alleviates concerns about uniformity in column usage.

Conclusions

The HTM Cortical Learning Algorithms is a recent approach to general pattern recognition inspired by the organization of the cerebral cortex. While the HTM theory has enjoyed previous published work by its creators, the CLA has not. Here I presented an implementation of the HTM Cortical Learning Algorithms, called 2D-CLA, which modifies the receptive fields of the algorithm to have limited 2D receptive fields. The

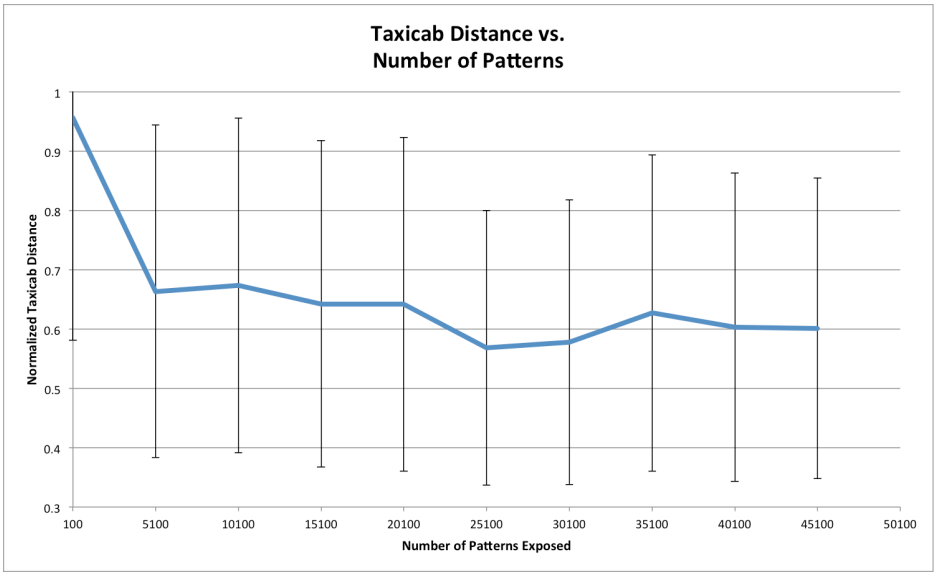


Figure 33. The average normalized taxicab distance of retrieved patterns as a function of the total number of patterns exposed to a cortical region. A higher distance implies that, after all patterns were exposed, the retrieved active columns representation was greatly different from its original representation.

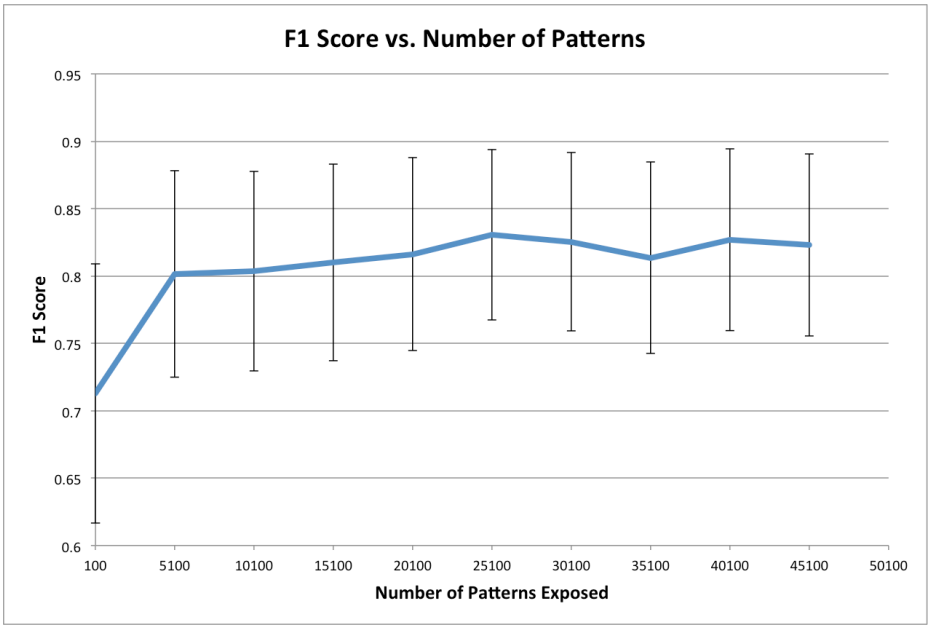


Figure 34. The average F_1 score for active column pattern retrieval as a function of the total number of patterns exposed to a cortical region. A higher score implies better retrieval.

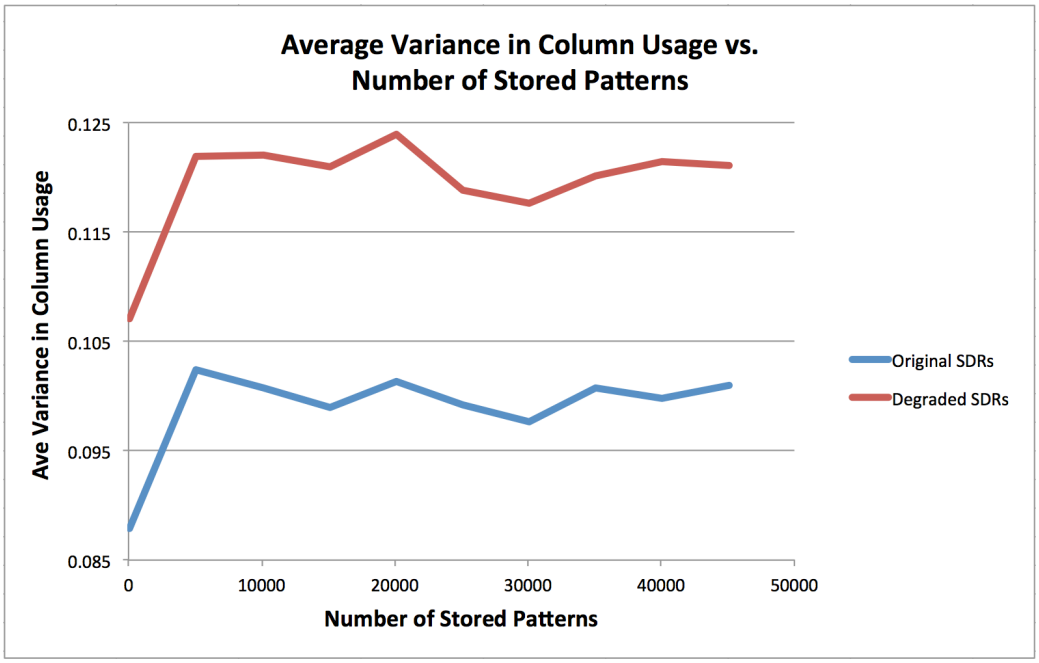


Figure 35. A comparison of the average variance in column usage among the original active columns representations (blue) and the degraded active columns representations.

tests in this chapter explore some of the basic properties of the spatial pooling portion of 2D-CLA on artificial 2D patterns. In several tests, I evaluated the sparse distributed output of spatial pooling; namely, the active columns representation.

The Sparsity Robustness Test demonstrated the robustness of the active column representation's *sparsity* for various *input activities*. The spatial pooler does well in producing sparse representations for inputs with about 80% or less true bits. The results also suggest keeping the *column competition threshold* low to maintain good sparsity for inputs with lower *input activity* (1–20%).

In the LCA Parameter Optimization Test I analyzed the problem of choosing the *local column activity* parameter when using the proposed 2D receptive fields. The data suggest linear relationships between *receptive field radius* and "optimal" LCA value, for a range of *input activities*. The *input activity* has a nonlinear effect on "optimal" LCA. If *input activity* can be stably maintained, then linear regression can produce a function to obtain a "good" LCA value given a particular *receptive field radius*.

The Noise Robustness Test demonstrated noise robustness for the sparse distributed representations of "learned" coincidence patterns. The data suggest better noise robustness for inputs with lower *input activities*, and this is especially true for higher amounts of added noise.

In the Representation Distribution Test I explored how well the spatial pooling algorithm makes evenly distributed use of a cortical region's columns. The results indicate that variance in column usage (high being desirable) decreases from baseline with increased *input activity*, suggesting keeping *input activity* low. In turn, this suggests preprocessing inputs to reduce *input activity* if possible. For sparse inputs (near 2%), the

effect of the boosting parameter, *minimum column activity*, on the variance in column usage was insignificant; however, for inputs with high activity rates this parameter has a greater effect.

In the Representation Stability Test I explore the stability of the active columns representation when "boosting" occurs. Boosting attempts to increase the evenness of column usage. On average, the active columns representation stabilized after about 45 cycles of input presentation with quicker stability for low values of the boosting parameter, *minimum column activity*. This suggests that there is a drawback to setting *minimum column activity* high, i.e., it will take a long time to obtain a stable representation of an input pattern. I suggest keeping *input activity* low, e.g.,2%, which also has the effect of increasing the evenness of column usage. Then the boosting parameter, *minimum column activity* can be kept at a low value, which will also maintain representation stability.

Finally, in the Co-Occurrence Memory Degradation Test I looked at the effect of "storing" increasing numbers of coincidence patterns on their active columns representations. I measured the difference between original representations of coincidence patterns and the representations after many such patterns had been stored in the memory. The data indicate a slight decrease in the taxicab distance with an increasing number of stored patterns. The accuracy of representation retrieval, measure by the F_1 score, appeared to improve slightly with larger numbers of stored patterns. Finally the average variance in column usage does not decrease due to a large number of stored patterns; in fact, there seemed to be a general improvement in column usage variance among the "degraded" patterns.

In this section, I have studied a 2D receptive field extension to the HTM Cortical Learning Algorithms (2D-CLA). Here, I explored the effects of the algorithm's various parameters, a little-studied topic. These test results provide new information about the performance of algorithm with respect to several parameters of the CLA's spatial pooler as well as parameters introduced with the 2D-CLA extension. A recurring lesson from these tests is that the algorithm performs best not only when it produces sparse distributed representations of inputs, but when it receives sparse distributed representations in the first place. While it takes some effort to configure, the spatial pooling algorithm showed the ability to sparsify general Boolean inputs, produce noise robust representations, and have an adequate storage capacity for co-occurrence memories.

5 Cortical Learning Algorithms with Predictive Coding for a Systems-Level Cognitive Architecture

Introduction

In this chapter I continue describing a detailed perception network intended for tight integration with a broad systems-level cognitive architecture, LIDA. Having introduced the HTM Cortical Learning Algorithms (CLA) in the previous chapter, here I present a predictive coding extension to them termed PC-CLA. I first outline the theoretical motivation for PC-CLA. Next, I describe PC-CLA and its implementation. Then, I report the results of tests of PC-CLA. Finally, I discuss the theoretical considerations of integrating PC-CLA with the systems-level LIDA model and its commitments.

Motivation. In the previous chapter I reviewed Generalized Filtering (Friston et al., 2010), a general, biologically plausible, Bayes-approximate *mathematical* prescription for an advanced filtering network. I am interested in a *computational* model of pattern recognition like several of those reviewed earlier (Chapter 2). In particular, the computational HTM Cortical Learning Algorithms (CLA) incorporates sparse distributed representation (a consequence of the free-energy principle), hierarchical decomposition, and representation of higher-order temporal dynamics, with online learning in a single algorithm. However, not all the aspects of Generalized filtering appear in CLA including 1) the use of uncertainty and its estimation, and 2) the optimization of network states, parameters, and, hyperparameters, based on prediction error minimization (free-energy minimization). The cross-pollination of the ideas in these two theories forms the theoretical guide of this work. The ambitious goal is to develop a perception network with the capacity to process a general range of sensory signals in accordance with the

free-energy principle, which, statistically speaking, performs the triple-estimation problem of inferring model states, parameters, and hyperparameters (Friston et al., 2008). From another view, such a research program aims to capture the suggested functionality of the neocortex as a general pattern learner, using only *time* as a guide in capturing co-occurrence patterns (based on simultaneity) and sequences (temporally ordered) of co-occurrence patterns.

In order to keep to scope of this problem manageable, I will not approach all aspects of the problem at once. While I have discussed the representation of uncertainty (via hyperparameters) theoretically, I will not approach it computationally in this work. Rather, I aim to develop a predictive coding extension to the HTM Cortical Learning Algorithms[1], which I term the Predictive Coding Cortical Learning Algorithms (PC-CLA), as a first step towards the vision of a computational network of the form outlined by Generalized Filtering. While GF does not restrict the types of state variables used, PC-CLA uses binary variables. Table 9 summarizes some of the pertinent features addressed by the CLA, PC-CLA, and GF models.

Again, while PC-CLA may have more narrow applications as a data analysis or pattern recognition algorithm, I mainly propose it here as a detailed, fundamental building block for cognitive systems. In particular, in a hierarchy, it can be seen as implementing the current representations, memory, and learning and attentional processes for a systems-level cognitive architecture. As with the CLA, the core algorithm as I will describe it is recursive, so that it can conceivably be repeated in a large tree-structured

[1] Numenta's CLA has been implemented in C++ and deployed for commercial purposes. Their whitepaper on CLA gives its pseudocode, from which I have developed my own implementation in Java with the changes mentioned in this dissertation.

Table 9. Comparison of CLA, PC-CLA and GF by the features they address.

Feature	PC-CLA	CLA (2011)	GF (2010)
Online learning	X	X	X
Unsupervised learning	X	X	X
Distributed representation	X	X	
Sparse representation	X	X	
State optimization / Predictive coding			
Temporal prediction within same level	X	X	X
Top-down prediction into lower level	X		X
Top-down prediction error passed feedforward	X		X
Parameter optimization / Learning			
Spatial (Coincidence) learning	X	X	X
Temporal learning of various orders	X	X	X
Hyperparameter optimization / Gain modulation			
Precision estimation			X
Precision weighting of prediction errors			X

hierarchy, possibly following Fuster's (2006) outline of two intertwined perceptual and executive memory hierarchies. I conjecture that, out of such a network, I will be able implement cognitive features traditionally studied by LIDA and similar cognitive systems. Additionally, an agent built from a PC-CLA network would have the capacity to deal with high dimensional "real-world" sensory data streams in a grounded manner (Barsalou, 1999). Currently, the LIDA model does not have any faculties, in the form of a detailed algorithm, to deal with such high-dimensional inputs. Previously, LIDA research has simplified or "stubbed" the perceptual aspects of the agent.

PC-CLA: A Predictive Coding Extension to the HTM Cortical Learning Algorithms

PC-CLA is constructed as an extension to the CLA; as such the cortical region data structure and others defined in the previous chapter on the 2D-CLA remain the same. Thus, the extension pertains mainly to the cortical region process. In particular, I add predictive coding message passing (Rao & Ballard, 1999) between hierarchical levels, which allows cortical regions to be replicated hierarchically.

The predictive coding extension to the Cortical Learning Algorithms introduces the following changes to the cortical region process: 1) For the spatial pooling operation (Step 1, previous chapter), the column *overlap score* is now computed using prediction error. Specifically, input is the unpredicted (false negative) portions of the current bottom-up input, with respect to the current top-down prediction of the region. This contrasts with CLA that simply processes the original bottom-up input. 2) For the spatial pooling operation, when a column is predicted in the immediately previous cycle to become active in the current cycle, its bottom-up *overlap score* is assigned a value based on the *predicted column overlap* parameter. This is done because predicted columns, due

to the previous change, effectively inhibit their bottom-up input; however, the column should still compete with other columns to be part of the active columns representation. The rationale here is that we must maintain the temporal context encoded by this column's cells, and since it will not receive any bottom-up input (since its being predicted effectively inhibits any bottom-up input), we must give it a typical *overlap score* for the column to remain competitive. 3) The spatial learning operation now updates the proximal synapses on each column in the region, not just the active columns. Additionally, this procedure now strengthens all synapses connected to an active input (to minimize future errors from that input) and weakens all synapses whose input was predicted, but did not become active. 4) A new function is added that processes top-down predictions received from the immediately higher level. It runs serially after the calculation of the currently predicted cells, but before the temporal pooler learning operation. It computes the prediction error between the current top-down prediction and the union of the current active cell state and the current predicted cell state. It also adds in the current top-down prediction into the current cell activity state (4a and 4b in Figure 36 below). 5) A new function is added that generates top-down prediction to be sent to the immediately lower level. Specifically, it converts the current predicted columns (columns with current predicted cells) into an expected input pattern, using the columns' proximal dendrites and proximal synapses (5b in Figure 36).

Predictive Coding Cortical Region Process. In this PC-CLA implementation, each cortical region is driven by its own *predictive coding cortical region process,* which runs a serial cycle that updates the region's state and performs learning. This cycle is depicted in Figure 36, and I first describe it at a high level below.

The predictive coding cortical region process can be roughly subdivided into three sub-processes, *spatial pooling*, *temporal pooling*, and *message passing*. Spatial pooling refers to the process of grouping similar inputs into the same (or nearly the same) sparse distributed set of active columns that represent the input. Spatial pooling approximately corresponds to Step 2 below in the detailed description. Temporal pooling, corresponding with Step 3, occurs sequentially after the spatial pooling step, and takes the active columns representation produced by spatial pooling, and the current temporal context encoded by the cells predicted in the last cycle to become active this cycle, and produces the current active cell state. From the current active cell state the temporal pooler also produces a current predicted cell state, the context for the next cycle. Both the current active cell state and the current predicted cell state comprise the temporal pooler's output. Covering at least two time steps, the union of these two states should exhibit some temporal invariance, hence the name temporal *pooler*. The *message passing* sub-process (Steps 4 & 5) implements hierarchical predictive coding, which involves computing the prediction error between 1) the top-down prediction from a higher cortical region and 2) the union of the active and predictive cell states of the region. This error is then passed forward to the next higher level. With this high-level description in mind, I now summarize the main loop of the predictive coding cortical region process (Figure 36) at an arbitrary cycle *t* as follows:

Step 1. Compute the current bottom-up prediction error, ε_v, between the current bottom-up Boolean input, y, and the previous cycle's top-down prediction, y^{TD}.

Step 2. Compute the active columns of the cortical region for cycle *t*, *L1*.

a) Perform process *g* taking the bottom-up prediction error, ε_v, and the columns' proximal

dendrites and associated proximal synapses, and outputting the columns' *overlap score*.

b) For each column predicted to become active this cycle (computed in Step 5a of cycle $t - 1$), set its *overlap score* to the *predicted column overlap* parameter.

c) For columns with an *overlap score* greater than the column overlap threshold, perform a local *k*-winners-take-all procedure to determine the active columns, *L1*. The constraint, *k*, limits the number of possible active columns within a given area ensuring that the active columns are distributed.

Step 3. Compute the active cells at cycle *t*, *L2*, the current cells predicted to be active at *some* future cycle, $PL2_t$, and their union, *U*.

a) Based on the active columns, *L1* and the currently predicted cells, $PL2_{t-1}$ (computed in Step 3b of cycle $t - 1$), compute the current active cells, *L2*.

b) Based on the active cells, *L2*, and the region's distal dendrites and synapses, perform process, *f*, producing the region's current predicted (for some future cycle) cells, $PL2_t$. Based on only the cells predicted for the next cycle $t + 1$, determine the columns, predicted this cycle, to be active next cycle, $PL1_t$ (used later in Step 5).

c) Compute the union, *U*, of the active cells, *L2*, and the current predicted cells, $PL2_t$.

Step 4. Process the current received top-down prediction, U^{TD}.

a) Compute the error between *U* and the current received top-down prediction, U^{TD}, and send the error to the next hierarchical level.

b) Update $PL2_t$, the current cells predicted to be active at *some* future cycle, by adding in those cells predicted in U^{TD}.

Step 5. Based on the columns predicted to be active next cycle, $PL1_t$, (found in Step 3b):

a) Compute each column's *predicted column overlap* (used in 2b of next cycle).

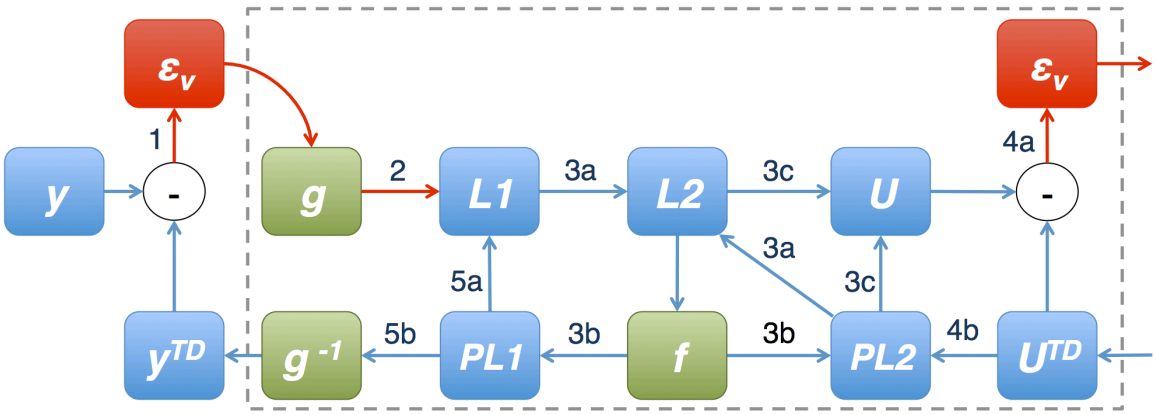


Figure 36. The predictive coding cortical region process for a single hierarchical level with an input. All the components of the single cortical region appear within the gray dashed box. Current representations are shown in blue with the exception of current errors, which are shown in red. Green boxes represent the processes using long-term synaptic connections.

b) Perform process g^{-1} to generate the region's current top-down prediction, y^{TD} .

Step 6. Perform the learning processes.

a) Perform spatial learning, updating the permanence of proximal synapses based on bottom-up prediction error. Also update each column's *boost* attribute based on its activity history.

b) Perform temporal learning, which updates distal synapse permanence, and possibly adds new distal synapses. This learning is driven by both unpredicted columns and predicted columns that did not actually become active. I describe the details of these processes in the next section.

This concludes a high-level description of the PC-CLA algorithm. For more detailed pseudocode of key portions of the algorithm, see Appendix B.

Learning in PC-CLA. Here I give the additional details for the learning Step 6 of the predictive coding cortical region process. In the original CLA, the spatial learning process iterates over each current *active* column, and updates the permanence values of

the column's synapses based on the synapse's input. Synapses with active inputs have their permanence incremented, while those with inactive inputs have their permanence decremented. This has the effect of tuning columns to be more selective to their current bottom-up inputs. In PC-CLA, this part of the algorithm is changed, and the process iterates over *all* columns in the cortical region and updates the permanence values of each column's synapses according to any bottom-up prediction error occurring for the synapse's input. In detail, synapses with active inputs due to false negative prediction error have their permanence increased, while those responsible for a false positive prediction error have their permanence decreased. In PC-CLA, the boosting update algorithm remains unchanged, as well as, in the temporal pooler, the learning updates to distal dendrites segments.

In this section I have described the basic details of the PC-CLA algorithm. In Chapter 6 I, provide more details of my specific computational implementation. In Appendix C, I give a detailed time complexity analysis of the algorithm.

Temporal Pooler Tests

Here I first present some tests exploring the temporal pooling aspect of the CLA (Step 3), which has seen little in the way of published reports. These tests study a single cortical region performing both the spatial pooling process and, additionally, the temporal pooling process. Temporal pooling attempts to learn temporal patterns in the temporal progression of the sparse distributed representations produced by the spatial pooler. Once learned, the temporal pooler predicts the next expected pattern. Assessing the accuracy of such predictions is the subject of the first test below. Additionally, the temporal pooler tries to predict more than one time step in the future, which serves to build temporal

invariance in the cortical region's output. Assessing this type of prediction accuracy is the subject of the second test below. Common parameters between these tests, and their values, are summarized in Table 10.

First-Order Temporal Prediction Accuracy Test. The cells of the columns of the cortical region encode the various temporal contexts required for variable-order temporal prediction, the ability to predict the next state (active cells) based on a variable number of previous states (Hawkins et al., 2011, pp. 16–17). In addition, the distal dendrite segments and their synapses implement the long-term temporal memory used to predict cell states from the different activity contexts encoded by active cells. In this test, I explore the first-order prediction capabilities of the temporal pooler, i.e., predictions of the expected set of active cells for the very next processing cycle. For the test I generate sequences of randomly generated 2D Boolean patterns having the form *prefix-cue-target* (Figure 37). The prefix is a subsequence, and each pattern in it is randomly generated for each sequence. The prefix's length is an independent variable of the test. The cue portion of sequences consists of a single pattern occurring immediately after the last pattern of the prefix. The cue pattern is fixed across all sequences, and has its true bits uniformly distributed throughout the input space to minimize potential bias. Finally, the target portion consists of a single pattern randomly generated for each sequence.

A single trial in this test consists of the generation of a sequence having the described form, which is then repeatedly input to a cortical region performing spatial and temporal pooling with learning. The number of repetitions is a test parameter termed *developmental exposures*. After the final pattern, the target pattern, is presented at the end of each developmental exposure, the current and predicted states of cells and columns are

Table 10. The temporal pooler parameters shared across the two temporal pooler tests.

Parameter name	**Explanation**	**Value(s) used**
Input dimensionality	Dimensionality of input signal	529
Input activity	Ratio of the number of true bits in the input to the total number of input bits	0.02
Initial synapses per distal dendrite segment	Initial potential synapses assigned to each distal dendrite segment at setup time	20
Maximum synapses per segment update	Limit on the number of new synapses added by a single segment update	20
Maximum possible synapses per distal segment	Limit on the number of possible synapses on any distal dendrite segment	32
Distal learning radius	Radius of the circle defining the neighborhood of cells about a distal dendrite from which synaptic connection can potentially be made	7
Distal dendrite segment activation threshold	The number of distal synapses that must be active before a distal dendrite segment is considered active	1
Segment learning threshold	The number of distal synapses required to be active on a distal segment for learning	1
Initial distal synapse permanence	The initial permanence of distal synapses at setup time. This value should be below the connection threshold	0.4
Distal permanence increment	Change in permanence for those distal synapses deemed worthy of strengthening	0.1
Distal permanence decrement	Change in permanence for those distal synapses in need of weakening	–0.1

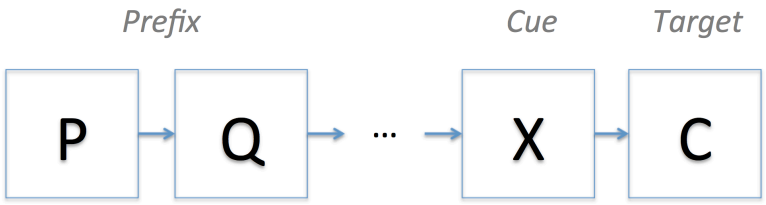


Figure 37. Illustration of the type of sequences used for the First-Order Temporal Prediction Accuracy Test. The letters represent randomly generated 2D Boolean patterns. The prefix length was an independent variable. The cue was fixed across all sequences and the target was randomly generated for each sequence.

cleared to clear the current context. The motivation for this is to control for the learning of a temporal transition from the last pattern to the first.

After the final developmental exposure of a sequence, the final set of active columns, which represents the active columns representation of the target pattern, are recorded. After the developmental period, I present the initial part of the sequence one final time with learning turned off. Specifically, I only present the patterns of the sequence up to, and including, the cue pattern, that is, the final target pattern is not shown. Then, I retrieve those columns predicted for the next time step, i.e., the prediction of the target pattern's active column representation. Based on these predicted columns and the original active columns representation of the target, the F_{10} score (the F-score with recall valued 10 times as much as precision) is computed and recorded for the trial. To test the cortical region's performance in representing the context of the sequence, I vary the size of the sequences' prefix. I also vary the number of distal dendrite segments, which gives a cortical region increased temporal memory. The results are summarized in Figure 38.

There were 7,500 trials in each condition. Each sequence had 4 developmental exposures to the region. The cortical region had 7 cells per column and 20 distal dendrites per cell.

Higher-Order Temporal Predictions Test. The CLA is capable of making greater than first-order predictions, that is, predictions of cell activity at two or more cycles in the future. These serve to produce temporal stability or invariance in the cortical region's output when viewed over time (Hawkins et al., 2011, p. 17). This can be seen by noting that 1) higher-order temporal predictions can put cells into a predicted activity state several cycles in advance, and 2) both the predicted and active cell states are combined in

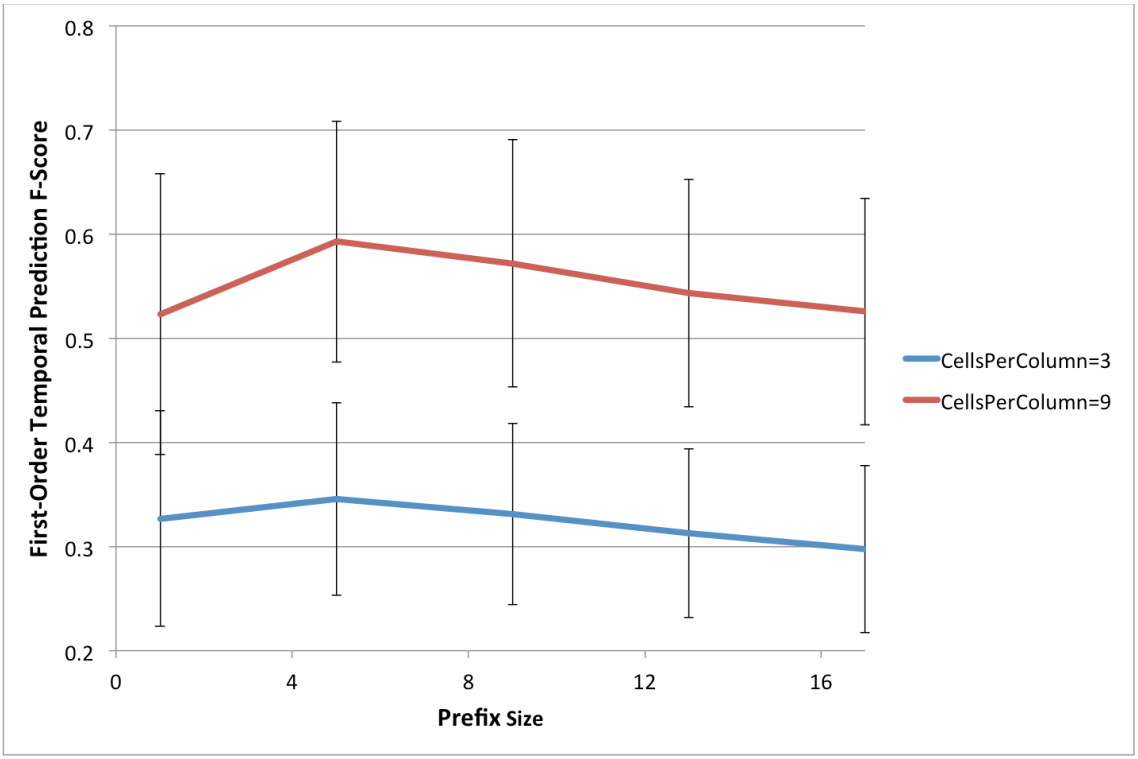


Figure 38. Accuracy of first-order temporal predictions versus the prefix size of inputs. The prefix size appears to have some, though not a great, effect on prediction accuracy for the range of sizes tested. The number of cells per column did have a significant effect on first-order temporal prediction accuracy as expected. Note adding cells increases the cost of the algorithm in terms of time and space.

producing a region's output. In this test, I aim to study how well CLA performs this function. Germane to this, I define a parameter of the cortical region called *prediction order limit* controlling the highest order of segment updates (and consequently the highest-order prediction of segments) that can be stored (and possibly performed) by the temporal pooler. If performed, such segment updates modify distal synapses to better predict patterns two or more cycles in advance.

In each trial of this test, I first generate a sequence of 2D Boolean patterns having a fixed size determined by the test parameter, *sequence size*, which was set to 4. With each generated sequence, I perform the following procedure for a range of *prediction order limits*. The procedure generates a cortical region using the specified *prediction order limit*, and then performs a developmental phase, repeatedly running the sequence on the cortical region, which performs spatial and temporal pooling with learning. At the end of each sequence presentation, the current and predicted states of cells and columns are cleared to control for learning that the last pattern predicts the first. After the final developmental sequence presentation, the final set of active columns, representing the final pattern of the sequence in its temporal context, are recorded. For the crucial test phase, I check the accuracy of predictions for this final set of active columns. For the test phase, I run the sequence a final time, pattern by pattern, at each step comparing the current temporally predicted columns, including all higher-order predictions, with the recorded final active columns representation of the sequence. For each comparison, the F_{10} score between the currently predicted columns and the actual active columns representation of the final pattern is recorded. The idea here is to determine the degree to which the final set of active columns is predicted in advance at each step in the sequence.

To summarize the results, I compute the average F_{10} scores, across each randomly generated sequence, for the sequence steps 1 to *sequence size* – 1, specifically, 1 to 3 (Figure 39).

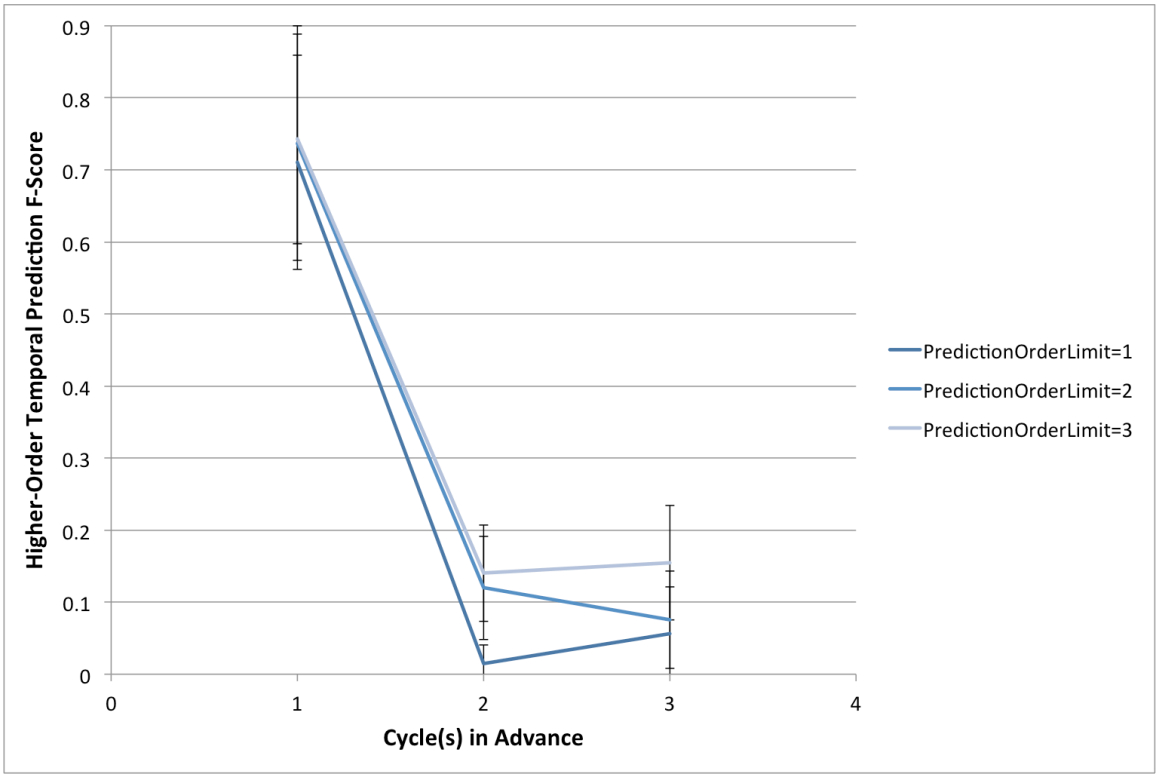


Figure 39. Accuracy of higher-order (> 1) temporal predictions versus number of cycles in advance of the final sequence pattern. For one cycle in advance there was no significant difference between learning higher-order predictions (e.g., two and three-order) and not. For two cycles in advance there was a significant improvement in accuracy for both two and three-order limits over only a one-order limit. Finally, while not significant, a three-order limit bested both one and two-order limits in predicting three cycles in advance.

There were 300 trials in each condition. The size of each sequence was 4. Each sequence had 1,000 developmental exposures to the region. The cortical region had 8 cells per column and 25 distal dendrites per cell. The results of the test suggest that while the cortical region can learn to predict in advance, there are significant limitations such

higher-order predictions, at least for sequences of randomly generated patterns. Future work should study this capability of the algorithm on more natural data streams.

PC-CLA Tests

In this section I present an initial test of the PC-CLA algorithm that implements the predictive coding cortical region process (Figure 36).

Reconstruction Accuracy Test. The motivation for this test comes from the desire to arrange multiple cortical regions in a hierarchy with adjacent levels passing messages according to hierarchical predictive coding. If top-down predictions are sent top-down from a cortical region at one level to a lower region, then information must be lost in the translation from the initial columnar representation of predictive columns in column space to an expected input in input space. This is because the journey in the feedforward direction involves a nonlinear, many-to-one generalization (many proximal synapses to one column). Also the input space and column space typically differ in dimensionality. To what degree is information lost (patterns become corrupted) with this inter-region translation? And what is the effect of columns' receptive field size?

To answer these questions, I developed the Reconstruction Accuracy Test. For a range of receptive field radii, I generated a cortical region with columns having the specified receptive field radius. In each such condition, a series of 5,000 random inputs, having an *input activity* of 2%, were then generated, and each input was presented to the cortical region. An *input activity* of 2% was used, since I found an increase in reconstruction accuracy for inputs with lower percentages of active bits. Furthermore, we can expect a potential lower cortical region to produce outputs of near 2% activity, in short because a well-tuned spatial pooler will tend to produce sparse sets of active

columns.

The cortical region performed the spatial pooling operation for a short developmental period of 200 cycles. After this period, the resultant (stable) active columns representation was recorded for the particular input. Next, a top-down prediction or "reconstruction," based on this same active columns representation, was generated, which had the same dimensionality as the original input. This prediction was created by iterating over the active columns, and, for each connected synapse on each column, marking the bits corresponding to the connected synapse's input index true. I experimented with other methods of producing this prediction, in particular, using scalar

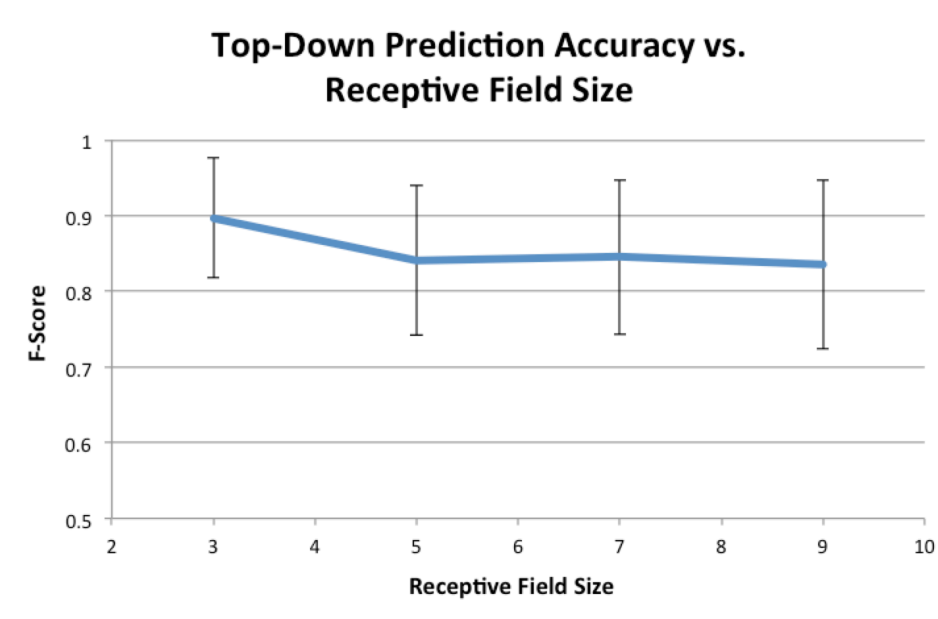


Figure 40. The reconstruction accuracy versus the receptive field size. Reconstruction accuracy was operationalized as the F_1 score between a Boolean input and the top-down reconstruction of its stable active columns representation. The data suggest a slight drop in accuracy with larger receptive field size.

measures of "sufficient" predictions; however, I found that a policy of counting one or more synapse(s) as a prediction for its source input bit produced the best reconstruction accuracy.

To evaluate the accuracy of the reconstruction I used the F_1 score between the original input and its reconstruction. The results of the test are shown in Figure 40. The data suggest a slight decrease in reconstruction accuracy as receptive field size is increased; however the result was not statistically significant. So, at least for the range tested, the effects of the receptive field size on prediction accuracy can be ignored.

Predictive Coding Spatial Learning Test. The purpose of this test is to compare a cortical region that processes and learns from the original input pattern (regular) with a cortical region that processes and learns from the top-down prediction error in its input (predictive coding). The difference between the original CLA's spatial learning and that of PC-CLA's is summarized in Tables 11 and 12 below. Comparing the last rows of each table, it can be surmised that PC-CLA learning saves on the number of operations performed since two of its four possibilities require no change.

For the predictive coding condition, the *predicted column overlap* (see bottom of p. 172 for definition) parameter was used and set to 0.5, which corresponds to half of the potential synapses on each column's proximal dendrite segment. As such, predicted columns should very strongly compete with bottom-up active columns in the inter-column inhibition.

For each of 10,000 trials, I performed the following steps: 1) Generate a sequence of length 3 of random patterns having 2% true bits, 2) Generate a single standard cortical region and a single predictive coding cortical region, 3) Develop each region on the

sequence by repeatedly exposing the sequence to the cortical region 50 times with learning, 4) Expose the sequence to each of the two cortical regions again, this time without learning, recording how well each of the region's current top-down prediction matches the next input pattern.

Table 11. Spatial learning in the original CLA for binary input. The synapses of active columns are updated to better match the input. During learning, synapses are changed solely in reponse to their inputs. Whether the synapse is connected does not influence the learning update.

Case	**Synapse connected**	**Synapse's source active**	**Change to synapse**
1	Yes	Yes	Strengthen
2	Yes	No	Weaken
3	No	Yes	Strengthen
4	No	No	Weaken

Table 12. Spatial learning in PC-CLA for binary input. The synapses of all columns are updated to minimize the current top-down prediction error. That is, learning only updates synapses whose connectivity is inappropriate given the current input. In contrast with Table 11, for cases 1 and 4, no change is made.

Case	**Synapse connected**	**Synapse's source active**	**Change to synapse**
1	Yes	Yes	No change
2	Yes	No	Weaken
3	No	Yes	Strengthen
4	No	No	No change

I first evaluated the spatial learning in both conditions by assessing the accuracy of the cortical regions' top-down predictions in terms of the F_1 score. Surprisingly, I found that the regular condition had a significantly better score than the predictive coding condition (Figure 41).

I wanted to get a better sense of why the standard learning approach outperformed the predictive coding one. I additionally measured the connection rate of the proximal

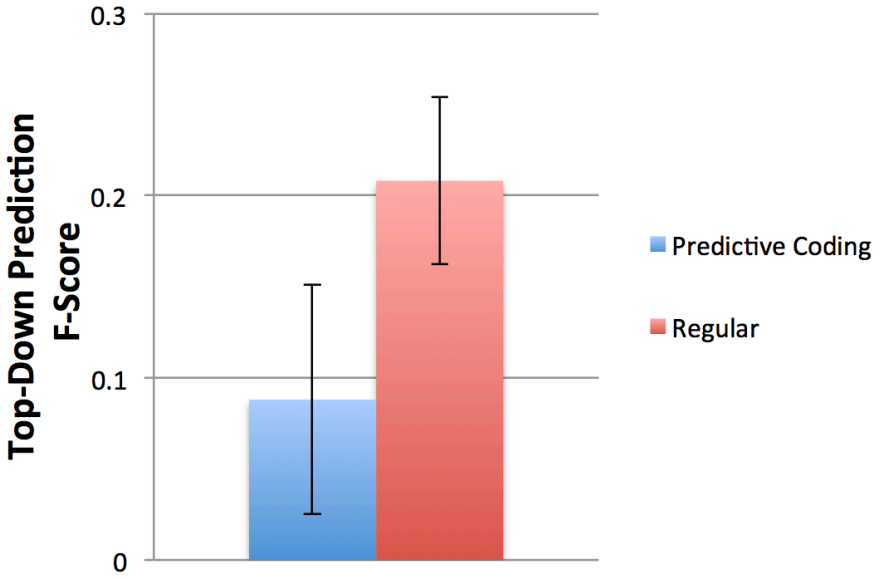


Figure 41. Top-down prediction F-score.

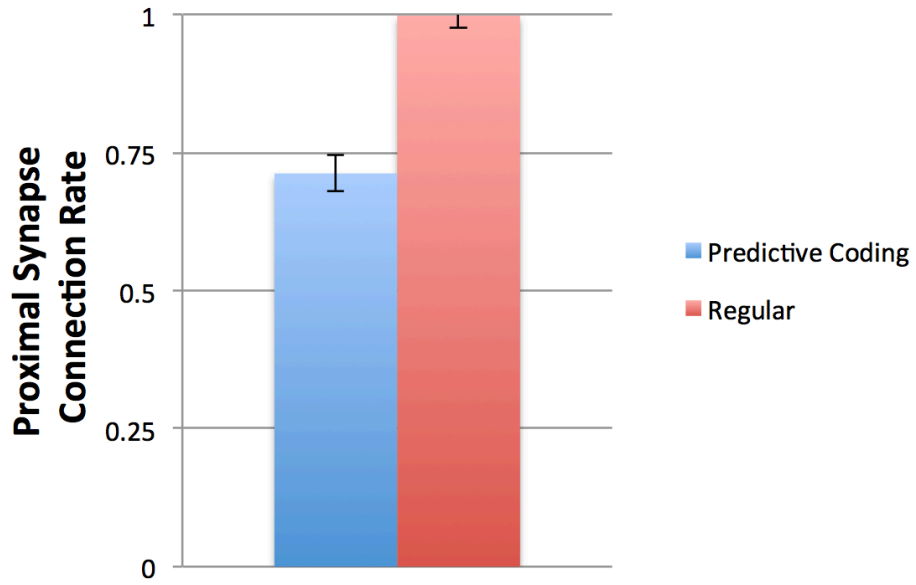


Figure 42. Proximal synapse connection rate.

synapses of each cortical region's columns since the *connected* synapses of predicted columns are counted in computing top-down predictions (see Appendix B for the pseudocode details). I found that the regular condition connected nearly all of its proximal synapses while the predictive coding condition connected significantly less (Figure 42).

Given that the regular condition outperformed predictive coding in top-down prediction accuracy, but used significantly more synapses in doing so, I desired to further explore what was happening. I suspected that with all the connected synapses, perhaps the regular condition was simply predicting more to get a better prediction score. To explore this, I additionally recorded the true bit rate of the top-down prediction generated by both conditions on this test. I found (Figure 43) that the predictive coding condition produced significantly sparser predictions (average of 1%) than the regular condition (average of 9.4%). I have not yet mentioned the necessity of sparse top-down predictions. Conceptually, this need arises in multi-region networks where top-down predictions are sent to influence lower cortical region. When this occurs the active bits in the prediction are added into the predicted cell state of the lower region (Step 4b). In short, if this addend is not sparse it can break the sparsity of the lower region's representation. The need for sparse top-down predictions also arises empirically, specifically, in the next test with multiple cortical regions.

In Tables 11 and 12 I hypothesized that the predictive coding would involve fewer synaptic changes than the regular condition. To verify this, I recorded a count of each change made to proximal synapses during the test. The results (Figure 44) indicate

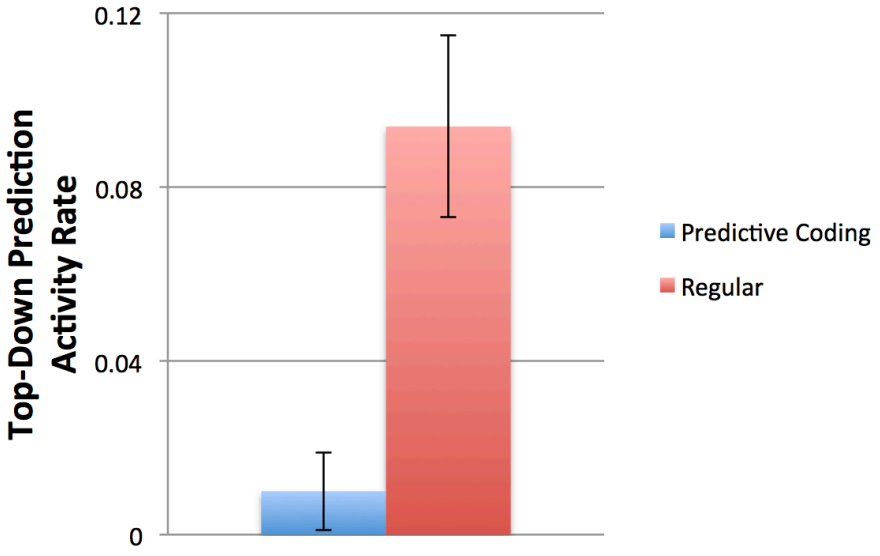


Figure 43. Top-down prediction activity rate.

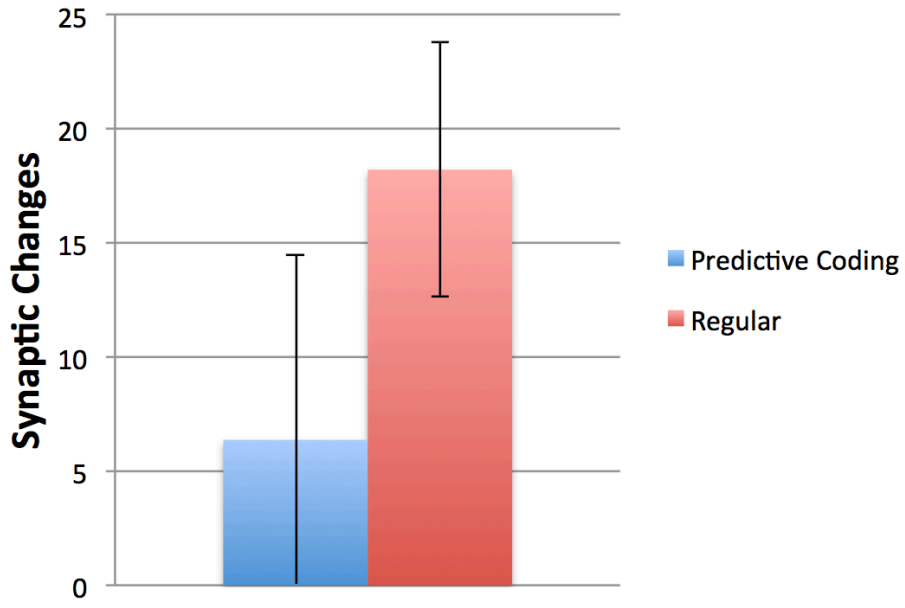


Figure 44. Proximal synapse changes.

that the regular condition indeed involves more synaptic changes than the predictive coding condition, although this difference was not quite significant.

In summary, the regular spatial learning condition showed improved top-down prediction accuracy at the cost of requiring more synaptic connections and changes to do so. In contrast, the predictive coding condition desirably produces sparse top-down predictions at the cost of prediction accuracy. While I suggest more exploration of this tradeoff, it appears that for single-level cortical region networks it may be better to use the regular spatial learning approach. However, for multi-level networks, the sparse top-down predictions of the predictive coding condition appear to make it the better choice.

Two Hierarchical Levels Test. This test studies the effects of having top-down predictions, from a higher cortical region to a lower one, in a two-level cortical region network. In particular, I compared a two-level two-region network where the higher region does *not* send top-down predictions into the lower level with a similar network in which the higher region *did* send top-down predictions. In both conditions, I presented the network with a battery of temporal sequences of randomly generated patterns. In order to measure the effects of the top-down prediction manipulation, I recorded both the top-down prediction accuracy and temporal prediction accuracy, using the F_1 score, in both regions for each condition. I expect the top-down predictions to improve prediction accuracies in the lower region. A separate consideration in this setting is the degree of invariance in each level's output. It has been suggested that hierarchically superior regions will exhibit greater output invariance than lower regions. In order to test this, I also recorded the variation in the outputs of both regions in this network while processing temporal patterns.

The test was performed as follows: a two-level network was created by generating two cortical regions with the same parameters except that their *projection factor* parameters, which govern the number of columns per input for the region, differed. The first (lower) cortical region had a *projection factor* of 4.0, as is typical among the tests described thus far; however, in order to maintain the same output size for both regions, the second cortical region used a *projection factor* of $1/cellsPerColumn$, since the cell representation of the first cortical region increases the size the original input by a factor of *cells per column*.

In each of the 1,500 trials, a sequence of randomly generated patterns having length 3 was generated. During a developmental period, the sequence was then repeatedly shown 25 times to the network in the following manner: the first region processed the false negative prediction error of the original input and its current top-down prediction for that input. After this, the first region's output was obtained and the second region processed the false negative prediction error of the first region's output and region two's current top-down prediction. Both regions performed the full predictive coding cortical region process (Figure 36). After the higher region ran each step, its current top-down prediction was sent, and incorporated into, to the first region. In the test phase of each trial, both regions' current states were reset, and the same sequence was again shown to the network as before except without any learning.

During this test presentation, for each pattern in the sequence (except the first), the first-order temporal prediction accuracy of the active columns was recorded for both regions in terms of the F_1 score. Similarly, the top-down prediction accuracy of each region's prediction of its input was also recorded. Additionally, each region's output to

the next hierarchical level (Step 4a) was recorded for each pattern as well. Once the entire sequence had been presented during the test presentation, the average top-down prediction accuracy and average temporal prediction accuracy for the sequence was computed as well as a single invariance measure across the sequence, for each region.

The invariance measure was determined by first computing the standard deviation of the usage of each of the output dimensions across the sequence. Then, the standard deviations were averaged for an overall invariance measure. Across all sequence trials these measures were again averaged for the final results. This measure of invariance is low (near 0.0) for little varying patterns and higher for patterns of high variance. For instance, I found that 500,000 random Boolean vectors of 256 dimensions had an invariance of about 0.138.

The top-down prediction accuracy results are summarized in Figure 45. At the first hierarchical level the top-down predictions were moderately accurate and top-down predictions had no significant effect on accuracy. However, at the second hierarchical level having top-down predictions signicantly improved top-down prediction accuracy. While not statistically significant, it appears that top-down predictions give a better prediction accuracy at the second level than either condition at the first level.

Figure 46 depicts the temporal prediction accuracy results of the test. Again at the first hierarchical level there was not a significant difference in prediction accuracy between the two top-down prediction conditions. However, at the second level having top-down predictions significantly improved the temporal prediction accuracy. Also worth noting is that the temporal prediction accuracy at the second level in the top-down influence condition is significantly better than at the first level. It appears likely that, even

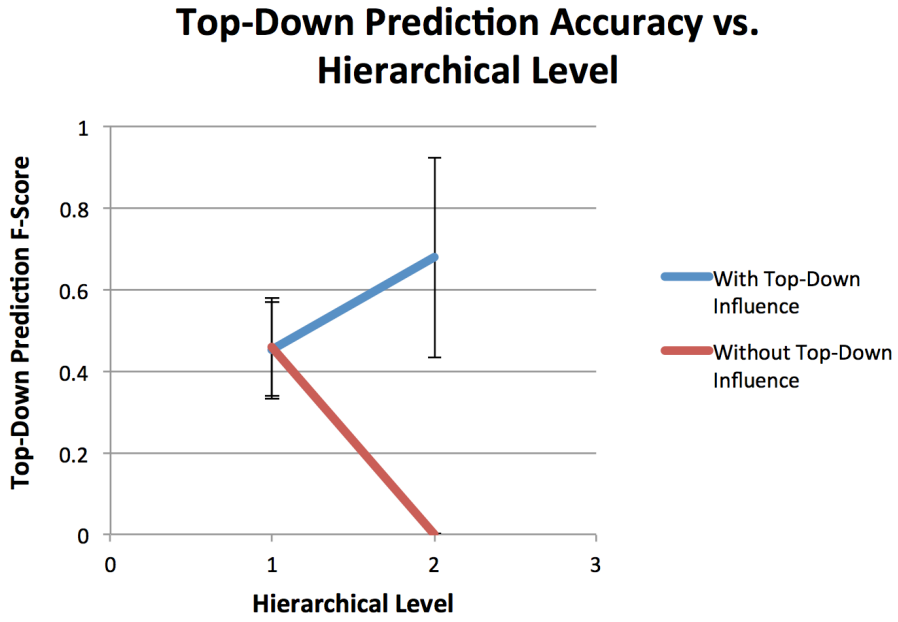


Figure 45. Top-down prediction accuracy versus hierarchical level.

without top-down influence, the second level might be outperforming the first in this respect although statistical significance cannot be claimed.

The output variation results of the test are plotted in Figure 47. There was no significant difference in output variation across the experimental conditions at the first hierarchical level. At the second hierarchical level there was a significant decrease in output variation compared to the first level for both conditions. A lower score signifies more temporally invariant outputs. To give these results context, I computed the variation of 500,000 random Boolean patterns having 256 dimensions (the test's input size) and a 2% true bit bits and obtained an invariance score of 0.138 (standard deviation of 0.00066). This suggests that the algorithm is indeed producing more invariant outputs than it receives.

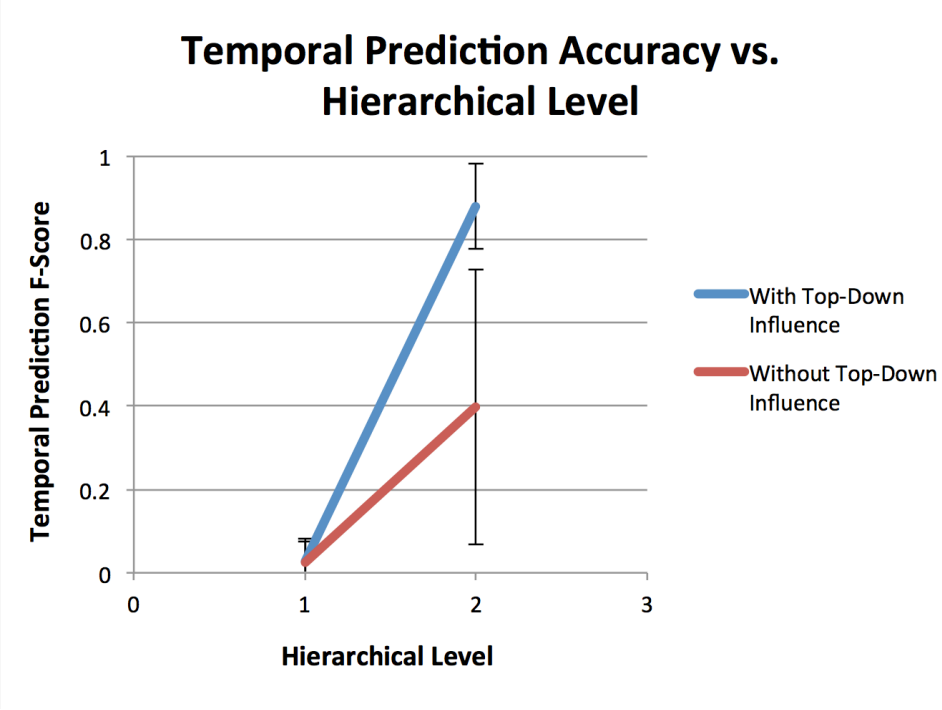


Figure 46. Temporal prediction accuracy versus hierarchical level.

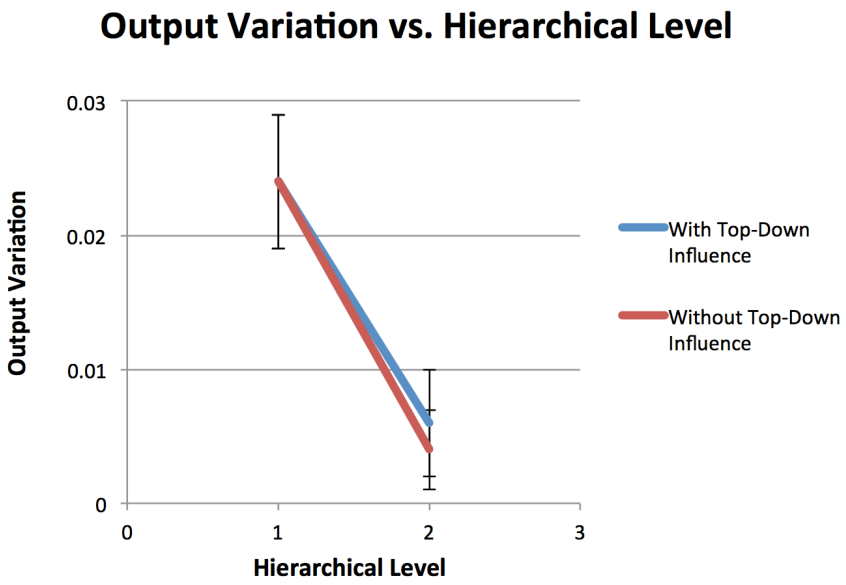


Figure 47. Output variation versus hierarchical level.

In this section I have presented the results of tests of the temporal pooling portion of the CLA algorithm along with tests pertaining to the addition of hierarchy to the CLA algorithm including the effects of predictive coding, multiple hierarchical levels, and top-down predictions.

The First-Order Temporal Prediction Accuracy Test explored the effects of the *cells per column* parameter and the amount of context on the accuracy of first-order temporal predictions in the CLA. It showed a signifcant improvement in accuracy for a larger value of the *cells per column* parameter. It also showed a less pronounced, but still discernable, effect for the context length manipulation.

The Higher Order Temporal Predictions Test showed that the CLA could learn to make higher-order predictions on sequences of random patterns. Nonetheless, the higher-order predictions were significantly worse than first-order predictions.

In preparation for testing multiple hierarchical levels, the Reconstruction Accuracy Test studied the effect of receptive field size on top-down prediction accuracy. It found that increasing the receptive field size had a slightly negative effect on accuracy for the ranges of receptive field size tested.

The Predictive Coding Spatial Learning Test compared a predictive coding style learning algorithm for spatial learning with the standard CLA-prescribed algorithm. Interestingly, the predictive coding condition gave worse top-down prediction accuracy than the standard. However, in its favor, predictive coding connected considerably fewer synapses, performed less synaptic updates, and produced sparser top-down predictions than the standard learning algorithm. Additional studies of these tradeoffs are in order.

Finally, the Two Hierarchical Levels Test studied the effects of top-down predictions in a two-level two cortical region network. I found top-down predictions to be a boon improving both the top-down and temporal prediction accuracy at the second hierarchical level with no significant difference at the first level. Top-down predictions did not have a significant effect on the invariance of either level's output. Notably, the second level exhibited greater output invariance than the first as is suggested by HTM theory.

Theoretical Considerations for PC-CLA and LIDA

I propose PC-CLA as a detailed, low-level, uniform algorithm, which, replicated recursively in a large-scale hierarchical network, might implement the more abstract, global, cognitive features comprised by the LIDA model, such as its Current Situational Model, various types of memory, and attention and learning processes. The various aspects of PC-CLA correspond to several entities in LIDA: the notion of hidden state variables and their values corresponds to the current representations in the Current Situational Model of LIDA's Workspace. The long-term synapse-like connections of PC-CLA could potentially implement one or several of LIDA's short-term Sensory Memory, Perceptual Associative (recognition) Memory, Procedural Memory, Declarative Memory, etc. PC-CLA creates segment updates corresponding to proposed synapse-like connections. Under certain conditions the segment update is performed and the connections are modified. Similarly, LIDA's structure building codelets can propose new temporal links in the Workspace, and, if such links are part of a structure winning the competition for conscious, then they are learned into any appropriate memory. While LIDA's attention mechanism employs attention codelets that create competing coalitions

for a winner-take-all competition for consciousness, Generalized Filtering's attentional mechanism is based on precision-weighted prediction errors, at each hierarchical level (Feldman & Friston, 2010). Though the connectionist PC-CLA theoretically overlaps with a wide range of cognitive entities in the systems-level LIDA model, reconciling the two requires several theoretical considerations.

Sparse Distributed Representation. Conceptually, PC-CLA adheres to sparse, efficient coding (Friston, 2010, p. 5), while, computationally, it makes strict use of sparse distributed representation (Hawkins et al., 2011). For instance, bottom-up inputs are (typically) represented by a sparse set of active columns (typically) distributed throughout a cortical region. If LIDA adopts such distributed representation for its underlying internal representations, it would be a departure from its previous use of localist, graphical, "node-and-link" representation. With PC-CLA, an instantiated node would be theoretically implemented as a sparse distributed representation involving one or more hierarchical levels, depending on the degree to which the node was hierarchically decomposed. Uninstantiated nodes would be encoded by the permanences of proximal and distal synapses likely across multiple cortical regions.

PC-CLA can be viewed as implementing two possible types of links: "structural links" encoding hierarchical decomposition and "dynamic links" encoding level dynamics. Structural links correspond to "feature-of" and "is-a" links in current LIDA terms (McCall et al., 2010). Furthermore, a particular collection of structural (proximal) links would constitute the memory for a node in the sense that they define the nodes component features in the immediate lower level. Structural links are implemented by a collection of PC-CLA's artificial dendrite segments and their synapses (described below),

with each synapse having its source in one hierarchical level and its sink in the lower level. Dynamic links correspond to LIDA's temporal and/or causal links. Dynamic links are implemented similar to structural links except that links' source and sink would be some computational unit within the same level. Related to this, Snaider (2012) has given a proposal for the use of high dimensional, distributed, "modular composite vectors" as the main data structure for the LIDA architecture.

PC-CLA and Global Workspace Theory. Hebb (1949) proposed the cell assembly theory, which first described Hebbian (or associative) plasticity. Under this theory groups of interconnected neurons are formed through a strengthening of synaptic connections based on correlated pre- and post-synaptic activity — simply, "cells that fire together wire together." Over repeated exposures associative plasticity allows the brain to glean statistical regularities in its sensory input. PC-CLA's (and some of the brain's) learning processes can be seen as a kind of Hebbian learning involving the plasticity of synaptic connections. On the other hand, Global Workspace Theory (GWT) (Baars, 1988) and its LIDA model, flesh out a distinct concept, *conscious learning.* LIDA's multiple modes of learning (Perceptual, Procedural, Transient Episodic, etc.) all occur continually, simultaneously, and online using each global broadcast of the contents of consciousness. I hypothesize that Hebbian learning must be the basis of LIDA's conscious learning, that is, numerous Hebbian connections, across multiple hierarchical levels, must be modified with each broadcast. Such a view also allows for synaptic plasticity with preconscious and never conscious processes. In support of this, synaptic plasticity has been suggested to account for unconscious priming (e.g., Chaumon, Schwartz, & Tallon-Baudry, 2009; Dehaene & Changeux, 2011; Dehaene et al., 1998).

In GWT, coalitions, incorporating the results of the processing of sensory data, compete for attention in the "global workspace." Does the same hold true for PC-CLA? Throughout a hierarchy of PC-CLA regions, messages are passed feedforward and top-down. Each such message can be viewed as a different interpretation of the current situation. Each message competes for representation in the region it is sent to with 1) the region's current expectations and 2) other received messages (each region receives one feedforward and one top-down message). It is possible for such a network to settle on a coherent consensus on the current situation across multiple regions via such message passing. The filter of GWT can be seen in a PC-CLA network as this process of the network settling on a particular set of current representations, which are typically widely distributed through the levels of a cognit-like network (Fuster, 2006). Such widespread settling on coherent representations in PC-CLA is proposed to correspond with the global broadcast of GWT. The "global workspace" in PC-CLA would be highly distributed across a hierarchical network with its locus dynamically changing from moment to moment, a view compatible with a recent account of a dynamic GWT (Baars, Franklin, & Ramsøy, 2013).

GWT supports the Conscious Learning Hypothesis: significant learning takes place via the interaction of consciousness with the various memory systems. PC-CLA is compatible with this view because the contents of consciousness in PC-CLA would typically concern a large representation distributed across thousands of representational units, which would likely require updating a significant number of long-term connections implementing PC-CLA's memory. One potential difference between PC-CLA and GWT is that PC-CLA's synaptic updating is primarily driven by top-down and temporal

prediction errors, in whichever areas they occur, while for GWT, learning is proposed to occur with each conscious broadcast. However, if prediction errors are taken to be highly salient then it becomes clear that the related representations will be highly likely to win the competition for consciousness, and become the target of learning. One last item to note is that learning based on a prediction error in this manner can be either instructionalist (adding a new representation) or selectionist (reinforcing an existing representation).

Precision and Salience. Previously in the section "Hyperparameter Optimization via Gain Control," I discussed the optimization of precision, considered as a hyperparameter from a statistical modeling viewpoint. Recall that precision in this context is a statistical term referring to the multiplicative inverse of the covariance of a model's hidden states. High statistical precision corresponds to a probability distribution highly concentrated about its mean, suggesting that the mean is known to a high degree of precision in the ordinary sense. Precision is then just another way to talk about a model's uncertainty. In the context of probabilistic inference on the states of hierarchical dynamic models, we are required to reconcile a given level's temporal predictions for the next time step with the actual bottom-up input (or observation) that is actually received. How do we determine which of these two quantities to "trust?" The answer is that we must perform a weighted average between the two quantities, with the precision being used as the weight. In Generalized filtering, this weighting is implemented by using precision quantities to weight prediction errors. The idea of the weighting is that sources having higher precision are "trusted" more. As a consequence, the prediction error from a model with low precision would carry less weight in updating another model's state, while a highly

certain model's prediction errors would carry the most. According to the equations derived in Generalized Filtering, the precision is updated principally using the sum of squared prediction errors, the same prediction errors that the precision weights. This has several implications for the LIDA model. Firstly, it suggests that precision (inverse covariance) modulates the salience of current representations via the multiplicative weighting of prediction errors. Also, it suggests that state-dependent (i.e., based on prediction errors) processes should be deployed at each hierarchical level of the Workspace to maintain and update precision quantities. Such processes are not concerned with what is being represented, rather with ongoing prediction error.

Interestingly, the neurobiological implementation of precision is thought to use those neurotransmitters that modulate synaptic gain (Feldman & Friston, 2010). Synaptic gain and its biasing effects have previously been associated with attention (e.g., Desimone & Duncan, 1995). Feldman and Friston (2010) have suggested that attentional processing can be simulated with a Bayes-optimal recognition scheme on models including uncertainty. They suggest attention can be seen as the "selective sampling of sensory data that have high precision in relation to the model's predictions." Put another way, this suggests that precision quantities contribute to the current salience of current representations. It also suggests that top-down predictions are another feature of salience as they can create an attentional "bottleneck" by biasing particular representations, preventing others from being expressed.

Following GTW (Baars, 1988) LIDA has defined attention as the process of bringing content to consciousness based on salience. Salience in this context includes importance, urgency, insistence, novelty, unexpectedness, motion, loudness, brightness,

goal or task relevance, etc. (Faghihi et al., 2012) Following the free-energy principle, salience in potential future versions of PC-CLA would appear as precision-weighted prediction errors. These errors drive the update of the representations of the causes of sensory input while the precision governs the magnitude of such changes. Happily, LIDA's psychological notions of salience don't contradict the idea of salience based on precision-weighted prediction error: novelty, unexpectedness, motion, loudness, and brightness all involve prediction error on sensory data. Importance, urgency and goal or task relevancy suggest salience involves *value*. In the free-energy framework, value *is* precision (Friston, 2009), implying that valued predictions are high-precision predictions, which exert more influence over the representations that get expressed, and consequently, over actions.

Structure Building Codelets and Attention Codelets. In the LIDA model, structure building codelets build new nodes, links, and other representations in the preconscious Workspace. If such new representations win the competition for consciousness and are broadcast, then they are learned into any appropriate memory. Here, intimately integrated within the mechanism of PC-CLA, I am essentially proposing several general structure building codelets that propose and reinforce links. Structural links connecting hidden units in neighboring hierarchical levels serve to implement the part of the memory storing co-occurrence patterns, and are used when computing top-down predictions. Dynamic links connecting hidden units in the same hierarchical level are the foundations for temporal predictions. Finally, if PC-CLA incorporates hyperparameter (synaptic gain) optimization, then the optimization could be performed by attention codelets, concerned solely with the activity of prediction error computing units, and performing the update

operation previously discussed (Ch. 4, “Hyperparameter Optimization via Gain Control”) to estimate precision. Previously in LIDA, attention codelets did not have such general concerns.

Conclusions

This chapter has suggested a common, general-purpose algorithm for current representations, memory, and learning on high-dimensional data streams for the systems-level LIDA cognitive architectures. In particular, I proposed a predictive coding extension to the HTM Cortical Learning Algorithms, termed PC-CLA, which is proposed as a foundational building block for the systems-level LIDA cognitive architecture. PC-CLA fleshes out LIDA’s internal representations, memory, learning and attentional processes, and takes an initial step towards the comprehensive use of distributed and probabilistic (uncertain) representation throughout the architecture. PC-CLA further extends the Cortical Learning Algorithms to support hierarchy.

Several tests of PC-CLA were presented to bolster the claims of the algorithm’s abilities. Two tests further explored the performance of temporal pooling portion of the CLA, which has not seen much in the way of published reports. In particular I studied how well the temporal pooler predicts the next pattern of active columns in time. Subsequent tests of multiple hierarchical levels showed 1) the cost of translating messages between regions, given my particular choice of receptive fields, 2) the difference between processing and learning from prediction error versus the original pattern, and 3) the effects of a cortical region receiving top-down predictions from a higher region in a two-level network. The work here additionally extends the LIDA software framework to support high dimensional pattern recognition (see Chapter 6).

Finally, I discussed the implications of PC-CLA as a principled algorithm underlying the LIDA architecture.

Future Work. While I discussed the representation of statistical uncertainty in the theoretical introduction in Chapter 4, I have tabled any implementation of it in this dissertation. Adding uncertainty to the algorithm would require precision units to be added to the algorithm, which would estimate the uncertainty in information sources based on accumulated prediction error and then, based on this estimate, weight these prediction error signals.

A second avenue of future work involves replacing binary representations with scalar ones. For instance Snaider (2012) has found that integer vectors of limited range (e.g., 0–15) constitute a useful tradeoff between the expressiveness of scalar representation and the computational efficiency of discrete (e.g., Boolean) representations. This added expressiveness may be particular helpful since PC-CLA intrinsically requires multiple representations to be integrated into a single one, since a single cortical region's state is a concomitant of both a bottom-up signal, a top-down prediction, and an internal temporal prediction.

Since PC-CLA makes few assumptions about the type of sensory data it receives there is the promise of it having applicability to a range of data streams including auditory, visual, haptic, etc. Much further research into such applications is required to validate the algorithm's generality. With regards to the implementation of PC-CLA, some future work might take aim at rewriting the code so that cortical region processes can run in parallel on separate threads as opposed to running synchronously as they currently do.

There are several open issues regarding PC-CLA's use as a building block for the LIDA model, including how PC-CLA would incorporate Sensory-Motor Memory, the memory for motor plans that specifies actuator execution routines. However, there are reasons to believe such research would be fruitful (Friston, 2011). Additionally if PC-CLA is used to implement a systems-level LIDA agent, a methodology for building in motivations to bias particular patterns might be required (Friston et al., 2009).

6 Implementations

In this chapter, I describe the key algorithms and data structures of the computational implementations of the conceptual constructs presented so far in this dissertation.

Motivational Extension to LIDA Software Framework

In this section I describe my implementation of the motivational extension to the LIDA software framework. For the basics of the LIDA framework, see Chapter 2 "The LIDA Software Framework."

Adding Incentive Salience. A primary component of the motivational extension to the LIDA framework is the addition of base-level incentive salience and current incentive salience to both node and link data structures. Since incentive salience functions similarly to activation, the implementation of incentive salience mirrors that of activation; however incentive salience make take on a value in the range [–1, 1] whereas, previously in the framework, activation values were limited to [0, 1]. Previously in the LIDA framework, current activation and base-level activation were specified by several methods of the `Activatible` and `Learnable` interfaces, respectively. To obtain activation, various other framework elements, such as nodes, links, and codelets, extend these interfaces. Here, I update these two interfaces by adding new methods similar to the activation-related ones for incentive salience (Figure 48 upper boxes). With these updates, it is also possible for activation and incentive salience to take on values in the [–1, 1].

A technicality of adding this new value to the LIDA framework arises because `Node` and `Link` objects are copied as they move from module to module. This necessitates that incentive salience also be copied. This, in turn, requires that `NodeImpl` and `PamNodeImpl` override their `updateNodeValues(Node)` method and

`LinkImpl` and `PamLinkImpl` override their `updateLinkValues(Link)` method to ensure incentive salience is copied when nodes or links are copied.

The Valenceable, FeelingNode, and FeelingLink Interfaces. The second main component of the motivational extension concerns feeling nodes and feeling links. To implement these I first define the `Valenceable` interface so that framework elements may have a valence sign. Two new interfaces, `FeelingNode` and `FeelingLink`, extend `Valenceable` as well as the existing `Node` and `Link` interfaces. Figure 48 depicts this for `FeelingNode` only, but `FeelingLink` is implemented analogously.

Note that while, conceptually, the motivation extension does not suggest incentive salience for feeling nodes, the computational implementation inherits the incentive-salience-related methods since `FeelingNode` extends `Node` (and thus `Activatible`). Nonetheless, this is not an issue since the computational implementation never references the incentive salience values for feeling nodes so they are ignored.

Motivation Perceptual Associative Memory (PAM) Module. The `MotivationPam` class extends from the LIDA framework's default PAM implementation, `PerceptualAssociativeMemoryImpl`. It adds the module parameters given in Table 13. Also, a specific initializer, `MotivationPamInitializer`, extending the LIDA framework's existing `BasicPamInitializer`, is required to initialize the nodes. This initializer parses the "nodes" parameter as specified in Table 13.

Figure 49 summarizes the `MotivationPam` module in a UML class diagram. Particularly notable is the `isOverPerceptThreshold()` method, which takes into

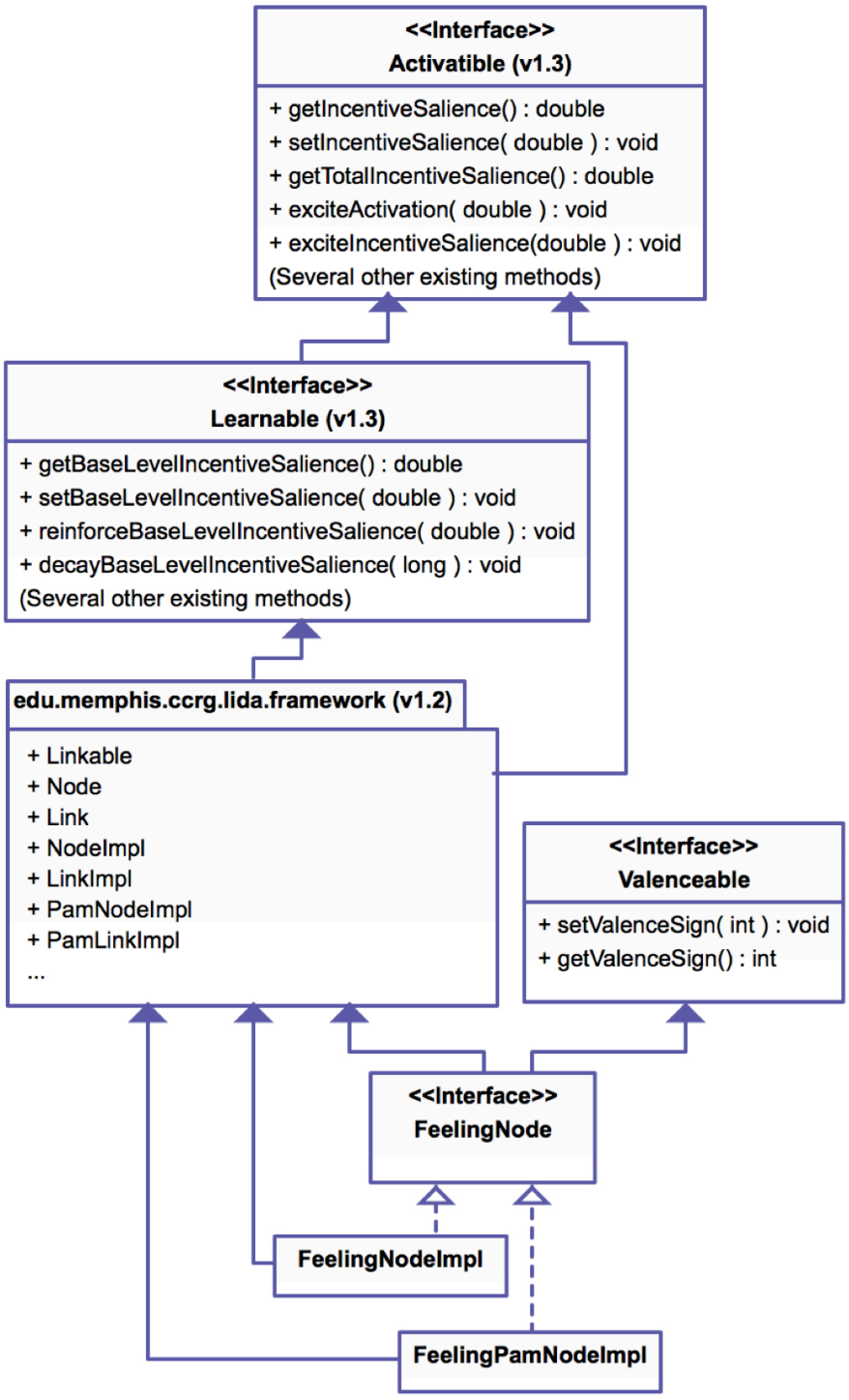


Figure 48. The UML diagram of part of the motivational extension to the LIDA software framework. Filled arrows denote an inheritance relationship. Open arrows imply an implementation-of relationship. While not shown, a similar implementation pattern is used for the `FeelingLink`, `FeelingLinkImpl`, and `FeelingPamLinkImpl` types.

Table 13. The module parameters introduced by the `MotivationPam` module.

Parameter Name	Explanation	Example Value
pam.perceptmapping.*	The * signifies wildcard. Each unique parameter beginning with this name specifies a mapping for PAM content that enters the percept: 1) type of object being mapped (node or link), 2) data type of element in PAM, 3) data type object is changed to when it enters the percept. This feature allows multiple node types to occur in PAM.	node, FeelingPamNodeImpl, FeelingNodeImpl
pam.learningRate	A parameter controlling the rate of all types of learning in PAM including of base-level activation and incentive salience.	0.2
pam.discountRate	A parameter controlling the rate of temporal difference learning.	0.1
nodes	The value of this parameter is sequence of node definitions, each separated by ‘,’. Each definition specifies a node’s label, initial base-level activation, name in the framework’s factory, optionally a valence sign, and, optionally, the type of feeling, drive or not.	thirst-feeling:0.1: FeelingPamNodeImpl: negative:drive or ball:0.5:PamNodeImpl

account both activation and incentive salience, and the `propagateActivation(PamLinkable, PamLink, double)` method, which adds support for current incentive salience passing.

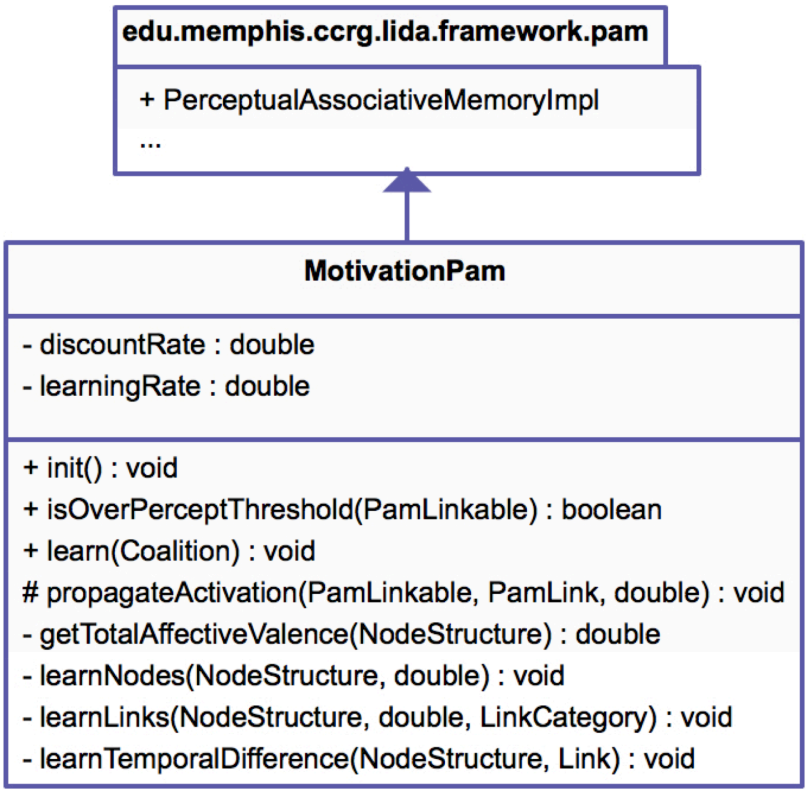


Figure 49. The UML class diagram for the `MotivationPam` class.

Workspace. The Workspace class was modified to support the usage of nonstandard `Node` and `Link` types; namely, the `FeelingNode` and `FeelingLink` types. Secondly, an temporal-link-building algorithm was built into the Workspace as a method. This method triggers whenever a new event node is added to the Workspace's Current Situational Model. This method attempts to build new temporal links from the previous conscious event in the Conscious Contents Queue (CCQ) to an event in the Current Situational Model (CSM). The pseudocode of this algorithm can be found in Appendix B.

Additionally, a `FeelingStructureBuildingCodelet` was added to the Workspace, which continually attempts to build new links from feeling nodes in the CSM to the strongest co-occurring event node in the CSM. Finally, for an attention codelet, the LIDA framework's `DefaultAttentionCodelet` was used, which generically forms coalitions from sufficiently active Workspace content. Note that no coalitions are created in the absence of CSM content. This is important because if no broadcast occurs then the Conscious Contents Queue does not advance. To remedy this, a periodic decay (every 1,000 ticks) was added to the CCQ that removes the oldest queue position.

Global Workspace. The standard LIDA framework `GlobalWorkspaceImpl` class is used, but the way the coalition class, `CoalitionImpl`, computes its activation is changed. Coalition activation, $a_{coalition}$, is still based on the base-level activation of the attention codelet creating the coalition and the average of the salience of each node and link in the coalition[1]. However, `Node` and `Link` salience is now the sum of the node or link's total activation and the absolute value of its total incentive salience.

$$a_{coaltion} = b_{attention-codelet} \cdot \sum_{i=1}^{n} i_{total-activation} + \left| i_{total-incentive-salience} \right| / n$$

In this section I have described the main changes to the LIDA framework required to implement the proposed motivation model. Chiefly, this extension allows nonstandard node and link objects to be used as a part of the common representational currency and adds base-level and current incentive salience to the `Learnable` and `Activatible` interfaces respectively.

[1] The conceptual LIDA model includes two additional factors in the calculation of coalition activation: 1) degree of intersection of the attention codelet's concern with the coalition's content, and 2) a general modulatory activation of the attention codelet module (Madl & Franklin, 2012). These factors were not used for this simple experimental replication and will not be further explained here.

2D-CLA and PC-CLA with the LIDA Software Framework

In this section I describe my implementation of 2D-CLA and PC-CLA as a software framework built on top of the existent LIDA software framework. For a time complexity analysis of the algorithm, see Appendix C.

Cortical Region Setup. The PC-CLA framework uses a factory class called `CorticalRegionFactory` to help create `CorticalRegionImpl` instances. This factory implements the factory design pattern and stores specifications of the parameters that define a `CorticalRegion`. Each of these specifications is implemented as a `CorticalRegionDef` (definition) object. To properly initialize an instance of `CorticalRegionImpl` with the LIDA software framework, one must declare a framework module using this class in the agent XML file. This module declaration must specify the `CorticalRegionInitializer` class as its initializer class. This initializer checks for the particular following module parameters: 1) "corticalRegionDefName," which specifies the name of the `CorticalRegionDef` used by the `CorticalRegionFactory`. If the name is new, a new definition is created and added to the factory; otherwise the factory uses the existing definition. In either case, the `CorticalRegionFactory` uses the `CorticalRegionDef` with this name to help build the `CorticalRegionImpl`. Note that this allows definitions to be reused across multiple `CorticalRegionImpl` instances (and XML declarations). 2) "backgroundTaskType" specifies the factory name (as defined in the factory data XML file) of the `FrameworkTask` implementing the processes of the cortical region. (This task specification is intentionally different from the standard way of initializing tasks in the LIDA framework. Typically tasks can be declared in the "initialtasks" tag of

module declaration.) 3) Finally, the "sourceModuleName" parameter specifies the module name providing the input to the cortical region being defined.

Object-Oriented Design. First I will describe the design of the implementation from a high level (Figure 50). I define a `CorticalRegion` interface specifying the basic operations for cortical regions and conforming to the LIDA framework by extending the framework's `FrameworkModule` interface. The central `CorticalRegionImpl` class implements `CorticalRegion` and `CorticalRegionBottomUpSource`. The latter is implemented because we want a cortical region to possibly be the source of input to a higher cortical region, and the separation of this interface allows other classes to implement `CorticalRegionBottomUpSource`. This implementation of inter-module communication is shaped by the decision to implement each cortical region's processes as a serial cycle. In an asynchronous implementation of `CorticalRegion` a listener design pattern might instead be used to define the communication. Next, I provide the details of the `CorticalRegionImpl` class.

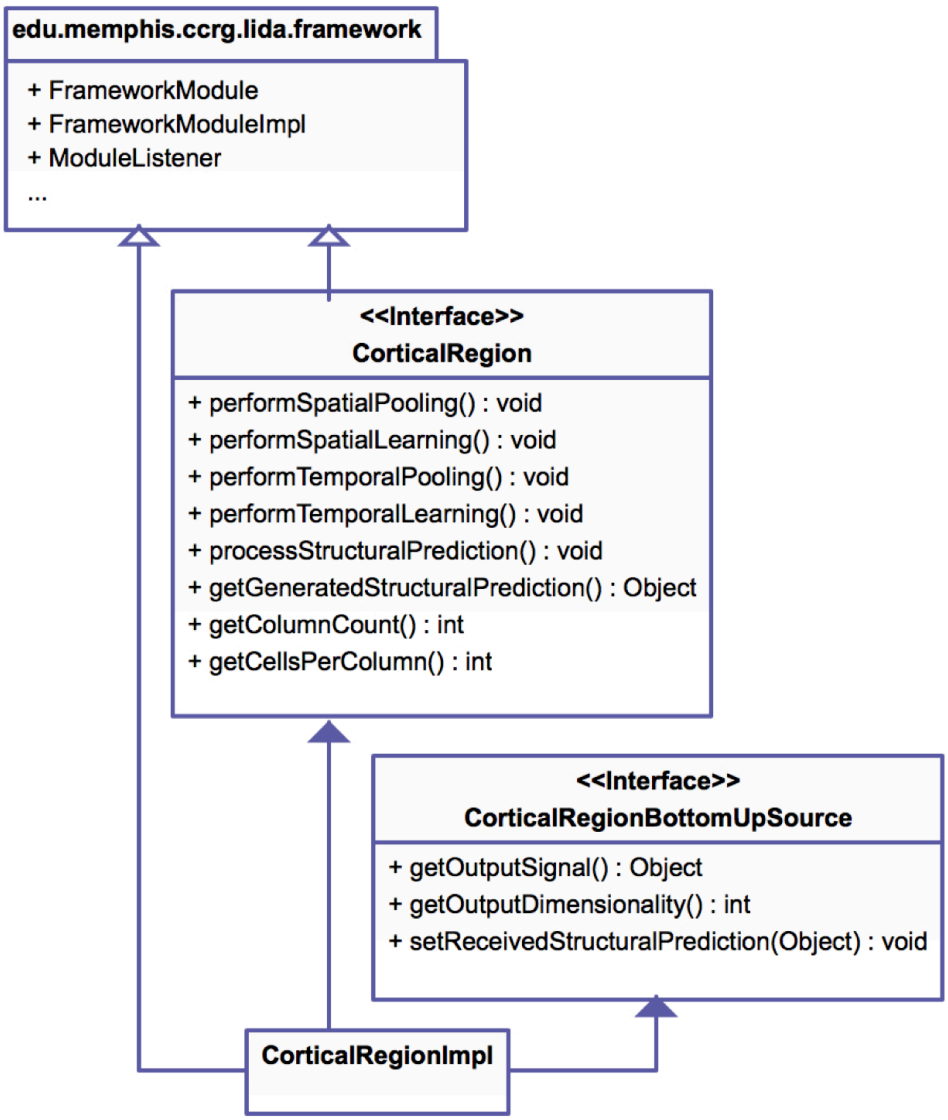


Figure 50. High-level view of the CLA implementation using the LIDA framework.

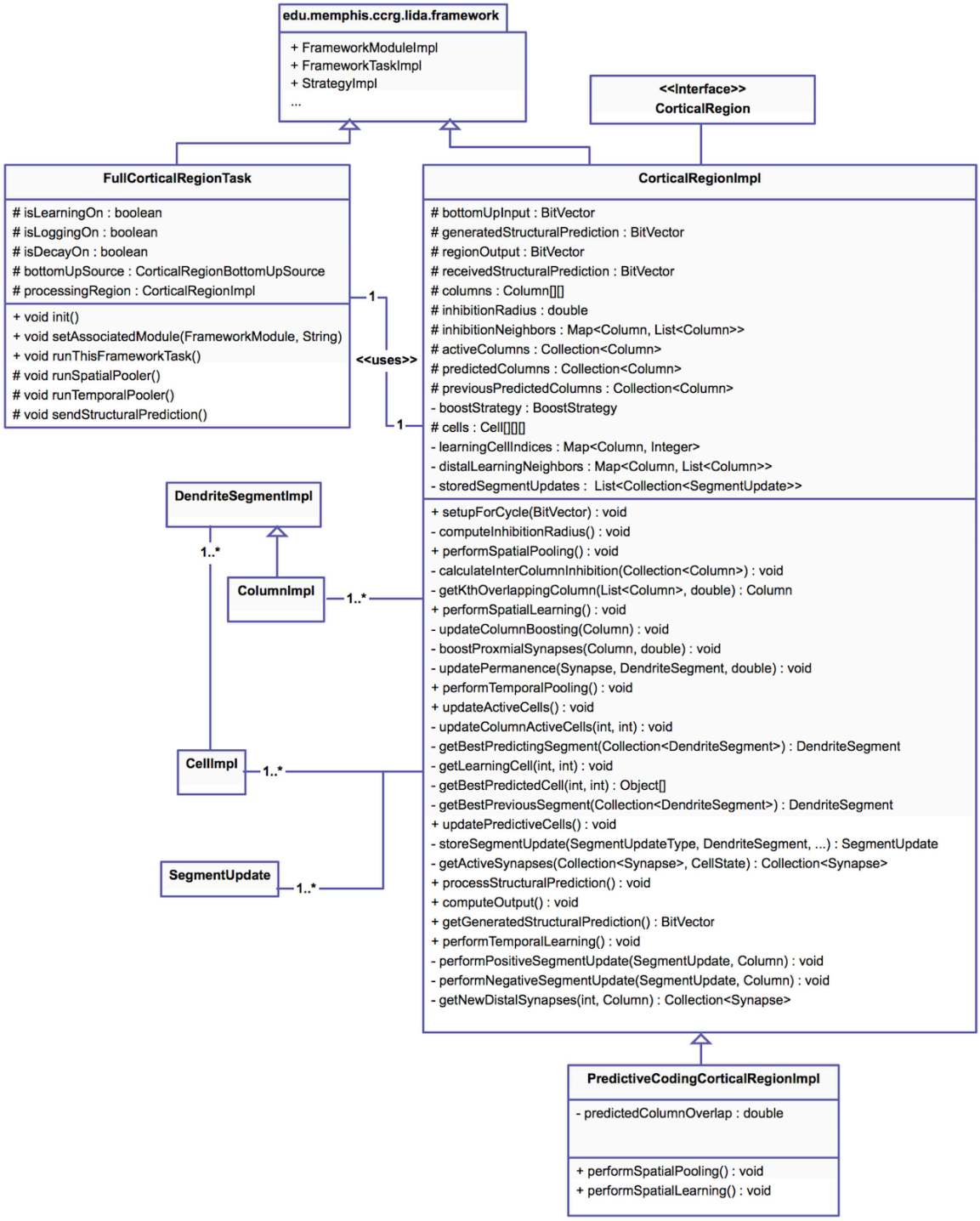


Figure 51. A detailed view of the CLA implementation using the LIDA framework. For clarity, some of the more extraneous class members were omitted.

The `CorticalRegionImpl` class is the central part of the CLA implementation. The details of this class are diagrammed in Figure 51. `BitVector`, an implementation of a Boolean vector is used for the input and output of the class. Each `CorticalRegionImpl` has a 2D array of `Column` instances and a 3D array of `Cell` instances as well as a queue of stored `SegmentUpdate` instances. `Column` neighborhoods, specifically, the *inhibitionNeighbors* and *distalLearningNeighbors* class members are implemented using maps where the key is a column and the value is the column's neighbors. The `FullCorticalRegionTask` runs the main loop of the algorithm. While each column has one proximal dendrite segment, in implementation, I simply extend `DendriteSegmentImpl` to help implement `ColumnImpl`. Predictive coding is implementing by a relatively simple extension of `CorticalRegionImpl` called `PredictiveCodingCorticalRegionImpl`.

TaskManager. The TaskManager was extended to be able to run a new main class after the framework is shutdown. In particular, I used this to run a program to parse the log file after each run of the framework. A new parameter, "postExecutionClass" was added to the `TaskManager`, which loads the class with specific canonical name (package name plus class name).

Another feature that was added is the "shutdown tick" functionality. This allows the user to specify a tick at which the `TaskManager` will automatically shutdown the entire framework application. The shutdown tick is set by specifying a parameter named "shutdownTick," having an integer value, in the `TaskManager` XML declaration.

Logging Analysis. One way I assessed the performance of the CLA implementation was using the standard Java logger. I added logging statements to record variable values after

each processing cycle of the algorithm. To aid this effort I used an enumeration called `ClaMeasures` to tag the log. Then, using the post execution class functionality describe in the last section, I used a simple class, `ProcessLogFile`, to load the log file by a run of the CLA framework application. Then, using some methods in the `DataUtils` class, I separated the logs of each measure into its own file. From here the stand-alone GUI class `DataVisualizer` can be used to view individual `ClaMeasures` as a time series and along with the series' average.

CLA GUI. To help visualize the application's runtime state, I created two GUI "panels," which are custom "windows" that can be added to the LIDA framework GUI. These GUI classes extend the LIDA framework's `GuiPanelImpl`. The first such panel, `CorticalRegionVisualizer`, displays various aspects of the `CorticalRegionImpl` with which it is associated. The possible aspects are defined by the `CorticalRegionAspect` enumeration, and include the active columns, currently predicted columns, columns' *boost* values, and current and previous cell states. The second GUI panel was a simple rendering of the Boolean input sent to the cortical region.

Conclusions

This chapter presents the software implementations of the motivational extensions to LIDA as well as the 2D-CLA and PC-CLA algorithms presented in this dissertation. Both implementations employ several software design patterns, and were designed for compatibility with the existing LIDA software framework.

7 Conclusions

Summary of Work

In Chapter 3, I explored how to implement the agenda of an autonomous agent in the systems-level LIDA cognitive architecture. To this end, I presented a motivation extension to LIDA covering a range of motivation-related concepts, including feelings, affective valence, incentive salience, emotion, appraisal, reinforcement learning, and model-free and model-based learning. This extension adds new faculties to the LIDA model including drive feelings, affective valence, incentive salience, and temporal difference learning. I discussed the role of motivations in a single LIDA cognitive cycle, as well as over multiple cycles. As an initial step in validating this motivational extension, I presented a computational implementation of a LIDA-based agent that replicates the results of an existing "reinforcer devaluation" experiment, which tested the agent's ability to learn, and later update, the reward predicting attributes of stimuli that drive its behavior.

In Chapter 4, I identified several guiding principles on the nature of perceptual representation, perceptual inference, and the associated learning processes. Guided by these principles, in particular, the free-energy principle, I suggested combining ideas from the computational HTM Cortical Learning Algorithms (CLA) and the mathematical Generalized Filtering. As an initial step in this direction, I presented my implementation of CLA, termed 2D-CLA, which modifies the receptive fields of the algorithm to have limited 2D receptive fields. Over several tests I explored some of the basic properties of the spatial pooling portion of 2D-CLA on artificial 2D patterns. I evaluated the sparse

distributed output of spatial pooling; namely, the active columns representation. Such testing provides new insights into this little-studied algorithm.

In Chapter 5, I proposed a general-purpose predictive coding extension to the HTM Cortical Learning Algorithms, termed PC-CLA, which is proposed as a foundational building block for the systems-level LIDA cognitive architecture. PC-CLA fleshes out LIDA's internal representations, memory, learning and attentional processes, and takes an initial step towards the comprehensive use of distributed and probabilistic (uncertain) representation throughout the architecture.

Summary of Contributions

In the Artificial Motivations Chapter (Chapter 3) I discuss how to extend the broad systems-level LIDA cognitive architecture to incorporate these various motivational concepts and mechanisms. The aspects implemented include feelings as motivators, valence, incentive salience, reinforcement learning, temporal difference learning, model-free control, and model-based control. These additions include changes to aspects of the LIDA model's memory, current representations, and notion of salience. To complement these additions to the LIDA conceptual model, this work additionally extends the software framework implementation of the LIDA model to include support for motivation. Specifically, this work 1) adds an implementation of feelings nodes having affective valence, 2) extends regular nodes allowing for current and base-level incentive salience, 3) extends links to pass incentive salience, and 4) adds learning algorithms related to incentive salience including temporal difference learning to LIDA's PAM. To begin validating the proposed motivational extension, a computational LIDA agent

replicates a psychological experiment testing motivational phenomena in biological agents.

Perception and pattern recognition are oft-ignored subjects in cognitive models, which are typically high-level. On the other hand, connectionist approaches to intelligent algorithms are not always concerned with algorithms suitable for systems-level cognitive architectures, which require online learning and unsupervised learning, and cannot have a human in the loop. In Chapters 4 and 5 I explore an algorithm capable of processing high dimensional inputs, which has relevance to cognitive architectures. In Chapter 4 I present a modification to the HTM Cortical Learning Algorithms (CLA), termed 2D-CLA. 2D-CLA adds limited 2D receptive fields to CLA to more closely model the way biological intelligence deals with high-dimensional "natural" data, e.g., audio and visual. I then describe several tests of the algorithm with relevance to both 2D-CLA and CLA. While the HTM theory has enjoyed some interest, to date there have been relatively few published works on the CLA. In Chapter 5 I present a novel predictive coding extension to the CLA algorithm, PC-CLA, which allows the algorithm to be repeated recursively in a hierarchy (McCall & Franklin, 2013). I then present additional tests of the CLA providing new insights into its properties and the effects of varying its parameters. I also report on initial tests of hierarchy with PC-CLA, which has not seen previous study.

In Chapter 6 I described the software implementations of the motivational extension to LIDA as well as the 2D-CLA and PC-CLA algorithms presented in this dissertation. Both implementations employ several software design patterns and have good compatibility with the existing LIDA software framework.

In summary, while my research has several goals, it has produced these particular contributions to Computer Science and Cognitive Science:

- Development of the generic parts of the LIDA software framework allowing for rapid and highly-customizable implementations of LIDA software agents (Ch. 2)
- Conceptual modeling of motivation in much of cognition using the LIDA model. Motivation is a fundamental, often-overlooked aspect of cognitive architectures. (Ch. 3)
- Extends the computational LIDA software framework to support the conceptual motivation model (Ch. 6)
- Part of the proposed work will validate the motivational extension of LIDA via the replication of an existing experimental result (Ch. 3)
- Conceptual modeling of high-dimensional perception in cognition necessary for intelligent agents with rich senses to succeed in complex environments (Ch. 4–5)
- Suggests a theoretical view (the free-energy principle) and accompanying algorithm (PC-CLA) that sees the processes of a systems-level cognitive architecture as performing general data assimilation based on patterns in time. This principled approach may help unify existing approaches by providing a core conceptual framework for mental processes (Ch. 4–5)
- Fleshes out a method for biologically plausible, limited, overlapping, 2D receptive fields for the Cortical Learning Algorithms (CLA) making the algorithm more suitable for processing visual stimuli (Ch. 4)
- Tests and evaluates some of the basic aspects of the CLA, previously undescribed by published work, including its sparse distributed representations, noise

robustness, parameters and their effects, “boosting” mechanism, and memory quality (Ch. 4–5)

- Extends the Cortical Learning Algorithms, by adding predictive coding, to support hierarchical versions of the algorithm. This allows for hierarchical decomposition, which is crucial in managing complexity in data analysis and pattern recognition (Ch. 5)
- Extends the LIDA software framework to support high-dimensional pattern recognition, which was previously unsupported. This allows LIDA agents to cope with sensory data streams of much greater complexity (Ch. 6)

Limitations

Concerning the work described in Chapter 3 on Artificial Motivations there are some limitations. This work did not attempt to model human-level emotions such as jealousy or shame, rather it provides lower-level mechanisms, which I hope will aid in explaining such phenomena. While I have described multi-cyclic deliberative decision-making conceptually and began approaching it computationally, deliberation, as a complex phenomenon, requires a more extensive study in a variety of settings, and not just simple experimental ones. While I theoretically proposed *drive* feeling nodes and describe their role in learning and their relation to current incentive salience, this proposal must be validated in more experimental replications and complex real-world tasks. Finally the motivational extension to LIDA should be evaluated in more complex unstructured domains.

For Chapters 4 and 5 on 2D-CLA and PC-CLA, I only conceptually described the representation of uncertainty in terms of precisions parameters as well as the estimation

of these parameters from prediction errors. Serious work is needed to best implement and validate such an extension. Additionally, the underlying representations of columns and cells in these algorithms are Boolean, as opposed to scalar, which might afford greater expressiveness. My computational implementation of these algorithms employs serial processes, but there is tremendous promise of runtime speedup if such processes were implemented in parallel and/or for GPUs. Lastly, the testing of these algorithms on a varied battery of real-world data, e.g., audio, visual, haptic, would help begin to support claims of the generality of the algorithm in the assimilation of data based on co-occurrence and sequence in time.

Future Directions

(In this section I summarize possible avenues for future work beyond the scope of this dissertation. I am *not* proposing to address the items below for the final version of this dissertation.)

In Chapter 3 in the section on the temporal difference learning of base-level incentive salience it was assumed that that the discount factor, γ, was fixed. However, changes in physiological state (e.g., stress) may serve to dynamically modulate this factor. Future work could, in general, explore the situations where emotions modulate the agent in a physiological way.

Another avenue of future research may involve the so-called, *eligibility traces*, which "trace" the credit assignment of temporal difference learning back further in time than the immediately previous event. Eligibility traces can be viewed as providing a short-term memory of multiple previous events so that several of these previous events may all be updated as each new observation arrives. Eligibility traces are usually

implemented by an exponentially decaying memory trace, with decay parameter λ (Barto, 2007).

In Chapters 4 and 5, I discussed the representation of statistical uncertainty theoretically; I have tabled the implementation of it in this dissertation. Adding uncertainty to the algorithm would require precision units to be added to the algorithm, which would estimate the uncertainty in information sources based on accumulated prediction error, and then, based on this estimate, weight these prediction error signals.

A second avenue of future work with the CLA involves replacing binary representations with scalar ones. For instance, Snaider (2012) has found that integer vectors of limited range (e.g., 0–15) constitute a useful tradeoff between the expressiveness of scalar representation and the computational efficiency of Boolean representations. This added expressiveness may be particular helpful since PC-CLA intrinsically requires multiple representations to be integrated into a single one, since a single cortical region's state is a concomitant of 1) a bottom-up signal, 2) a top-down prediction, and 3) an internal temporal prediction.

Since PC-CLA makes few assumptions about the type of sensory data it receives there is the promise of it having applicability to a range of data streams including auditory, visual, haptic, etc. Much further research and testing of the general efficacy of the algorithm are required. While I developed tests of the degradation of spatial memory and the noise robustness of the spatial pooling process, analogous tests are still needed for temporal memory and the temporal pooler. Finally, with some "deep" learning networks employing up to ten hierarchical levels (Ye, 2013), more than two levels should be tested with PC-CLA.

With regards to the implementation of PC-CLA, some future work might take aim at rewriting the code so that cortical region processes can run in parallel on separate threads as opposed to running synchronously as they currently do.

There are several open issues regarding PC-CLA's use as a building block for the LIDA model, including how PC-CLA would incorporate LIDA's Sensory-Motor Memory, the memory for motor plans that specifies actuator execution routines. However, there are reasons to believe such research would be fruitful (Friston, 2011). Additionally if PC-CLA is used to implement a systems-level LIDA agent, a methodology for building in motivations to bias particular patterns would be required (Friston et al., 2009). This would require implementing such constructs as feeling nodes and incentive salience using distributed connectionist representation, and not atomic ones as was done in Chapter 3.

Bibliography

Alvarado, N., Adams, S., & Burbeck, S. (2002). The Role of Emotion in an Architecture of Mind. *IBM Research*.

Arel, I., Rose, D., & Coop, R. (2009). DeSTIN : A Scalable Deep Learning Architecture with Application to High-Dimensional Robust Pattern Recognition. In *AAAI Workshop on Biologically Inspired Cognitive Architectures* (pp. 1150–1157).

Baars, B. J. (1988). *A Cognitive Theory of Consciousness*. Cambridge: Cambridge University Press.

Baars, B. J., & Franklin, S. (2003). How conscious experience and working memory interact. *Trends in Cognitive Science*, *7*, 166–172.

Baars, B. J., Franklin, S., & Ramsøy, T. (2013). Global workspace dynamics: Cortical "binding and propagation" enables conscious contents. *Frontiers in Psychology*, *4*. doi:10.3389/fpsyg.2013.00200

Bach, J. (2003). The MicroPsi Agent Architecture The MicroPsi architecture. In *Proceedings of ICCM5 International Conference on Cognitive Modeling Bamberg Germany* (Vol. 1, pp. 15–20). Citeseer. doi:10.1002/jsfa.1939

Bach, J. (2009). *Principles of Synthetic Intelligence: Psi: An Architecture of Motivated Cognition*. (F. E. Ritter, Ed.). Oxford: Oxford University Press.

Bach, J. (2012). A framework for emergent emotions, based on motivation and cognitive modulators. *International Journal of Synthetic Emotions*, *3*, 43–63. doi:10.4018/jse.2012010104

Bach, J. (2012). Modeling Motivation and the Emergence of Affect in a Cognitive Agent. In P. Wang & B. Goertzel (Eds.), *Theoretical Foundations of Artificial General Intelligence* (Vol. 4, pp. 241–263). Paris, France: Atlantis Press.

Bach, J., Goertzel, B., & Iklé, M. (Eds.). (2012). *Artificial General Intelligence: 5th International Conference*. Oxford, UK: Springer.

Baddeley, A., Conway, M., & Aggleton, J. (2001). *Episodic Memory*. Oxford: Oxford University Press.

Bar, M. (2009). The proactive brain: memory for predictions. *Phil. Trans. of the R. Soc. of London B: Biological Sciences*, *364*(1521), 1235–43. doi:10.1098/rstb.2008.0310

Bar, M., Aminoff, E., Mason, M., & Fenske, M. (2007). The units of thought. *Hippocampus*, *17*(6), 420–428. doi:10.1002/hipo.20287

Barsalou, L. W. (1999). Perceptual symbol systems. *Behavioral and Brain Sciences*, *22*, 577–609.

Barto, A. (2007). Temporal Difference Learning. *Scholarpedia*, *2*(11), 1604.

Bedny, M., Pascual-Leone, A., Dodell-Feder, D., Fedorenko, E., & Saxe, R. (2011). Language processing in the occipital cortex of congenitally blind adults. *Proceedings of the National Academy of Sciences of the United States of America*, *108*(11), 4429–34. doi:10.1073/pnas.1014818108

Belavkin, R. V. (2001). The Role of Emotion in Problem Solving. In *Symposium on Emotion, Cognition, and Affective Computing* (pp. 49–57). Heslington, York, UK.

Berridge, K. C., & Aldridge, J. W. (2008). Decision utility, the brain, and the pursuit of hedonic goals. *Social Cognition*, *26*(5), 621–646. doi:10.1521/soco.2008.26.5.621

Berridge, K. C., & Kringelbach, M. L. (2008). Affective neuroscience of pleasure: reward in humans and animals. *Psychopharmacology*, *199*(3), 457–480. doi:10.1007/s00213-008-1099-6

Berridge, K. C., & Robinson, T. E. (1998). What is the role of dopamine in reward: hedonic impact, reward learning, or incentive salience? *Brain Research. Brain Research Reviews*, *28*(3), 309–69.

Bindra, D. (1978). How adaptive behavior is produced: A perceptual-motivational alternative to response reinforcements. *Behavioral and Brain Sciences*, *1*, 41–91. doi:10.1017/S0140525X00059380

Bishop, Chris & Nasrabadi, N. (2006). *Pattern Recognition and Machine Learning* (1st ed., p. 740). New York, NY: Springer.

Bitzer, S., & Kiebel, S. J. (2012). Recognizing recurrent neural networks (rRNN): Bayesian inference for recurrent neural networks. *Biological Cybernetics*, *106*(4–5), 201–17. doi:10.1007/s00422-012-0490-x

Bogacz, R. (2007). Optimal decision-making theories: linking neurobiology with behaviour. *Trends in Cognitive Sciences*, *11*(3), 118–25. doi:10.1016/j.tics.2006.12.006

Bogacz, R., & Gurney, K. (2007). The basal ganglia and cortex implement optimal decision making between alternative actions. *Neural Computation*, *19*(2), 442–77. doi:10.1162/neco.2007.19.2.442

Breazeal (Ferrell), C. (1998). A Motivational System for Regulating Human-Robot Interaction. In *AAAI* (pp. 118–125). Madison, Wisconsin, USA.

Buxhoeveden, D., & Casanova, M. (2002). The minicolumn hypothesis in neuroscience. *Brain*, *125*(5), 935–951. doi:10.1093/brain/awf110

Camras, L. A. (2011). Differentiation, dynamical integration and functional emotional development. *Emotion Review*, *3*(2), 138–146.

Cañamero, L. (1997). Modeling motivations and emotions as a basis for intelligent behavior. In *Proceedings of the First International Conference on Autonomous Agents.* Marina Del Rey, CA, USA.

Cañamero, L. D. (2003). Designing Emotions for Activity Selection in Autonomous Agents. In R. Trappl, P. Petta, & S. Payr (Eds.), *Emotions in Humans and Artifacts* (pp. 115–148). Cambridge, MA: MIT Press.

Cannon, W. B. (1927). The James-Lange theory of emotions: A critical examination and an alternative theory. *The American Journal of Psychology*, *39*(1/4), 106–124.

Cannon, W. B. (1929). Organization for physiological homeostasis. *Physiology Review*, *9*, 399–431.

Chalmers, D. J. (1996). *The Conscious Mind*. Oxford: Oxford University Press.

Chaumon, M., Schwartz, D., & Tallon-Baudry, C. (2009). Unconscious learning versus visual perception: dissociable roles for gamma oscillations revealed in MEG. *Journal of Cognitive Neuroscience*, *21*(12), 2287–2299.

Cochran, R. E., Lee, F. J., & Chown, E. (2006). Modeling Emotion: Arousal's Impact on memory. In *Proceedings of the 28th Annual Conference of the Cognitive Science Society (pp. 1133-1138) . Vancouver, British Columbia, Canada* (pp. 1133–1138). Vancouver, British Columbia, Canada.

D'Mello, S. K., Ramamurthy, U., Negatu, A., & Franklin, S. (2006). A Procedural Learning Mechanism for Novel Skill Acquisition. In T. Kovacs & J. A. R. Marshall (Eds.), *Adaptation in Artificial and Biological Systems, AISB'06* (pp. 184–185). Bristol, England: Society for the Study of Artificial Intelligence and the Simulation of Behaviour.

Damasio, A. (2003). *Looking for Spinoza: Joy, Sorrow, and the Feeling Brain* (1st ed.). New York, NY: Mariner Books.

Damasio, A. R. (1999). *The Feeling of What Happens*. New York: Harcourt Brace.

Danks, D., Griffiths, T. L., & Tenenbaum, J. B. (2003). Dynamical causal learning. *Advances in Neural Information Processing Systems 15* (pp. 67-74). Cambridge, MA.: The MIT Press.

Davidson, R. J., Maxwell, J. S., & Shackma, A. J. (2004). The privileged status of emotion in the brain. *PNAS*, *101*, 11915–11916.

Davis, D. N., & Lewis, S. C. (2004). Affect and Affordance: Architectures without Emotion. *Architectures for Modeling Emotion: Cross-Disciplinary Foundations: Papers from the 2004 Spring Symposium*, 25–32.

Daw, N. D., Niv, Y., & Dayan, P. (2005). Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience*, *8*(12), 1704–1711.

De Garis, H., & Goertzel, B. (2009). Report on the First Conference on Artificial General Intelligence (AGI-08). *AI Magazine*, *30*(1), 115–116.

Dehaene, S., & Changeux, J.-P. (2011). Experimental and theoretical approaches to conscious processing. *Neuron*, *70*(2), 200–27. doi:10.1016/j.neuron.2011.03.018

Dehaene, S., Changeux, J.-P., Naccache, L., Sackur, J., & Sergent, C. (2006). Conscious, preconscious, and subliminal processing: a testable taxonomy. *Trends in Cognitive Sciences*, *10*, 204–211.

Dehaene, S., Naccache, L., H, G. L. C., Koechlin, E., Mueller, M., Dehaene-Lambertz, G., … Le Bihan, D. (1998). Imaging unconscious semantic priming. *Nature*, *395*, 597–600.

Derdikman, D., & Moser, M. B. (2010). A dual role for hippocampal replay. *Neuron*, *65*(5), 582–584. doi:S0896-6273(10)00139-X [pii] 10.1016/j.neuron.2010.02.022

Desimone, R., & Duncan, J. (1995). Neural mechanisms of selective visual attention. *Annual Reviews in Neuroscience*, *18*, 193–222.

Diener, E. (1999). Introduction to the Special Section on the structure of emotion. *Journal of Personality and Social Psychology*, *76*(5), 803–804.

Dijkstra, T. M. H., Schöner, G., & Gielen, C. C. A. M. (1994). Temporal stability of the action-perception cycle for postural control in a moving visual environment. *Experimental Brain Research*, *97*(3), 477–486.

Dörner, D., & Hille, K. (1995). Artificial Souls: Motivated Emotional Robots. In *IEEE International Conference on Systems, Man, and Cybernetics; Intelligent Systems for the 21st Century.* (pp. 3828–3832). Vancouver, British Columbia, Canada.

Douglas, R. J., & Martin, K. A. C. (2004). Neuronal circuits of the neocortex. *Annual Review of Neuroscience*, *27*, 419–51. doi:10.1146/annurev.neuro.27.070203.144152

Drescher, G. L. (1991). *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. Cambridge, Mass.: MIT Press.

Efron, B., & Morris, C. (1972). Stein's estimation rule and its competitors — an empirical Bayes approach. *Journal of the American Statistical Association*, *68*, 117–130.

Ekman, P., Sorenson, E. R., & Friesen, W. V. (1969). Pan-cultural elements in facial displays of emotion. *Science*, *164*, 86–88.

Ernst, M. O., & Banks, M. S. (2002). Humans integrate visual and haptic information in a statistically optimal fashion. *Nature*, *415*(6870), 429–33. doi:10.1038/415429a

Faghihi, U., & Franklin, S. (2014). A Computational Model of Causal Learning in a Cognitive Agent.

Faghihi, U., Fournier-Viger, P., & Nkambou, R. (2012). A computational model for causal learning in cognitive agents. *Knowledge-Based Systems*, *30*, 48–56.

Faghihi, U., McCall, R., & Franklin, S. (2012). A computational model of attentional learning in a cognitive agent. *Biologically Inspired Cognitive Architectures*, 25–36. doi:10.1016/j.bica.2012.07.003

Faghihi, U., Nkambou, R., Poirier, P., & Fournier-Viger, P. (2009). Emotional Learning and a Combined Centralist-Peripheralist Based Architecture for a More Efficient Cognitive Agent. In *7th IEEE International Conference on Industrial Technology (ICIT 2009)*.

Feldman, H., & Friston, K. J. (2010). Attention, Uncertainty, and Free-Energy. *Frontiers in Human Neuroscience*, *4*(December), 23.

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

Fellous, J.-M. (2004). From Human Emotions to Robot Emotions. In *American Association for Artificial Intelligence Spring Symposium 3/2004*. Stanford University, Palo Alto CA, USA.

Franklin, S. (1995). *Artificial Minds*. Cambridge, MA: MIT Press.

Franklin, S. (2000). Deliberation and Voluntary Action in "Conscious" Software Agents. *Neural Network World*, *10*, 505–521.

Franklin, S. (2003). IDA: A Conscious Artifact? *Journal of Consciousness Studies*, *10*, 47–66.

Franklin, S. (2005). Evolutionary Pressures and a Stable World for Animals and Robots: A Commentary on Merker. *Consciousness and Cognition*, *14*, 115–118.

Franklin, S., & Baars, B. (2010). Two Varieties of Unconscious Processes. In E. Perry, D. Collerton, H. Ashton, & F. LeBeau (Eds.), *New Horizons in the Neuuroscience of Consciousness* (pp. 91–102). Amsterdam: John Benjamin.

Franklin, S., Baars, B. J., Ramamurthy, U., & Ventura, M. (2005). The Role of Consciousness in Memory. *Brains, Minds and Media*, *1*(1), 1–38.

Franklin, S., & Graesser, A. C. (1997). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Intelligent Agents III* (pp. 21–35). Berlin: Springer Verlag.

Franklin, S., Madl, T., D'Mello, S. K., & Snaider, J. (2013). LIDA: A Systems-level Architecture for Cognition, Emotion, and Learning. *IEEE Transactions on Autonomous Mental Development*.

Franklin, S., & Patterson, F. G. J. (2006). The LIDA Architecture: Adding New Modes of Learning to an Intelligent, Autonomous, Software Agent. In *IDPT-2006 Proceedings (Integrated Design and Process Technology)*. Society for Design and Process Science.

Franklin, S., & Ramamurthy, U. (2006). Motivations, Values and Emotions: Three sides of the same coin. In *Proceedings of the Sixth International Workshop on Epigenetic Robotics* (pp. 41–48). Paris, France: Lund University Cognitive Studies.

Franklin, S., Strain, S., McCall, R., & Baars, B. (2013). Conceptual Commitments of the LIDA Model of Cognition. *Journal of Artificial General Intelligence*, *4*(2), 1–22. doi:10.2478/jagi-2013-0002

Franklin, S., Strain, S., Snaider, J., McCall, R., & Faghihi, U. (2012). Global Workspace Theory, its LIDA model and the underlying neuroscience. *Biologically Inspired Cognitive Architectures*, *1*, 32–43. doi:10.1016/j.bica.2012.04.001

Freeman, W. J. (1999). *How Brains Make Up Their Minds*. London: Weidenfeld & Nicolson General.

Freeman, W. J. (2002). The limbic action-perception cycle controlling goal-directed animal behavior. *Neural Networks*, *3*, 2249–2254.

Friston, K. (2005). A theory of cortical responses. *Phil. Trans. R. Soc. Lond. B*, *360*(1456), 815–836. doi:10.1098/rstb.2005.1622

Friston, K. (2009). The free-energy principle: a rough guide to the brain? *Trends in Cognitive Science*, *13*(7), 293–301. doi:10.1016/j.tics.2009.04.005

Friston, K. (2010). The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, *11*(2), 127–138. doi:10.1038/nrn2787

Friston, K. (2011). What is optimal about motor control? *Neuron*, *72*(3), 488–498.

Friston, K. (2012). Prediction, perception and agency. *International Journal of Psychophysiology*, *83*(2), 248–52. doi:10.1016/j.ijpsycho.2011.11.014

Friston, K., & Kiebel, S. (2009). Predictive coding under the free-energy principle. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, *364*(1521), 1211–21. doi:10.1098/rstb.2008.0300

Friston, K., & Kiebel, S. (2011). Predictive coding under the free-energy principle. In M. Bar (Ed.), *Predictions In the Brain: Using Our Past to Generate a Future.* (pp. 231–246). Oxford, UK: Oxford University Press.

Friston, K., Daunizeau, J., & Kiebel, S. (2009). Reinforcement Learning or Active Inference? *PLoS ONE*, *4*(7), e6421. doi:10.1371/journal.pone.0006421

Friston, K., Stephan, K., Li, B., & Daunizeau, J. (2010). Generalised Filtering. *Mathematical Problems in Engineering, 2010*.

Friston, K., Trujillo-Barreto, N., & Daunizeau, J. (2008). DEM: A Variational Treatment of Dynamic Systems. *NeuroImage*, *41*(3), 849–885. doi:10.1016/j.neuroimage.2008.02.054

Fum, D., & Stocco, A. (2004). Memory, Emotion, and Rationality: An ACT-R interpretation for Gambling Task Results. In *Sixth International Conference on Cognitive Modeling* (pp. 106–111). Pittsburgh, PA, USA.

Fuster, J. M. (2004). Upper processing stages of the perception–action cycle. *Trends in Cognitive Science*, *8*(4), 143–145.

Fuster, J. M. (2006). The cognit: a network model of cortical representation. *International Journal of Psychophysiology*, *60*(2), 125–132. doi:10.1016/j.ijpsycho.2005.12.015

Fuster, J. M. (2007). Cortical Memory. *Scholarpedia*, *2*(4), 1644.

Gabbot, P., & Somogyi, P. (1986). Quantitative distribution of GABA-immunoreactive neurons in the visual cortex (area 17) of the cat. *Experimental Brain Research*, *61*, 323–331.

Gallagher, M., McMahan, R. W., & Schoenbaum, G. (1999). Orbitofrontal Cortex and Representation of Incentive Value in Associative Learning. *The Journal of Neuroscience*, *19*(15), 6610–6614.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. *Design* (Vol. 206, p. 395). doi:10.1093/carcin/bgs084

George, D., & Hawkins, J. (2009). Towards a mathematical theory of cortical micro-circuits. *PLoS Computational Biology*, *5*(10), e1000532. doi:10.1371/journal.pcbi.1000532

Gmytrasiewicz, P. J., & Lisetti, C. L. (2002). Emotions and Personality in Agent Design and Modeling. In S. Parsons, P. Gmytrasiewicz, & M. J. Wooldridge (Eds.), *Game Theory and Decision Theory in Agent-Based Systems* (pp. 81–95). Spring US. doi:10.1007/978-1-4615-1107-6_5

Goertzel, B., & Pennachin, C. (2007). *Artificial General Intelligence*. Berlin: Springer.

Hawkins, J., Ahmad, S., & Dubinsky, D. (2011). *Hierarchical Temporal Memory including HTM Cortical Learning Algorithms* (Vol. 32). Retrieved from http://numenta.org/

Hawkins, J., & Blakeslee, S. (2004). *On Intelligence*. New York, NY: Henry Holt.

Hawkins, J., George, D., & Niemasik, J. (2009). Sequence memory for prediction, inference and behaviour. *Phil. Trans. R. Soc. B*, *364*(1521), 1203–1209. doi:10.1098/rstb.2008.0322

Hebb, D. O. (1949). *Organization of behavior*. John Wiley.

Hinton, G., McClelland, J., & Rumelhart, D. E. (1986). Distributed Representations. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations* (pp. 77–109).

Hintzman, D. L. (2011). Research Strategy in the Study of Memory: Fads, Fallacies, and the Search for the "Coordinates of Truth." *Perspectives on Psychological Science*, *6*(3), 253–271. doi:10.1177/1745691611406924

Hollerman, J., & Schultz, W. (1998). Dopamine Neruons Report an Error in the Temproal Prediction of Reward during Learning. *Nature Neuroscience*, *1*, 304–309.

Houk, J. C., Adams, J. L., & Barto, A. G. (1995). A model of how the basal ganglia generate and use neural signals that predict reinforcement. In J. C. Houk, J. L. Davis, & D. G. Beiser (Eds.), *Models of Information Processing in the Basal Ganglia* (Vol. 13, pp. 249–270). Cambridge, MA: MIT Press.

Huang, Y., & Rao, R. P. N. (2011). Predictive coding. *WIREs Cogn Sci*. doi:10.1002/wcs.142

Huys, Q. J. M., Eshel, N., Nions, E. O., Sheridan, L., Dayan, P., Jonathan, P., … Roiser, J. P. (2012). Bonsai Trees in Your Head: How the Pavlovian System Sculpts Goal-directed Choices by Pruning Decision Trees. *PLoS Computational Biology*, *8*(3), e1002410. doi:10.1371/journal.pcbi.1002410

Izard, C. (1993). Four Systems for Emotion Activation: Cognitive and Noncognitive Processes. *Psychological Review*, *100*, 68–90.

James, W. (1884). II. — What is an Emotion? *Mind*, *34*, 188–205.

James, W. (1890). *The Principles of Psychology*. Cambridge, MA: Harvard University Press.

Johnston, V. S. (1999). *Why We Feel: The Science of Human Emotions*. Reading, MA: Perseus Books.

Kahneman, D. (2003). Maps of Bounded Rationality: Psychology for Behavioral Economics. *The American Economic Review*, *93*(5), 1449–1475.

Kahneman, D. (2011). *Thinking, Fast and Slow*. New York, NY: Farrar, Straus, and Giroux.

Kahneman, D., Wakker, P., & Sarin, R. (1997). Back to Bentham? Explorations of Experienced Utility. *The Quarterly Journal of Economics*, *112*, 375–405.

Kalis, A., Kaiser, S., & Mojzisch, A. (2013). Why we should talk about option generation in decision-making research. *Frontiers in Psychology*, *4*. doi:10.3389/fpsyg.2013.00555

Kalman, R. E., & Bucy, R. S. (1961). New Results in Linear Filtering and Prediction Theory. *Journal of Basic Engineering*, *83*(1), 95–108. doi:10.1115/1.3658902

Kanerva, P. (1988). *Sparse Distributed Memory*. Cambridge, MA: MIT Press.

Kass, R. E., & Steffey, D. (1989). Approximate Bayesian inference in conditionally independent hierarchical models (parametric empirical Bayes models). *Journal of the American Statistical Association*, *84*(407), 717–726.

Keller, L., & Ho, J. (1988). Decision problem structuring: generating options. *IEEE Transactions on Systems, Man, and Cybernetics*, *18*(5), 715–728.

Klein, G., Wolf, S., Militello, L., & Zsambok, C. (1995). Characteristics of Skilled Option Generation in Chess. *Organizational Behavior and Human Decision Processes*, *62*(1), 63–69.

Knill, D. C., & Pouget, A. (2004). The Bayesian brain: the role of uncertainty in neural coding and computation. *Trends in Neurosciences*, *27*(12), 712–719. doi:10.1016/j.tins.2004.10.007

Körding, K. P., & Wolpert, D. M. (2004). Bayesian integration in sensorimotor learning. *Nature*, *427*(6971), 244–7. doi:10.1038/nature02169

Kringelbach, M. L., & Berridge, K. C. (2009). Towards a functional neuroanatomy of pleasure and happiness. *Trends in Cognitive Sciences*, *13*(11), 479–487. doi:10.1016/j.tics.2009.08.006

Kruschke, J. K. (2011). Models of attentional learning. In E. M. Pothos & A. J. Wills (Eds.), *Formal Approaches in Categorization* (pp. 120–152). Cambridge, U.K: Cambridge University Press.

Laird, J. E., Newell, A., & Rosenbloom, P. (1987). SOAR: An Architecture for General Intelligence. *Artificial Intelligence*, *33*, 1–64.

Lang, P. J., & Davis, M. (2006). Emotion, motivation, and the brain: reflex foundations in animal and human research. *Progress in Brain Research*, *156*, 3–29. doi:10.1016/S0079-6123(06)56001-7

Langley, P., Laird, J. E., & Rogers, S. (2009). Cognitive Architectures: Research Issues and Challenges. *Cognitive Systems Research*, *10*(2), 141–160. doi:10.1016/j.cogsys.2006.07.004

Lazarus, R. (1991). *Emotion and Adaptation*. New York, NY: Oxford University Press.

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, *3361*.

LeDoux, J. (2000). Emotion Circuits in the Brain. *Annual Review of Neuroscience*, *23*, 155–184.

LeDoux, J. E. (2006). Emotional Memory: In Search of Systems and Synapsesa. *Annals of the New York Academy of Sciences*, *702*(1), 149–157.

Lee, T. S., & Mumford, D. (2003). Hierarchical Bayesian Inference in the Visual Cortex. *Journal of the Optical Society of America A*, *20*(7), 1434. doi:10.1364/JOSAA.20.001434

Lee-Johnson, C. P., & Carnegie, D. A. (2009). Robotic Emotions: Navigation with Feeling. In J. Vallverdú & D. Casacuberta (Eds.), *Handbook of Research on Synthetic Emotions and Sociable Robotics* (pp. 88–117). IGI Global.

Lewin, K. (1951). *Field theory in social science: selected theoretical papers*. New York: Harper & Row.

Liddell, B. J., Brown, K. J., Kemp, A. H., Barton, M. J., Das, P., Peduto, A., … Williams, L. M. (2005). A direct brainstem-amygdala-cortical "alarm" system for subliminal signals of fear. *NeuroImage*, *24*(1), 235–43. doi:10.1016/j.neuroimage.2004.08.016

Linsker, R. (1990). Perceptual neural organization: some approaches based on network models and information theory. *Annual Review of Neuroscience*, *13*, 257–81. doi:10.1146/annurev.ne.13.030190.001353

Lucantonio, F., Stalnaker, T., Shaham, Y., Niv, Y., & Schoenbaum, G. (2012). The impact of orbitofrontal dysfunction on cocaine addiction. *Nature Neuroscience*, *15*(3). doi:10.1038/nn.3014

MacDonald, K. (2008). Effortful Control, Explicit Processing and the Regulation of Human Evolved Predispositions. *Psychological Review*, *115*(4), 012–1031.

Madl, T., Baars, B. J., & Franklin, S. (2011). The Timing of the Cognitive Cycle. *PLoS ONE*, *6*(4), e14803. doi:10.1371/journal.pone.0014803

Madl, T., & Franklin, S. (2012). A LIDA-based Model of the Attentional Blink. In *ICCM 2012 The 11th International Conference on Cognitive Modeling*.

Madl, T., Franklin, S., Chen, K., & Trappl, R. (2013). Spatial Working Memory in the LIDA Cognitive Architecture. In R. West & T. Stewart (Eds.), *Proceedings of the 12th International Conference on Cognitive Modeling* (pp. 384–390). Ottawa, Canada: Carleton University.

Maes, P. (1989). How to do the right thing. *Connection Science*, *1*, 291–323.

Marieb, E. N., & Hoehn, K. (2007). *Human Anatomy & Physiology* (7th ed.). San Francisco, CA: Pearson Benjamin Cummings.

Marinier, R. P., & Laird, J. E. (2008). Emotion-Driven Reinforcement Learning. In *Cognitive Science* (pp. 115–120).

Marinier, R. P., Laird, J. E., & Lewis, R. L. (2009). A computational unification of cognitive behavior and emotion. *Cognitive Systems Research*, *10*(1), 48–69. doi:10.1016/j.cogsys.2008.03.004

Martinez-Trujillo, J. C., & Treue, S. (2004). Feature-based attention increases the selectivity of population responses in primate visual cortex. *Current Biology*, *14*, 744–751.

McCall, R., Franklin, S., & Friedlander, D. (2010). Grounded Event-Based and Modal Representations for Objects, Relations, Beliefs, Etc. In *FLAIRS-23*. Daytona Beach, FL, USA.

McCall, R., Franklin, S., Snaider, J., & Faghihi, U. (2014). Artificial Motivations for Cognitive Software Agents.

McCall, R. J., & Franklin, S. (2013). Cortical Learning Algorithms with Predictive Coding for a Systems-Level Cognitive Architecture. In *Second Annual Conference on Advances in Cognitive Systems* (pp. 149–166). Baltimore, MD, USA.

Merker, B. (2005). The liabilities of mobility: A selection pressure for the transition to consciousness in animal evolution. *Consciousness and Cognition*, *14*, 89–114.

Métin, C., & Frost, D. O. (1989). Visual responses of neurons in somatosensory cortex of hamsters with experimentally induced retinal projections to somatosensory thalamus. *Proceedings of the National Academy of Sciences of the United States of America*, *86*(1), 357–61.

Meyer, H. S., Egger, R., Guest, J. M., Foerster, R., Reissl, S., & Oberlaender, M. (2013). Cellular organization of cortical barrel columns is whisker-specific. *Proceedings of the National Academy of Sciences*, (201312691). doi:10.1073/pnas.1312691110

Montague, P., Dayan, P., & Sejnowski, T. J. (1996). A Framework for Mesencephalic Dopamine Systems Based on Predictive Hebbian Learning. *The Journal of Neuroscience*, *16*(5), 1936–1947.

Mountcastle, V. (1978). An Organizing Principle for Cerebral Function: The Unit Model and the Distributed System. In G. M. Edelman & V. B. Mountcastle (Eds.), *The Mindful Brain* (pp. 7–50). Cambridge, MA: MIT Press.

Mumford, D. (1992). On the computational architecture of the neocortex. *Biological Cybernetics*, *66*, 241–251. doi:10.1007/BF00198477

Nakayama, Y., Yamagata, T., Tanji, J., & Hoshi, E. (2008). Transformation of a virtual action plan into a motor plan in the premotor cortex. *The Journal of Neuroscience*, *28*(41), 10287–10297.

Negatu, A. (2006). *Cognitively Inspired Decision Making for Software Agents: Integrated Mechanisms for Action Selection, Expectation, Automatization and Non-Routine Problem Solving*. The University of Memphis, Memphis, TN, USA.

Neisser, U. (1976). *Cognition and Reality: Principles and Implications of Cognitive Psychology*. San Francisco: W. H. Freeman.

Newell, A. (1973). You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. In W. G. Chase (Ed.), *Visual information processing*. New York: Academic Press.

Ng, A. (2013). Error Metrics for Skewed Classes [Video lecture]. Retrieved from https://class.coursera.org/ml/

O'Doherty, J. P., Dayan, P., Friston, K., Critchley, H., & Dolan, R. J. (2003). Temporal difference models and reward-related learning in the human brain. *Neuron*, *38*(2), 329–37.

O'Reilly, R. C. (1998). Six principles for biologically based computational models of cortical cognition. *Trends in Cognitive Sciences*, *2*(11), 455–62.

O'Reilly, R. C., & Munakata, Y. (2000). *Computational Explorations in Cognitive Neuroscience*. Cambridge, MA: MIT Press.

Olshausen, B. A., & Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Letters to Nature*, *381*, 607–609.

Pasquereau, B., Nadjar, A., Arkadir, D., Bezard, E., Goillandeau, M., Bioulac, B., … Boraud, T. (2007). Shaping of motor responses by incentive values through the basal ganglia. *The Journal of Neuroscience*, *27*, 1176–1183. doi:10.1523/JNEUROSCI.3745-06.2007

Pessoa, L. (2008). On the relationship between emotion and cognition. *Nature Reviews Neuroscience*, *9*, 148–158.

Phelps, E. A. (2006). Emotion and Cognition: Insights from Studies of the Human Amygdala. *Annual Review of Psychology*, *57*(1), 27–53. doi:10.1146/annurev.psych.56.091103.070234

Phillips, W. A., & Singer, W. (1997). In search of common foundations for cortical computation. *Behavioral and Brain Sciences*, *20*, 657–683; discussion 683–722.

Picard, R. (1997). *Affective Computing*. Cambridge, MA: The MIT Press.

Picard, R. (2003). Affective computing: challenges. *International Journal of Human-Computer Studies*, *59*(1-2), 55–64. doi:10.1016/S1071-5819(03)00052-1

Plate, T. (2003). *Holographic Reduced Representation: Distributed Representation of Cognitive Structure*. Stanford, CA: CSLI Publications.

Purves, D., Augustine, G., Fitzpatrick, D., Katz, L. C., LaMantia, A.-S., McNamara, J. O., & Williams, S. M. (2001). The Formation of Topographic Maps. In

Neuroscience. Sunderland, MA: Sinauer Associates Inc. Retrieved from https://www.ncbi.nlm.nih.gov/books/NBK10904/

Purves, D., Brannon, E., Cabez, R., Huettel, S. A., Labar, K., Platt, M., & Woldorff, M. (2008). *Principles of Cognitive Neuroscience*. *Addiction* (1st ed., Vol. 1). Sunderland, MA: Sinauer Associates Inc. doi:10.1086/592635

Puskorius, G. V., & Feldkamp, L. A. (1991). Decoupled extended Kalman filter training of feedforward layered network. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN '91)* (pp. 771–777).

Raab, M., de Oliveira, R. F., & Heinen, T. (2009). How do people perceive and generate options? *Progress in Brain Research*, *174*, 49–59.

Raichle, M. E., MacLeod, A. M., Snyder, A. Z., Powers, W. J., Gusnard, D. A., & Shulman, G. L. (2001). A Default Mode of Brain Function. *Proceedings of the National Academy of Sciences*, *98*(2), 676–682. doi:10.1073/pnas.98.2.676

Ramamurthy, U., & Franklin, S. (2011). Memory Systems for Cognitive Agents. In *Human Memory for Artificial Agents Symposium at the Artificial Intelligence and Simulation of Behavior Convention (AISB'11)* (pp. 35–40). University of York, UK.

Rao, R. P. (1999). An optimal estimation approach to visual perception and learning. *Vision Research*, *39*(11), 1963–89. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/10343783

Rao, R. P., & Ballard, D. H. (1999). Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, *2*(1), 79–87. doi:10.1038/4580

Rao, S. C., Rainer, G., & Miller, E. K. (1997). Integration of what and where in the primate prefrontal cortex. *Science*, *276*, 821–824.

Richard, J. M., & Berridge, K. C. (2011). Nucleus accumbens dopamine/glutamate interaction switches modes to generate desire versus dread: D1 for appetitive eating but D1 and D2 together for fear. *Journal of Neuroscience*, *31*(36), 12866–12879.

Riesenhuber, M., & Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature Neuroscience*, *2*(11), 1019–25. doi:10.1038/14819

Rinkus, G. J. (2010). A cortical sparse distributed coding model linking mini- and macrocolumn-scale functionality. *Frontiers in Neuroanatomy*, *4*(June), 17. doi:10.3389/fnana.2010.00017

Rockel, A. J., Hiorns, R. W., & Powell, T. P. (1980). The basic uniformity in structure of the neocortex. *Brain*, *103*, 221–244.

Roe, A. W., Pallas, S. L., Kwon, Y. H., & Sur, M. (1992). Visual projections routed to the auditory pathway in ferrets: receptive fields of visual neurons in primary auditory cortex. *The Journal of Neuroscience*, *12*(9), 3651–64. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/1527604

Roseman, I. J., & Smith, C. A. (2001). Appraisal theory: Overview, assumptions, varieties, controversies. In *Appraisal Theory: Overview, Assumptions, Varieties, Controversies*. New York, NY: Oxford University Press.

Rowe, J., Hughes, L., Eckstein, D., & Owen, A. M. (2008). Rule-selection and action-selection have a shared neuroanatomical basis in the human prefrontal and parietal cortex. *Cerebral Cortex*, *18*, 2275–2285. doi:10.1093/cercor/bhm249

Russell, S., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (Third., Vol. 3nd). Upper Saddle River, NJ: Prentice Hall.

Samsonovich, A. V. (2010). Toward a Unified Catalog of Implemented Cognitive Architectures. In A. V Samsonovich, K. R. Jóhannsdóttir, A. Chella, & B. Goertzel (Eds.), *Proceeding of the 2010 Conference on Biologically Inspired Cognitive Architectures* (pp. 195–244). Amsterdam: IOS Press.

Samsonovich, A. V., Jóhannsdóttir, K. R. J., Chella, A., & Goertzel, B. (Eds.). (2010). *Biologically Inspired Cognitive Architectures 2010 - Proceedings of the First Annual Meeting of the BICA Society* (Vol. 221). Amsterdam: IOS Press.

Schmidhuber, J., Thorisson, K. R., & Looks, M. (2011). Artificial General Intelligence , Proceedings. In *4th International Conference, AGI 2011* (p. 416). Mountain View, CA: Springer.

Schoenbaum, G., Takahashi, Y., Liu, T., & McDannald, M. (2011). Does the orbitofrontal cortex signal value? *Annals of the New York Academy of Sciences*, *1239*, 87–99.

Schultz, W., Dayan, P., & Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, *275*(5306), 1593–1599.

Serre, T., Kreiman, G., Kouh, M., Cadieu, C., Knoblich, U., & Poggio, T. (2007). A quantitative theory of immediate visual recognition. *Progress in Brain Research*, *165*, 33–56. doi:10.1016/S0079-6123(06)65004-8

Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., & Poggio, T. (2007). Robust Object Recognition with Cortex-Like Mechanisms. *IEEE Transations on Pattern Analysis and Machine Intelligence*, *29*(3), 411–426. doi:10.1109/TPAMI.2007.56

Shanahan, M. (2010). *Embodiment and the Inner Life*. Oxford: Oxford University Press.

Shariff, A. F., & Tracy, J. L. (2011). What Are Emotion Expressions For? *Current Directions in Psychological Science*, *20*(6), 395–399. doi:10.1177/0963721411424739

Shin, Y. K., Proctor, R. W., & Capaldi, E. J. (2010). A review of contemporary ideomotor theory. *Psychol Bull*, *136*(6), 943–974. doi:10.1037/a0020541

Skarda, C., & Freeman, W. J. (1987). How Brains Make Chaos in Order to Make Sense of the World. *Behavioral and Brain Sciences*, *10*, 161–195.

Sloman, A. (1998). Damasio, Descartes, Alarms and Meta-management. In *Proceedings Symposiumon Cognitive Agents: Modeling Human Cognition*. San Diego: IEEE.

Sloman, A. (1999). What Sort of Architecture is Required for a Human-like Agent? In M. Wooldridge & A. S. Rao (Eds.), *Foundations of Rational Agency* (pp. 35–52). Dordrecht, Netherlands: Kluwer Academic Publishers.

Sloman, A., & Croucher, M. (1981). Why robots will have emotions. In *7th Int. Joint Conference on AI (ICJAI)* (pp. 1–10). Vancouver, British Columbia, Canada.

Smith, K. S., Berridge, K. C., & Aldridge, J. W. (2011). Disentangling pleasure from incentive salience and learning signals in brain reward circuitry. *Proceedings of the National Academy of Sciences of the United States of America*, *108*(27), E255–64. doi:10.1073/pnas.1101920108

Snaider, J. (2012). *Integer Sparse Distributed Memory and Modular Composite Representation*. The University of Memphis.

Snaider, J., McCall, R., & Franklin, S. (2011). The LIDA Framework as a General Tool for AGI. *The Fourth Conference on Artificial General Intelligence Springer Lecture Notes in Artificial Intelligence*, *6830*, 133–142. doi:10.1007/978-3-642-22887-2_14

Snaider, J., McCall, R., & Franklin, S. (2012). Time production and representation in a conceptual and computational cognitive model. *Cognitive Systems Research*, *13*(1), 59–71. doi:10.1016/j.cogsys.2010.10.004

Squire, L. R., & Kandel, E. R. (2000). *Memory: From Mind to Molecules*. New York, NY: Henry Holt.

Sun, R. (2003). A Tutorial on CLARION 5.0. Retrieved November 11, 2012, from www.cogsci.rpi.edu/~rsun/sun.tutorial.pdf

Sun, R. (2009). Motivational representations within a computational cognitive architecture. *Cognitive Computation*, *1*(1), 91–103.

Sutton, R. S. (1988). Learning to Predict by the Method of Temporal Differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Swindale, N. V. (2008). Visual map. *Scholarpedia*, *3*(6), 4607.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., & Goodman, N. D. (2011). How to grow a mind: statistics, structure, and abstraction. *Science*, *331*(6022), 1279–85. doi:10.1126/science.1192788

Thompson, R. F., & Madigan, S. A. (2007). *Memory: The Key to Consciousness*. *Memory* (p. 280). Princeton University Press.

Thornton, J. R., Main, L., & Srbic, A. (2012). Fixed Frame Temporal Pooling. In M. Thielscher & D. Zang (Eds.), *AI 2012: Advances in Artificial Intelligence* (pp. 707–718). Sydney, Australia: Springer Berlin Heidelberg. doi:10.1007/978-3-642-35101-3_60

Thornton, J. R., & Srbic, A. (2013). Spatial Pooling for Greyscale Images. *International Journal of Machine Learning and Cybernetics*, *4*(3), 207–216. doi:10.1007/s13042-012-0087-7

Thornton, J., Srbic, A., Main, L., & Chitsaz, M. (2011). Augmented spatial pooling. In *AI 2011: Advances in Artificial Intelligence* (pp. 261–270). Springer Berlin Heidelberg.

Tindell, A. J., Smith, K. S., Berridge, K. C., & Aldridge, J. W. (2009). Dynamic computation of incentive salience: "wanting" what was never "liked". *The Journal of Neuroscience*, *29*(39), 12220–8. doi:10.1523/JNEUROSCI.2499-09.2009

Toates, F. (1986). *Motivational Systems*. Cambridge, U.K.: Cambridge University Press.

Van der Merwe, R., Doucet, A., de Freitas, N., & Wan, E. (n.d.). The unscented particle filter. In *NIPS* (pp. 584–590).

Van Rijsbergen, C. J. (1979). *Information Retrieval*. *Butterworths*. Retrieved from http://www.dcs.gla.ac.uk/Keith/Preface.html

Von Helmholtz, H. (2005). Concerning the Perceptions in General. In *Treatise on Physiological Optics* (3rd ed., Vol. 3). New York, NY: Courier Dover Publications.

Von Melchner, L., Pallas, S. L., & Sur, M. (2000). Visual behaviour mediated by retinal projections directed to the auditory pathway. *Nature*, *404*(6780), 871–6. doi:10.1038/35009102

Walls, C., & Breidenbach, R. (2005). *Spring in action*. Dreamtech Press.

Wang, P., Goertzel, B., & Franklin, S. (2008). *Artificial General Intelligence 2008*. Amsterdam: IOS Press.

Ward, P., Suss, J., Eccles, D., Williams, A., & Harris, K. (2011). Skill-based differences in option generation in a complex task: a verbal protocol analysis. *Cognitive Processing*, *12*(3), 289–300.

Westen, D. (1999). *Psychology: Mind, Brain, and Culture*. *J Wiley* (2nd ed.). Wiley.

Wilson, N. R., Sun, R., & Mathews, R. C. (2009). A motivationally-based simulation of performance degradation under pressure. *Neural Networks*, *22*(5-6), 502–8. doi:10.1016/j.neunet.2009.06.022

Wimmer, G. E., & Shohamy, D. (2012). Preference by association: how memory mechanisms in the hippocampus bias decisions. *Science*, *338*(6104), 270–3. doi:10.1126/science.1223252

Yerkes, R. M., & Dodson, J. D. (1908). The Relationship of Strength of Stimulus to Rapidity of Habit Formation. *Journal of Comparative Neurology and Psychology*, *18*, 459–482.

Yiend, J. (2010). The effects of emotion on attention: A review of attentional processing of emotional information. *Cognition and Emotion*, *24*(1), 3–47. doi:10.1080/02699930903205698

Zacks, J. M., Speer, N. K., Swallow, K. M., Braver, T. S., & Reynolds, J. R. (2007). Event Perception: A Mind–Brain Perspective. *Psychological Bulletin*, *133*(2), 273–293. doi:10.1037/0033-2909.133.2.273

Zacks, J. M., & Tversky, B. (2001). Event Structure in Perception and Conception. *Psychological Bulletin*, *127*(1), 3.

Zhang, J., Berridge, K. C., Tindell, A. J., Smith, K. S., & Aldridge, J. W. (2009). A Neural Computational Model of Incentive Salience. *PLoS Computational Biology*, *5*(7), 14. doi:10.1371/journal.pcbi.1000437

Zhu, J., & Thagard, P. (2002). Emotion and action. *Philosophical Psychology*, *15*, 19–36.

Appendix A – Author's Refereed Publications

McCall, R., & Franklin, S. (2013). Cortical Learning Algorithms with Predictive Coding for a Systems-Level Cognitive Architecture. *Proceedings of the Second Annual Conference on Advances in Cognitive Systems* (pp. 149–166). Baltimore, MD, USA.

Franklin, S., Strain, S., McCall, R., & Baars, B. (2013). Conceptual Commitments of the LIDA Model of Cognition. *Journal of Artificial General Intelligence, 4*(2), 1–22.

Faghihi, U., McCall, R., & Franklin, S. (2012). A Computational Model of Attentional Learning in a Cognitive Agent. *Biologically Inspired Cognitive Architectures*, *2*, 25–36.

Franklin, S., Strain, S., Snaider, J., McCall, R., & Faghihi, U. (2012). Global Workspace Theory, its LIDA Model and the Underlying Neuroscience. *Biologically Inspired Cognitive Architectures*, *1*, 32–43.

McCall, R. & Franklin, S. (2012). Meta Learning, Change of Internal Workings, and LIDA: A commentary on Thórisson and Helgasson's "Cognitive architectures and autonomy: A Comparative Review" [Peer commentary by R. McCall & S. Franklin]. *Journal of Artificial General Intelligence, 3*(2), 42–44.

Snaider, J., McCall, R., & Franklin, S. (2012). Time production and representation in a conceptual and computational cognitive model. *Cognitive Systems Research, 13*(1), 59–71.

Snaider, J., McCall, R., Franklin, S., (2011). The LIDA Framework as a General Tool for AGI. *The Proceedings of the Fourth Conference on Artificial General Intelligence (AGI-11)*.

McCall, R., Franklin, S., & Friedlander, D. (2010). Grounded Event-Based and Modal Representations for Objects, Relations, Beliefs, Etc. Paper presented at the FLAIRS-23, Daytona Beach, FL.

Snaider, J., McCall, R., & Franklin, S. (2009). Time Production and Representation in a Conceptual and Computational Cognitive Model. Paper presented at the AAAI Fall Symposium on Biologically Inspired Cognitive Architectures, Washington, DC.

Dale, R., Roche, J., Snyder, K., & McCall, R. (2008). Exploring Action Dynamics as an Index of Paired-Associate Learning. *PLoS ONE*, *3(3)*, e1728.

McCall, R., Franklin, S., Snaider, J., & Faghihi, U. (in preparation). Artificial Motivations for Cognitive Software Agents.

Appendix B – Selected Pseudocode

In this Appendix I present the pseudocode for a selection of the algorithms described in this Dissertation. The pseudocode is present as one or more functions, each taking a set of inputs and returning an output.

Motivational Agent. A critical aspect of the motivational agent is the ability to build novel temporal links between events having a temporal order. Here I present the pseudocode of the algorithm employed in the Workspace to build such links.

```
Function BuildTemporalLink(n, csmStructure, temporal)
Adds potential links to the current situational model based on temporal order of
conscious events
inputs:         n, A node that is a part of the current percept
                csmStructure, A node-and-link graph representing
                                the current situation
                temporal, Flag signifying a temporal link
local variables:    broadcastQueue, A queue of recent conscious broadcasts
                    latestBroadcast, Most recent conscious broadcast, a graph
                    source, Potential source of temporal link
                    sink, Potential sink of temporal link
latestBroadcast ← broadcastQueue.Pop()
sink ← n
// If latest broadcast already contains n then n assume n
// has been ongoing and that no temporal link should be created
if latestBroadcast.Contains(sink) = false then
        source ← GetFirstEvent(latestBroadcast, temporal)
        if source is not null then
                // Add source if not already present in CSM
                if csmStructure.Contains(source) = false then
                        csmStructure.AddNode(source)
                // Add link if it does not create a two-link cycle
                if csmStructure.Contains(sink, source, temporal) = false then
                        csmStructure.AddLink(source, sink, temporal)
```

2D-CLA. 2D-CLA adds limited 2D receptive fields to the CLA. The shape of the receptive fields is modeled using a bivariate Normal distribution. Here I present the pseudocode that generates these 2D receptive fields.

function AddProximalSynapses(*column*, *heightBound*, *widthBound*, *synapseFactor*, *rfSigma*, *connectionThreshold*)
Adds synapses to specified column connecting it to the input space. The receptive field's shaped is determined by a bivariate Normal distribution.

inputs: *column*, Proximal synapses will be added to this column
heightBound, Height of the square-shaped input space
widthBound, Width of the square-shaped input space
synapseFactor, Multiplicative constant used to calculate the number of synapses connected to a given input bit
rfSigma, Controls the dispersion of the receptive field
connectionThreshold, Threshold at which synapses are "connected"
local variables: *boundingRadius*, Radius of a circle that overestimates the extent of the area where Bivariate Normal distribution gives a positive output
boundingRadiusSquared, Square of *boundingRadius*
sigmaSquared, Square of *rfSigma*
twoSigmaSquared, Twice *sigmaSquared*
normalization, Normalization term of Normal distribution function
heightMin, Lower *y* bound of a square bounding the receptive field
heightMax, Upper *y* bound of a square bounding the receptive field
widthMin, Lower *x* bound of a square bounding the receptive field
widthMax, Upper *x* bound of a square bounding the receptive field
distanceSquared, Square of distance between an input dimension column's position
synapses, Number of synapses to be connected to a given input position
newSynapse, A single newly generated synapse

boundingRadius ← 2 * *rfSigma* + 1
boundingRadiusSquared ← Power(*boundingRadius*, 2)
sigmaSquared ← Power(*rfSigma*, 2)
twoSigmaSquared ← 2 * *sigmaSquared*
normalization ← *synapseFactor* / (*twoSigmaSquared* * *PI*)
// Calculate height and width bounds of a square bounding the circular receptive field
heightMin ← Floor(*column*.InputHeightPosition – *boundingRadius*)
if *heightMin* < 0
    heightMin ← 0
heightMax ← Floor(*column*.InputHeightPosition + *boundingRadius*)
if *heightMax* > *heightBound* – 1

```
    heightMax ← heightBound – 1
widthMin ← Floor(column.InputWidthPosition – boundingRadius)
if widthMin < 0
    widthMin ← 0
widthMax ← Floor(column.InputWidthPosition + boundingRadius)

if widthMax > widthBound – 1
    widthMax ← widthBound – 1
for i = heightMin to heightMax do
    for j = widthMin to widthMax do
        distanceSquared ← Power(i – heightPosition, 2) + Power(j – widthPosition, 2)
        if distanceSquared <= boundingRadiusSquared then
            synapses ← Round(normalization * Exponential(–distanceSquared /
                                                          twoSigmaSquared))
            for s = 0 to synapses do
                newSynapse ← AddPotentialSynapse(column, i, j)
                if newSynapse.Permanence >= connectionThreshold then
                    AddConnectedSynapse(column, newSynapse)
```

PC-CLA. In this section I present some pseudocode for the PC-CLA algorithm. I focus on the parts of algorithm unique to PC-CLA, or those parts that are modified from the original CLA. I start with those functions, a part of the spatial pooling portion of the algorithm, which were modified. The function ComputeOverlappingColumns is invoked first, followed immediately by ComputeActiveColumns, which uses GetKthColumn as a subroutine. Finally PerformSpatialLearning would be called by the main loop of the algorithm.

```
function ComputeOverlappingColumns(falseNegativeError, columns, predictedColumns,
                                   predictedColumnOverlap,
columnOverlapThreshold)
returns a set of columns whose receptive fields significantly overlap with active inputs

inputs:       falseNegativeError, Current unpredicted input activity
              columns, The cortical region's columns
              predictedColumnOverlap, Overlap score given to predicted columns
              columnOverlapThreshold, Threshold defining sufficient column overlap
local variables:  newBoost, New boost value for a column
                  activityScore, Synaptic activity of a particular column
                  overlappingColumns, Columns whose synaptic connections
                                      sufficiently overlap with active inputs
for each column in columns do
    newBoost ← 0
    if column.PotentialSynapseCount > 0 then
        activityScore ← 0
        if predictedColumns.Contains(column) then
            activityScore ← predictedColumnOverlap
        else
            for each synapse in column.ConnectedSynapses do
                if inputError[synapse.SourceInputIndex] = true then
                    activityScore++
            activityScore ← activityScore / column.PotentialSynapseCount
        if activityScore > columnOverlapThreshold then
            newBoost ← activityScore * column.Boost
        overlappingColumns.Add(column)
    column.BoostedOverlap ← newBoost
return overlappingColumns
```

function ComputeActiveColumns(*overlappingColumns*, *inhibitionRadius*, *columnCount*, *k*)
returns a set of active columns representing the current bottom-up input

inputs: *PI*, Ratio of a circle's circumference to its diameter
overlappingColumns, Columns whose synaptic connections sufficiently overlap with active inputs
inhibitionRadius, Average distance of a connected synapse from its column across all proximal synapses in the cortical region
columnCount, Number of columns in the cortical region
k, Sparsity parameter governing density of active columns per unit area
local variables: *inhibitionArea*, An estimate of the size of a column's receptive field
maxPossibleNeighbors, Largest possible inhibition neighborhood for a column
kthColumn, K-th most active column in a neighborhood
activeColumns, list of active columns

```
inhibitionArea ← PI * Power(inhibitionRadius, 2)
//Inhibition area could be larger than area of all columns for some parameters
maxPossibleNeighbors ← Min(inhibitionArea, columnCount)
for each column in overlappingColumns do
    kthColumn ← GetKthColumn(GetNeighbors(column), maxPossibleNeighbors, k)
    if kthColumn is not null then
        if column.BoostedOverlap >= kthColumn.BoostedOverlap then
            activeColumns.Add(column)
return activeColumns
```

function GetKthColumn(*neighbors*, *maxNeighbors*, *k*)
returns the column with *k*-th highest overlap among a specified collection

inputs: *neighbors*, List of columns neighboring each other
maxNeighbors, most possible neighbors a column could possibly have
k, Place of the returned column in terms of its boosted overlap
local variables: *kthColumn*, Column with *k*-th highest overlap
scaling, Measure of how large neighborhood is as compared to the maximumally sized neighborhood
scaledIndex, Index of *k*-th most active column, scaled for neighborhood size

```
kthColumn ← null
if maxNumNeighbors > 0 then
    scaling ← neighbors.Size / maxNumNeighbors
    // As neighborhood size decreases, less columns can be k-th highest or better
    scaledIndex ← Round(scaling * k) – 1
    if scaledIndex >= 0 and scaledIndex < neighbors.Size
        Sort(neighbors) //Sort by boosted overlap score
        kthColumn ← neighbors.Get(scaledIndex)
return kthColumn
```

function PerformSpatialLearning(*columns*, *falseNegativeError*, *falsePositiveError*, *proximalLearningIncrement*, *proximalLearningDecrement*)
Updates the proximal synapses of columns in response to bottom-up prediction error.

inputs: *columns*, Cortical region's columns
falseNegativeError, Current unpredicted input activity
falsePositiveError, Current falsely predicted input activity
proximalLearningIncrement, Positive change amount for proximal synapses
proximalLearningDecrement, Negative change amount for proximal synapses

```
for each column in columns do
    UpdateBoost(column)
    for each synapse in column.PotentialSynapses do
        if falseNegativeError[synapse.InputIndex] = true then
            UpdatePermanence(synapse, proximalLearningIncrement)
        if falsePositiveError[synapse.InputIndex] = true then
            UpdatePermanence(synapse, proximalLearningDecrement)
```

The subsequent functions are related to the inter-cortical-region message passing. The first describes how a cortical region generates a top-down prediction of its input. The second describes how a cortical region incorporates a top-down prediction into its state, and the third illustrates how a cortical region's output signal, for a higher cortical region, is produced.

function GetTopDownPrediction(*oneOrderPredictedColumns*, *inputCoverage*, *predictionThreshold*)
returns The cortical region's current top-down prediction of its bottom-up input

inputs: *oneOrderPredictedColumns*, Columns predicted for the next time step
inputCoverage, Array, over the input dimensions, containing the number of synaptic connections to each input dimension
predictionThreshold, Threshold determing whether top-down synaptic activity counts toward the top-down prediction
local variables: *predictionCounts*, Number of predictions for each input dimension
prediction, Boolean prediction of cortical region's next bottom-up input

```
for each column in oneOrderPredictedColumns do
    for each synapse in column.ConnectedSynapses do
        predictionCounts[synapse.SourceInputIndex]++
for i = 0 to predictionCounts.Length – 1 do
    if inputPredictions[i] / inputCoverage[i] > predictionThreshold then
        prediction[i] ← true
return prediction
```

function ProcessTopDownPrediction() incorporates a received top-down prediction with the cortical region's cell state

inputs: *tdPrediction*, Current top-down prediction received from higher region
cells, Cells of the cortical region as a 1D array
regionHeight, Height of the cortical region's column and cell space
regionWidth, Width of the cortical region's column and cell space
cellsPerColumn, Number of cells in each column
local variables: *heightOffset*, Offset for 3D to 1D array conversion

```
heightOffset = regionWidth * cellsPerColumn
for i = 0 to regionHeight – 1 do
    for j = 0 to regionWidth – 1 do
        for k = 0 to cellsPerColumn – 1 do
            index ← i * heightOffset + j * cellsPerColumn + k
            if tdPrediction[index] = true then
                cells[i][j][k].IsPredictedCurrently ← true
```

function ComputeOutput(*cells*, *regionHeight*, *regionWidth*, *cellsPerColumn*) **returns** the current output of a cortical region to its immediately hierarchically superior cortical region.

inputs: *cells*, Cells of the cortical region as a 1D array
regionHeight, Height of the cortical region's column and cell space
regionWidth, Width of the cortical region's column and cell space
cellsPerColumn, Number of cells in each column
local variables: *output*, The cortical region's boolean output
heightOffset, Offset for 3D to 1D array conversion

```
heightOffset ← regionWidth * cellsPerColumn
for i = 0 to regionHeight – 1 do
    for j = 0 to regionWidth – 1 do
        for k = 0 to cellsPerColumn – 1 do
            if cells[i][j][k].IsActiveCurrently or
             cells[i][j][k].IsPredictedCurrently then
                output[i * heightOffset + j * cellsPerColumn + k] ← true
return output
```

Appendix C – PC-CLA Complexity Analysis

Below, I provide an analysis of the time requirements of the PC-CLA algorithm. The definitions of the variables used are given in Table 14. I perform this analysis with some parameters involved. While the asymptotic runtime complexity does not change for these various parameter values, I present the work in terms of these parameters to get a better sense of the cost of the algorithm as a function of these parameters.

Initialization Phase

Function: SetupForCycle

Time Complexity: $O(cpn)$

Comment: Requires updating state of each cell.

Function: ComputeInhibitionRadius

Time Complexity: $O(s_p pn)$

Comment: Requires computing the average distance of each connected proximal synapse from its column's center.

Function: UpdateInhibitionRadius

Time Complexity: $O(r_i^2 pn)$

Comment: For each column (pn), requires computing the column's neighboring columns having size at most.

Phase Total: Sum of these functions' runtimes is and, under the condition $s_p > r_i^2$, this simplifies to $O(s_p pn)$.

Table 14. The parameters used in the complexity analysis of the PC-CLA algorithm.

Parameter	**Associated variable**	**Description**	**Example value**
Input size	n	The dimensionality of the input	529
Projection factor	p	Ratio of number of columns to the input size	4
Connected synapses per column	s_p	An upper bound on the number of connected synapses per column	50
Inhibition radius	r_i	Radius of the circle about a column that defines that column's inhibition neighborhood	5.2
Inhibition neighbors	*nbrs*	Number of columns within a circle of radius *inhibition radius*	85
Active column ratio	k	Ratio of active columns to total columns in a cortical region	0.02
Cells per column	c	Number of cells in each column	7
Dendrite segments per cell	*ds*	Number of unique distal dendrite segments associated with each cell	5
Maximum synapses per distal segment	s_d	Absolute limit on number of synapses per distal dendrite segment	32
Distal learning radius	r_d	Radius of a circle about the column where new distal synapses are being adding. Only cells within this circle can be selected as a new synapse's source.	7
Highest segment prediction order	o	Largest number of cycles in advance a distal dendrite segment will predict for future activity	4

Spatial Pooling Phase

Function: ComputeOverlappingColumns

Time Complexity: $O(s_p pn)$

Comment: Computes each column's boosted *overlap score*.

Function: ComputeActiveColumns

Time Complexity: $O(nbrs \cdot \log_2(nbrs) \cdot kpn)$

Comment: For each overlapping column, this function performs a merge sort, which has $O(n \log_2(n))$ time complexity.

Function: PerformSpatialLearning

Time Complexity: $O((2s_p + nbrs)pn)$

Comment: In the worst case, all synapses on all columns must be updated twice.

Phase Total: $O(dpn)$, $d = \max\{s_p, k(nbrs \log_2(nbrs)), 2s_p + nbrs\}$

Assuming the active columns ratio is near 0.02, i.e., $k = 0.02$, then the third argument wins the max operation over the second argument and $d = 2s_p + nbrs$. This gives us a total time complexity of $O((2s_p + nbrs)pn)$.

Temporal Pooling

Function: UpdateActiveCells

Time Complexity: $O((ds \cdot ck)(s_d pn))$

Comment: For each cell of each active column of which there are $ckpn$, we must check its distal dendrites segments and their synapses for various conditions. In the worst case this amounts to $s_d(2ds+1)ckpn$ operations. Dropping constants in the parenthesis and refactoring for comparison with other functions gives $O((ds \cdot ck)(s_d pn))$.

Function: UpdatePredictiveCells

Time Complexity: $O((ds \cdot c)(s_d pn))$

Comment: In the worst case, for every cell, cpn, we have to check all of its distal dendrite segments and their synapses for $s_d \cdot ds \cdot cpn$ operations. Factoring this term for easy comparison with the other functions gives $O((ds \cdot c)(s_d pn))$.

Function: PerformTemporalLearning

Time Complexity: $O(ock(s_d pn))$

Comment: The time complexity can be easily stated as $O(u \cdot s_d)$, where *u* is the number of segment updates processed on average during each cycle. Segment updates are queued in the functions UpdateActiveCells and UpdatePredictiveCells. The former will queue at most kpn updates to be performed in the same cycle. The latter will store as many updates as there are predicted cells, which I estimate to be about $kcpn$. Additionally, at a given cycle updates queued a variable number of cycles previously may require processing. Let *o* represent the highest order of segment update queued in the function UpdatePredictiveCells. We then have:

$$u = kpn + okcpn$$

$$u = (oc + 1)kpn$$

$$\Rightarrow O(s_d(oc + 1)kpn) = O(ock(s_d pn))$$

Phase Total: The first function loses out to the second because of the added *k* term, which is always less than 1.0. The third function also loses out to the second since *ok* will almost certainly be less than 1.0 as well. So, due to the function UpdatePredictiveCells, this phase has an overall time complexity of $O(s_d\, ds \cdot cpn)$

Message Passing Phase

Function: ComputeTopDownPrediction

Time Complexity: $O(s_p pn + n) = O((s_p p + 1)n)$

Comment: In the worst case we must check each proximal synapse on each column in the region counting up the number of times an input is predicted. Then a final pass over each input bit checks if its prediction activity meets a threshold.

Function: ProcessTopDownPrediction

Time Complexity: $O(cpn)$

Comment: In the worst case this involves updating the state of all cells in the region.

Function: ComputeOutput

Time Complexity: $O(cpn)$

Comment: Possibly involves all cells in the region.

Phase Total: Technically, it could be $O((s_p p + 1)n)$ or $O(cpn)$. However, since the cost of *cells per column*, *c*, grows much faster than s_p, it is unlikely that $c > s_p$. Furthermore, since $s_p > c$ in this implementation, I will assume that $O((s_p p + 1)n)$ dominates.

Algorithm Total

The temporal pooling phase dominates the others phases with a time complexity of $O(s_d\, ds\,(cpn))$. This casts the spotlight on the parameters, *c*, *ds*, and s_d. With high values for these parameters the cortical region has great capacity to learn temporal patterns with the drawback of greatly increasing the time complexity of the algorithm. To illustrate this total cost using the somewhat modest example values given in Table 14 we have a theoretical time complexity of $O(4480n)$.