Preface
10 Years Later

It has been over 10 years since writing this document, and nearly that long since the project was last active. I think there were good ideas here. Many of them I still find interesting, a few of them I'd like to change, and some I think may be useful to others. In any case, I thought it might be helpful to post it online along with a few brief comments, at least as a historical record of what I was thinking in 2009.

This project always involved a certain amount of risk. It attempted to achieve two challenging goals simultaneously: 1) a reasonably accurate model of brain structure and function, and 2) a general purpose learning system to be implemented in software. It seemed clear from the start that, due to the ambitiousness of these endeavors, both might suffer. Indeed, I think they did suffer to an extent, but I'm not unhappy with the result. I expected it to be a work in progress... a sort of research conversation starter. The various components were to be first approximations, not detailed models, of the actual brain regions. The theme was brain-inspiration, not brain replication. Borrowing from [8]:

> What we have outlined here is an ambitious program, and certainly we have not reached anything like completion. [...] It is much easier to convey the sense of 'work in progress' when speaking than when writing, and we hope that the necessary formalities of text do not obscure the fact that we are still groping for the correct formulation of our ideas. We also hope that, incomplete as it is, others will find the current state of our understanding useful and perhaps even provocative.

That being said, I would like to write down here a few specific changes I would make, given the benefit of hindsight, and to describe my personal shift in research direction in the years following this work.

Learning Instability

One practical problem became apparent when I was implementing and testing the first version of the Sapience software. I saw a sort of instability that occurred when the system was learning both perceptions and actions online. Given a random initial control policy, it would learn to represent the resulting observations, and given that sensory representation, it would try to improve its actions. Unfortunately, this dual optimization would often result in a terribly useless result, essentially getting "stuck in a corner" of sensory/motor space. After getting stuck, it would simply learn to devote all perceptual resources towards representing a single observation (the "corner"). This often occurred after receiving a reward, in which case the action just preceding the reward would be reinforced. But with the perceptual system not yet established, there was no clear mapping from sensory data to any particular internal state, so it could not learn to associate the action with any context. Instead, the action was presumably being reinforced in all states. Thus, it would end up with the entire sensory system trained on a single unchanging input pattern, and one motor output pattern being repeated indefinitely. In short, the overall learning process was highly unstable.

This problem was both frustrating and interesting. The solution I found at the time was to avoid learning from immediate observations, and to store all observed data in a sort of repository to be learned from later. So the system's experience was separated into distinct "experience" and "learning" (or "awake" and "asleep") phases: one for information gathering, and the other for learning. During the learning phase, data from the stored repository was replayed in a shuffled order, sampled according to some distribution. I initially resampled the data uniformly (i.e. according to the experienced distribution), but later achieved better results with a biased distribution which was weighted by motivational relevance (e.g. temporal difference error magnitude).

Interestingly, the idea of "experience replay" (or "hippocampal replay") seemed to have a lot to do with certain phases of sleep. Research on live animals shows experience gathered while awake being replayed in various ways (shuffled order, backwards, etc.) during sleep. Also, in the machine learning literature, it appears that the experience replay approach has become popular in recent years due to the stability it provides. It is worth noting that much of the underlying issue here, i.e. learning from online vs. stored experience, has long been discussed in the reinforcement learning

literature, especially when using nonlinear function approximators. Guaranteeing stability in such scenarios has always been an imporant research problem. I think I learned this lesson the hard way.

Architecture Improvements

If I were to produce a second version of the Sapience architecture, I'd make the following changes:

- The Sequential Memory component was originally described as analogous to the hippocampus, but I think it should represent the thalamus instead, or at least certain aspects of it. The thalamus makes more sense as the link between successive sensorimotor states. In addition, this component should have input/output links with all nodes in the Sensorimotor Belief Network, not just the root node.
- The hippocampus should be represented by a new component (called e.g. the Data Archive) whose role is to store observations for later learning. This is directly related to the above discussion on "experience replay."
- The Sequential Decision Memory should have used a standard reinforcement learning algorithm (e.g. an actor-critic architecture with temporal difference learning) rather than the PVLV method. I had decided to base this component on PVLV because I saw it as the most well-defined functional model of the basal ganglia available in the literature. However, a more standard algorithm would have been much more straightforward to implement, test, and understand in relation to the other components the architecture.

Broadly speaking, a different (probably better) way to develop a general purpose brain-inspired cognitive architecture like Sapience would be to start from a solid theoretical foundation, e.g. considering the set of all probabilistic models, and then gradually incorporating constraints from biology. This would help maintain a sense of confidence in the result. What I did instead was essentially the opposite, starting with a high-level structure based on the various brain regions and their connectivity, then trying to find algorithms to fit the presumed function of each brain region.

Change of Focus

I have not worked on this project since 2010. Within a year after writing the following document, I began to realize the difficulty of trying to maintain both biological plausibility and practical functionality in the same design. My attention was divided trying to satisfy the two simultaneously. It seemed prudent to choose one or the other, so I began to shift my focus away from the brain-inspired approach and move towards a deeper understanding of machine learning principles.

This was a difficult transition, as I had always gained a great deal of inspiration from studying biological intelligence, but ultimately I cared more about finding better learning algorithms. Also, I enjoyed the measurability of progress that is inherent in machine learning research. The alternative (construct biologically realistic brain models) presented a different set of challenges that did not appeal to me as a researcher (e.g. trying to convince an audience that "This is how the brain works...").

Over the next few years, I worked to solidify my knowledge of core concepts like Bayesian inference, information theory, and probabilistic graphical models. As I gained improved understanding of these topics, I saw the importance of deriving learning methods from principled objectives. This path held a certain reassurance that was hard to ignore.

Now in 2020, my overarching research goals are similar to those back in 2009, but with improved theoretical foundations. I continually search for ways to derive and design methods that are more simple and elegant. I think it's possible that all aspects of Sapience here could emerge automatically from a simple but powerful machine learning model. Specifically, consider the purpose of the various Sapience components: high-dimensional multimodal observation and action representations, temporal pattern learning, reinforcement learning, high-dimensional action memory, and long-range context storage/retrieval. All of these aspects might possibly emerge naturally from a single powerful model class (e.g. some form of graph-based model, which could be either directed or undirected and either probabilistic or deterministic), paired with efficient inference and learning algorithms. This general mindset is what drives my current work, and I am excited to see where it leads.

Tyler Streeter
December 2020

Sapience: A Brain-Inspired Cognitive Architecture

Tyler Streeter

June 1, 2009

Thesis proposal for the degree of Doctor of Philosophy

Human Computer Interaction

Iowa State University

Program of Study Committee

James Oliver, Major Professor

James Bloedel

Steven Herrnstadt

Adrian Sannier

Alex Stoytchev

Eliot Winer

Abstract

We present a novel cognitive architecture, Sapience, inspired by the high-level organization of the mammalian brain. This architecture includes five components based on abstract functional representations of the brain's sensorimotor cortex, hippocampus, basal ganglia, cerebellum, and prefrontal cortex regions. Each component provides a unique computational benefit to the system. Our focus is primarily biologically-inspired engineering, not biologically plausible brain modeling. Simply, we aim to build a useful artifact, not explain brain functions, although the latter is hopefully achieved to some degree. We intend the Sapience architecture to be applicable generally to all kinds of applications involving autonomous real-time learning and control, including video games and robotics. Crucially, it is designed to be practical: fully implementable in software, scalable in performance based on available computing hardware, and general enough to be applied to a wide variety of control problems.

The objective of our system is to learn complex motor control tasks defined as reinforcement learning problems. This includes any general situation involving real-valued arrays of sensory inputs and motor outputs. The system's behavior is shaped by two sources of reinforcements: 1) external rewards for achieving goals, defined by the programmer, and 2) internal rewards for improved understanding of the world, aka "curiosity rewards." This curiosity drive is based on recent theoretical work in artificial curiosity which provides a powerful model of autonomous self-development.

Our contribution is a concrete, implementable, brain-inspired cognitive architecture which integrates several key features. It uses a general environment interface definition (real-valued arrays) for sensory inputs (visual images, audio, proprioception, touch, etc.) and motor outputs (servo control variables). It uses topographic maps to learn mappings from arbitrary sensor arrangements. It learns a robust probabilistic context representation using a hierarchical Bayesian network with adaptable (unsupervised learning) kernel mixture models for the conditional probability distributions. It learns sequential pattern storage and recall of Bayesian priors. It performs reinforcement learning of context-dependent actions based on external and internal rewards. It uses a curiosity drive based on world model improvements, which encourages active sensory exploration. It learns to automate parallel action selection via supervised learning. Finally, it includes a gated working memory with read/write actions, enabling reinforcement learning of arbitrary programs.

After describing both the high-level and component-level architecture design, we evaluate our implementation on several experiments. A simulated testing environment is employed involving a 3D physically

realistic human arm and hand with tactile, visual, and proprioceptive sensory inputs and servo-like motor outputs. The Sapience implementation is connected to this simulated body, and we selectively enable and disable its various components, measuring changes in learning progress, with the intent of showing distinct computational benefits for each component. The primary metrics of learning progress are based on the system's two objectives: achieving externally-defined goals and improving the internal world model.

Contents

1 Introduction

Our species has the distinct ability to accumulate knowledge across generations. Any new useful knowledge gained by one generation is passed on to the next. (By "useful" we mean knowledge that helps us achieve our goals.) Thus, we are involved in a long sequence of cultural evolution, survival of the fittest ideas. Similar to biological evolution, which uses mutation-based exploration of gene space to achieve its goals (proliferation of stable genes), human cultural evolution uses our curiosity-based exploration through idea space to achieve human goals.

An important component of our cultural evolution is the evolution of our technology. We leverage technology to extend our reach in every domain, both physically and mentally. We utilize physical technologies to shape the physical world, and we use information technologies to enhance our mental (computational, memory, and communication) abilities. Each successive generation in our technological evolution helps us achieve our goals more quickly and efficiently; we continually achieve more with less effort, increasing global wealth, enabling us to focus on new problems with more powerful tools. Arguably, this progression is generally beneficial to our global society. Although technology both amplifies our constructive and destructive behaviors, the benefits usually outweigh the risks.

One particular milestone in the progression of human technology will be the creation of an intelligent entity with mental capabilities comparable to our own. Such a system would necessarily be as capable as a human in all kinds of tasks, including physical and mental labor. The mass production of such systems, designed to assist in solving human problems, would greatly accelerate the growth of global wealth and, therefore, lead us more quickly to a better society.

Now, at the "knee of the curve" [30] in our technological progress, the necessary computing hardware for a human-level artificial intelligence will be available very soon (if it does not already exist in some form). However, much work remains to be done in terms of designing and implementing the necessary software systems which capture, in some form, the intelligent process running within our own brains. Such an advance is arguably one of the most important conceptual and technological achievements possible, as enunciated by William James in 1904 – "*the* scientific achievement, before which all past achievements would pale" – in 1904 and more recently by Bill Gates – "If you invent a breakthrough in artificial intelligence, so ma-

chines can learn, that is worth 10 Microsofts." Therefore, our goal is to make progress towards the practical design and implementation of human-like artificial intelligence. In this document we describe one possible design of such a system.

We do not discuss here the vast array of societal and ethical implications of such a creation. Fortunately, public awareness of these issues is rapidly growing due to efforts by various organizations (e.g., The Singularity Institute for Artificial Intelligence). By actively working toward this goal, however, we are asserting that, although such an intelligent system is potentially dangerous, the probable benefits will outweigh the probable risks.

In this introductory chapter we discuss various possible approaches to developing advanced AI, especially our chosen brain-inspired engineering strategy – to design a brain-inspired cognitive architecture which is directly implementable in software and can be embodied within simulated avatars and real robots.

1.1 General Strategy

There are many possible routes towards advanced AI. Some of these options include artificial evolution of neural networks [59], cognitive architecture engineering (SOAR [31], ACT-R [5], brain-inspired [18, 43], 4D/RCS [3], OpenCog Prime AGI [47]), complete neuron-level brain simulation [38], and intelligent search within program space [54, 37, 29]. Currently it is difficult to tell which approach is most promising, but it seems likely that each method will provide us with a unique understanding of intelligence.

The approach we adopt here is an engineered brain-inspired cognitive architecture. Besides utilizing results from machine learning and information theory, we gain inspiration from the mammalian brain, a powerful existence proof that advanced intelligence is possible. In this respect, our approach is a blend of the cognitive architecture, neural network, and neuroscience-based methods. We attempt to gather the best existing machine learning algorithms and use them to engineer software-based modules based abstractly on the major brain regions. Within the spectrum of brain-based systems, with "realistic brain models" on one end and "abstract engineered systems" on the other, our approach falls on the abstract side. We are less concerned with biological realism (making comparisons with known biological or psychological results), more with producing a practical, implementable archi-

tecture. We attempt to focus on the most important large-scale biological constraints, mimicking finer details only when they appear to provide significant functional advantages.

Even after narrowing our scope to brain-inspired architectures, there are still an overwhelming number of decisions to be made. When engineering a brain-like system, exactly how much detail should be included? If we try to include too many details, the system runs the risk of growing too complex, too difficult to implement, and too time-consuming to evaluate. If we abstract away too much, we run the risk of over-simplifying the brain's organization and losing any benefit of studying it in the first place. The difficulty is in deciding what to keep and what to ignore.

Here we focus on the brain's systems-level structures and functionality, i.e. the largest, highest-level brain structures. Namely, we include in our architecture components corresponding to the sensorimotor cortex, hippocampus, basal ganglia, cerebellum, and prefrontal cortex. Our general strategy is to replicate the assumed functional role of each of these structures, preferring more abstract representations where possible, but sometimes using more detailed components when there is a clear advantage to doing so. In designing each component, we progress through three main bodies of knowledge: neuroscience literature, systems-level computational brain models, and machine learning algorithms. The neuroscience literature provides general constraints on each brain region: how it is connected to other regions and how its removal affects behavior. Computational models provide a certain level of abstraction, removing many of the biological details and focusing squarely on function. Finally, machine learning algorithms can be seen as the most abstract, idealized versions of what the various brain regions are "trying to do."

In this architecture we include only the most prominent structures and just a subset of the interconnections present in the real mammalian brain. This simplification can be justified in that there is still much uncertainty about which connections are functionally important. In the brain every major structure is connected to nearly every other major structure to some degree; the difficulty lies in determining which connections truly provide crucial functions. Are some connections redundant? If we cut one connection, can its role be fulfilled by another? Are some connections "accidental," merely a result of imprecise growth algorithms during initial development? Are some connections simply evolutionary baggage which don't provide any benefit or drawback to the organism? These questions are very difficult to answer, but

there are ways to gain intuition about them. For example, if we study the relative sizes of various connections, we can see that some are much larger and, when lesioned, result in much more severe behavioral deficits.

Another way to approach the problem is this: what was evolution trying to create? In other words, we can interpret biological evolution as an intelligent process with certain design goals. (Whether this interpretation is true is not the issue, and indeed the question might not even be meaningful. We only care about whether it is a useful concept in teasing out the brain's functions.) Helpfully, we can assume that each brain region has evolved to solve a particular problem while minimizing functional redundancy with the existing parts. This view is presented in [6], where the authors describe three distinct brain regions (posterior cortex, hippocampus, and basal ganglia/prefrontal cortex) as each solving fundamentally different computational problems. Similarly, Doya [15] describes the cerebral cortex, basal ganglia, and cerebellum in terms of three distinct learning principles (unsupervised learning, reinforcement learning, and supervised learning, respectively). These types of analyses provide much guidance in terms of understanding the global brain architecture.

Our brain-inspired cognitive architecture, which we call Sapience, is described in detail throughout this thesis. Some details of our architecture are currently still in flux. Due to time constraints, we have not yet tested each component extensively, much less the entire integrated system operating within a real or simulated body. Despite its current status, our system provides a novel, cohesive system which connects many complex components and provides an excellent platform for experimentation with embodied intelligent systems.

Our general research strategy has been the following: start with literature (neuroscience and machine learning) related to each of the five components, design and implement an abstract representation of each component, and perform many component-level tests (simulations with real-time visualization). To test the entire integrated system, we plan to develop a curriculum of increasingly difficult tasks, starting with the experiments performed in this thesis.

Note that this work primarily comes from the same inspiration as the author's master's thesis research [60, 63]: that it is desirable to have a general intelligent software system which can be applied to a wide variety of learning tasks with minimal changes. The work described here, however, is intended to be much more robust, scalable, and useful.

1.2 Outline of Thesis

Chapter 2 explicitly describes the learning objectives of our architecture, which provide constraints on the architecture and concrete metrics we can use to evaluate its performance. Chapter 3 presents a high-level overview of the architecture. The following several chapters describe in detail the core functional components of the architecture, including the biological inspiration for each, the design of our functional abstraction, and the various algorithms employed. We then list the work completed already (implementation and testing framework) and a set of proposed experiments on which we will evaluate the architecture implementation.

2 Learning Objectives

In this chapter we define the overall learning objectives for our architecture, which will influence many aspects of its design.

What is the overall "purpose" of our system? Put simply, it should learn to achieve goals as quickly as possible with minimal human intervention. It is most helpful to classify it as a reinforcement learning agent which tries to maximize its long-term reward intake. Within the theoretical reinforcement learning framework, we can make the concept of a "goal" very concrete by speaking in terms of rewards (positive reinforcement). Any task with clearly-defined goal states can be considered as a reinforcement learning problem, where we provide a reward signal to the learner when the goal state is reached. Thus, if the system learns to maximize rewards, we can say that it is achieving its goals. Fortunately, reinforcement learning requires relatively little human intervention; once the problem has been defined in terms of rewards, the agent learns primarily from trial-and-error, not from a teacher providing constant feedback.

We divide the set of reinforcements into two varieties: 1) external, provided by the programmer, and 2) internal, provided by the curiosity drive.

External Reinforcement External reinforcements are defined arbitrarily (e.g., by the programmer, by artificial evolution, etc.) to achieve some desired task and correspond conceptually to goal states. Qualitatively, we want our system to do what we ask of it – to complete its given task. Quantitatively, this occurs when the system learns to maximize its external reward intake.

Internal Reinforcement Internal reinforcements are a special type which motivate active exploration and model learning, as described below. Qualitatively, we want our system to display intrinsically-motivated exploration, play behavior, and to seek out interesting situations. Quantitatively, this occurs when the system learns to maximize its internal reward intake.

2.1 Reinforcement Learning

Here we describe briefly the process of reinforcement learning, which applies to all types of rewards (external and internal). One of the most well-cited sources on theoretical reinforcement learning is the textbook by Sutton & Barto [66].

The general reinforcement learning problem is this: an "agent" (autonomous system) lives in a world which, at each instant, provides the agent with a sensory observation and a reinforcement signal (a positive or negative scalar). Upon receiving the observation and reinforcement the agent must choose an appropriate action to execute, which will influence the environment and the subsequent observation and reinforcement. The agent must learn to choose its actions to maximize its long-term reinforcement intake. The reinforcement signal is problem-specific and should be chosen to represent the intended goals for the agent. Usually the agent utilizes an internal world model to make decisions.

Where do the reinforcement signals come from? Biologically, they are defined in our brains as a set of reward/punishment signals which represent evolutionarily-defined goals (food, water, mating, pain, etc.). Within our computational system, in general they can be specified arbitrarily to fulfill any desired task. Intuitively, the situation is similar to training a dog: all kinds of desired behaviors can be elicited from the dog as long as a reward is given immediately afterward. Notably, only one type of reward (e.g., a dog treat) is needed for all types of behavior. This greatly simplifies the overall process: the trainer does not need to provide detailed feedback about the dog's behavior, only the sparse signal at the very end (either a treat or no treat). In the same way, we can design a reinforcement learning system to perform a wide variety of tasks, all using the same type of externally-provided reward - in our case, a simple numerical signal (for example, a decimal value within [0, 1])[1].

[1]Note that this process requires the learner to recognize some pattern/place/object as being consistently paired with a reward. In the dog training example, the sight and smell

For our purposes the reinforcement learning framework provides a very useful abstraction of all kinds of learning tasks. It considers reward maximization as the core thing an agent tries to do; essentially, the ultimate purpose in an agent's life is to find more rewards and avoid punishment. With reinforcement learning as the core principle, any auxiliary computational system can be added which improves the agent's reward acquisition rate, whether by improving its world model, enabling it to plan ahead, favoring novel situations, etc. Indeed, we might even say that reinforcement learning is the *easy* part, given a good world model; the *hard* part is *learning* a good model of the world with which to make good decisions.

2.2 Model Learning

A reinforcement learner will only be as successful as its model of the world. Therefore, in parallel with the reinforcement learning process (i.e. learning context-dependent values and actions), the system should learn to improve its world model (the Sensorimotor Belief Network in our architecture, described later). To the extent that this model accurately reflects the true state of the world, the system will be more successful at achieving its goals (maximizing rewards).

Here we discuss the general process of learning a world model and how its progress might be measured. The ultimate goal for a world model is to achieve perfect predictions (zero prediction errors), which is closely related to data compression.

2.2.1 Prediction and Compression

At each instant, any learner receives sensory data samples – snapshots of the state of the universe – which are limited to the current instant in time, limited to a very small piece of space, and are corrupted by the noisy transduction process of sensation. These data samples, despite their imperfections, are all that is available to the learner from which it must build a model of the world. What is the best way to build such a model based on the given data samples?

---

of a treat is associated with upcoming reward. In the case of a simulated entity or real robot, this might be a special object that is rewarding to touch, which can be provided by the programmer for a short time upon completing a task.

First, we can start by considering a theoretical perfect world model, which contains an exact copy of the true external world, including the processes which generate noise during sensory transduction. Further, let's assume that this perfect model does not compress anything: it simply contains a huge array representing the state of the universe at each point in time (e.g., all state variables of all subatomic particles). Such a model would be able to predict sensory data perfectly at each instant. This would be the best possible resource for a reinforcement learning system which had the luxury of infinite computational resources. We could say that the ultimate world model would be the one with perfect sensory predictions. This serves as a learning target for non-optimal world models.

Next, let's consider a variation where the perfect world model is also compressed in a lossless manner. Rather than simply storing the entire state of the universe, it might contain models of all physical processes. Thus, instead of using an exhaustive table to *store* the state of things, it uses generative models to *compute* the state of things. (In [13], Chapter 10, the general principle behind "representational learning" is to learn "causal models" of the world, i.e. models of the processes which caused the sensed data samples.) Since we're assuming lossless compression, we can assume that the system's predictive capabilities remain identical to the uncompressed model. It is still a perfect predictor, but it uses a different internal representation.

Taking things one step closer to reality, what if limited memory storage constrained the world model to be compressed in a lossy manner? It would no longer be a perfect predictor since some details would be lost due to compression. Its sensory predictions would now have some non-zero degree of error. Before we said that a perfect world model would make perfect predictions. An imperfect world model should produce predictions which are as good as possible, given its limitations. This is achieved if the world model prioritizes its resources to model those physical processes which provide the best predictive capabilities, i.e. those that provide the greatest compression of sensory data. The more a model can compress, the better it can predict because it is implicitly better at capturing the underlying data-generating process. Since data prediction and data compression are essentially equivalent, we could say that the ideal realistic (lossy-compressed) world model is the one that compresses the most data.

2.2.2 Compression Progress

In most unsupervised machine learning situations, data samples are provided by some external system. Thus, the learner's goal is to model the distribution of the provided data space. Performance can be measured based on compression over the given data set.

When the learner has an active role in choosing its own data samples, the situation is fundamentally different. Here the learner is performing what Schmidhuber calls "active unsupervised learning" [53]. Assuming that the learner lives in a complex environment, and that it will never experience all possible data samples in its lifetime (or even come close), it is in the learner's best interest to find those data samples that provide the most benefit/the greatest reduction in uncertainty/the greatest model improvements. Now, instead of simply focusing on how well a model compresses a static data set, we must encourage the learner to *find good data samples*. This is the essence of Schmidhuber's principle of "artificial curiosity," reviewed in [53]. This simple principle provides a powerful approach to the problem of autonomous sensory data selection. If a learner has a limited lifetime (a realistic assumption), it must focus its sensory apparatus in a way that improves its world model *quickly*. The learner cannot waste much time on uninformative data[2]. It should try to compress as much data as possible per unit time.

To explain things in another way: in a memory space- and processing time-limited learner, there are multiple sources of modeling error. One source, discussed in the previous section, is lossy data compression. Another source of error, which is not as obvious, is a poor sampling of the sensory space – lack of experience. Any lossy-compressed model can be evaluated on a given data set, but in general the data set is not *given*, it must be *chosen* by the learner itself. Most unsupervised machine learning problems assume a fixed data set: the learner is isolated from the world except for the data samples provided by some external entity which decides the data distribution, and the learner must simply compress the given data. In contrast, for an embodied learner operating in a complex environment with potentially limitless data samples, the data distribution is *not* fixed beforehand. It is not good enough to sit in one place and wait for data to appear. Rather, the learner actively chooses its own path through sensory space, i.e. its own

[2]Intuitively, the most informative data is often that which demonstrates distinctions between categories.

data distribution. Its actions should be guided towards those parts of the world that provide the greatest improvements to its world model.

How can this be implemented within a reinforcement learning system? As initially described by Schmidhuber [52], a curiosity drive can be implemented as an extension of an existing reinforcement learning system, where the learner is simply given extra rewards whenever it makes improvements to its world model. Schmidhuber's principle appears to be general enough to be useful for any intelligent system which includes a learned world model. As reinforcement learning in complex environments is utterly dependent upon an accurate representation of the world, we assume that such an internal drive to improve the world model should be a standard, hard-wired motivation[3].

2.2.3 Measurable Objective

Two types of model errors exist: limited storage space necessarily demands lossy compression errors, and limited time necessitates an incomplete data sampling of the world. These errors demonstrate two different needs for our system: compression errors on the given sensory data can be improved by improving the world model's internal representation (through learning), and errors due to a poor sensory sampling can be improved with a good action selection system which is motivated to find more informative data samples. Both types of improvements are reflected in a single measurement: the model's compression improvement over time. This measure summarizes very simply the quality of the model itself *and* of the data distribution used to train it. (We describe in a later chapter how we measure compression improvements in our system.)

2.3 Two Learning Objectives and Metrics: External Rewards & Compression Progress

We define two learning objectives for our system: achieving externally-provided goals, and improving the world model. These objectives represent specific motivational drives for the learner, and they provide us with corresponding metrics for evaluating system performance.

[3] As Aristotle said in The Metaphysics, "All human beings by nature desire to know."

Objective 1: Achieve External Goals This objective is implemented within the learner as a drive to acquire *external* rewards. As an evaluation metric we can measure the learner's external reward intake over time.

Objective 2: Improve World Model This objective is implemented within the learner as a drive to acquire *internal* rewards – proportional to model improvements, which includes finding good data plus compressing the existing data. As an evaluation metric we can measure the learner's internal reward intake (or, equivalently, model improvements) over time.

Note that from the learner's perspective, all types of rewards are beneficial, so the ideal strategy is to achieve both objectives by maximizing total reward intake. Also, note that achieving objective 2 indirectly helps to achieve objective 1; a good world model helps achieve external rewards faster [61].

3 The Sapience Architecture

In this chapter we describe the high-level design of the Sapience architecture. We begin with a discussion of the brain. Then we move on to the architecture itself: its overall organization, major components, interconnections, and update process. Note that the entire architecture has already been implemented. No aspect of the components being described here is merely a hypothetical mechanism; every part has been instantiated with a concrete software representation.

3.1 Brain Inspiration

The mammalian brain can be described at many levels of abstraction, from the molecular level up to the highest systems level. Each of these types of description is useful in its own way. For example, Doya [15] divides the brain into three major components (cerebellum, basal ganglia, cerebral cortex) based on their assumed learning mechanisms, a very helpful way to gain intuition into the functional roles of these brain regions. Similarly, Atallah et al [6] describe a cognitive architecture including posterior cortex, hippocampus, and basal ganglia/frontal cortex, based on differing computational trade-offs in neural network models.

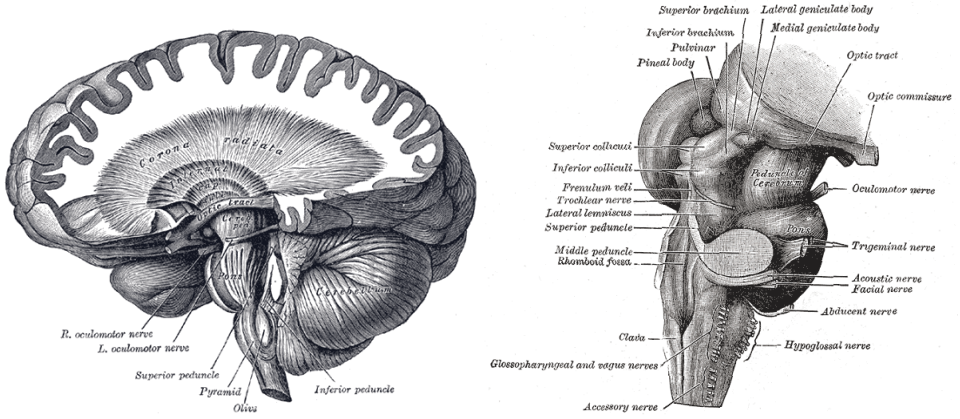


Figure 1: Illustrations of the brain showing cerebrospinal fibers and midbrain/hindbrain anatomy. From Gray's Anatomy (public domain).

One interesting point is that the mammalian brain architecture appears to follow roughly the same general organization across different species. The main differences are numerical: higher mammals have larger cerebral cortices with corresponding larger neural projections from cortex to basal ganglia, cerebellum, etc. However, despite an overall difference in scale, we assume that the overall functions remain similar across mammalian species. Thus, similarly, we design our brain-inspired cognitive architecture to provide a certain set of functionality but to be scalable in terms of computational power (e.g., when executed on different computing hardware) – essentially a speed vs. accuracy trade-off.

In this chapter it seems most useful to start with an informal, grossly simplified description of the brain's general operation. The biological details of these brain structures are covered more thoroughly in later chapters; for now, it is best to eschew the finer details and focus on overall function. The following is an informal (possibly wrong, but hopefully useful) explanation of global brain function:

- The basal ganglia is the "core," a sort of mini brain within the brain. If we had to choose one part to be the homunculus (in the "Cartesian theater" sense – the "little man" within), the basal ganglia would be it[4]. The basal ganglia essentially act as the decision making center: based on the current state of things, which action should be chosen

[4]Of course this is not literally true, but the analogy is helpful.

next? It learns its action selection from trial-and-error based on rewards and punishment.

- The sensorimotor cortex is a "mental model" of the external world. It learns from sensory data to build a compact/compressed representation of the underlying processes which generated the data. Other brain regions use this as a general context representation: what is happening in the world? Since it includes a model of motor space (i.e. motor variables like muscle length and tension), the motor regions can be used by the basal ganglia as a set of potential action choices.

- The hippocampus stores snapshots of specific events (i.e. global cortical states). When a similar event is later encountered, it can recall the original cortical state, biasing the cortex to expect the same patterns as before. This can operate sequentially, enabling storage and recall of cortical "video," i.e. successive frames of cortical states.

- The cerebellum learns to automate well-learned behaviors. The basal ganglia is an information bottleneck: serial decisions routed through it take a while. However, when the same decision is always chosen in the same context, it is useful to store that decision elsewhere (in the cerebellum), freeing the basal ganglia to focus on novel decisions. Furthermore, it can operate in parallel, driving many motor/cognitive actions at once, whereas the basal ganglia seems to be limited to serial action selection.

- The prefrontal cortex acts as a set of working memory cells with read and write capabilities, not unlike computer RAM. It provides to the basal ganglia a set of memory actions (read/write), an extension of motor actions. Instead of choosing to perform a particular motor action (e.g., initiating a gesture), the basal ganglia can choose to write something to working memory (e.g., the state of a particular sensorimotor variable). Later, the basal ganglia can choose to read these contents back out, allowing them to influence other brain regions, including the sensorimotor cortex and the basal ganglia itself. The resulting basal ganglia-prefrontal cortex subsystem acts something like a general purpose digital computer with the ability to learn arbitrary programs to increase rewards. Such a system provides the brain with powerful capabilities like planning – mental simulation of "what if" scenarios in order to train the decision maker without direct interaction with the real world.

- The brain as a whole works to increase positive reinforcements. The basal ganglia is the central decision maker, and all other components serve to help it make better decisions, whether by providing a more informative summary of the world, by offloading frequent decisions, training through mental simulation, etc.

3.2 High-Level Description

Our cognitive architecture includes five main components inspired by the major structures of the mammalian brain, as shown in Figure 2. Each component has a unique computational purpose. Here we introduce them with a summary of their roles. More detailed descriptions are given in later chapters.

Sensorimotor Belief Network This component is a probabilistic model of the external world. It receives raw input data from the environment via the body's sensors, and it produces control signals to be applied to the environment via the body's effectors. It learns to represent the discrete causes of the data it receives (i.e. a causal model). In other words, given a sensory pattern, what is the assumed cause of that pattern? This is accomplished by employing Bayesian inference, organized in a hierarchical Bayesian network, and with learning kernel mixture models. It has reciprocal connections with the Sequential Memory, which provides sequential predictions/priors based on past experience. It can be influenced by Working Memory, and its motor representations can also be driven by the Serial Decision Maker and Parallel Decision Memory. It provides a probabilistic "belief" representation to these other components.

Sequential Memory This component learns to make sequential predictions based on past experience. It watches the sequence of data coming from the Sensorimotor Belief Network and can learn to store and recall these sequential patterns. This adds an element of temporal representation to the system's model of the world. It also sends a copy of its internal temporal representation to the Serial Decision Maker for making precise reward predictions.

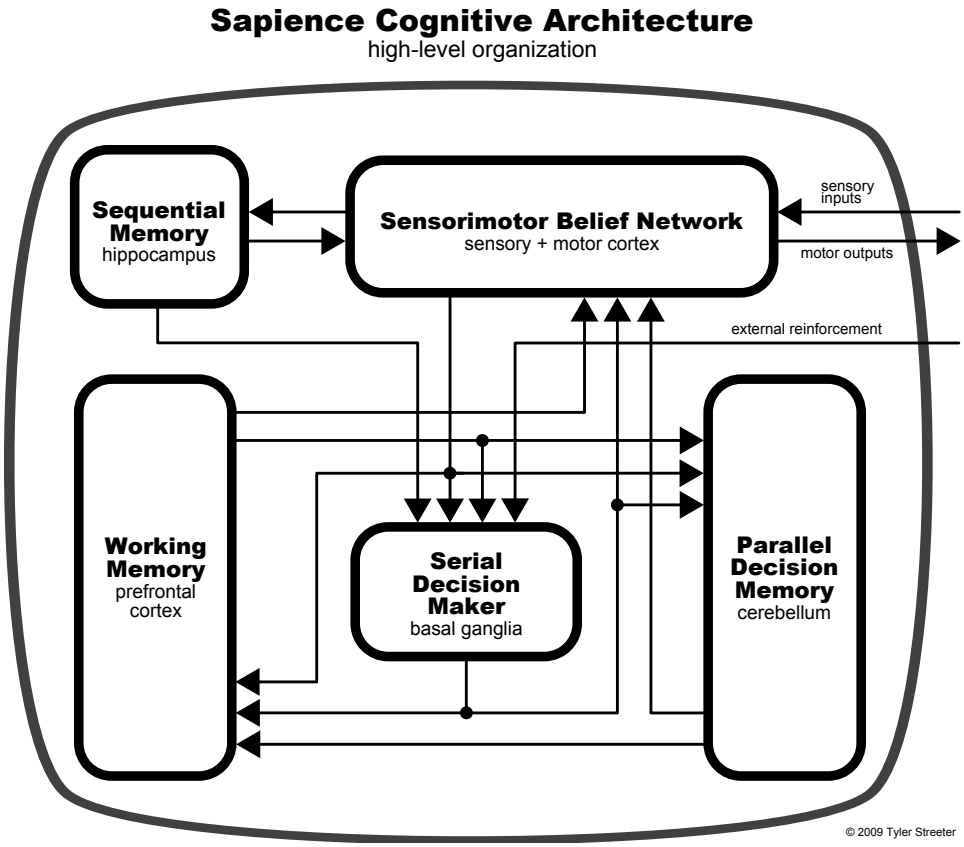


Figure 2: Primary components and connections of the Sapience architecture. Each component is labeled based on its intended function, along with the name of the corresponding brain region.

Serial Decision Maker This component performs two main functions: learning context-dependent value/utility, and learning context-dependent action selection. Based on the estimated state of the world (as represented by the Sensorimotor Belief Network and the Sequential Memory's temporal representation), the Working Memory state, and the current external reinforcement, it learns to estimate the value of each situation. Concurrently, it learns to select from a discrete set of actions, which includes both motor actions (influencing the Sensorimotor Belief Network's probabilistic representation of motor space) and working memory actions (reading and writing Working Memory contents). Its action choice is copied to the Parallel Decision Memory as a teacher signal for supervised learning.

Parallel Decision Memory This component automates well-learned decisions. It constantly watches which actions (Serial Decision Maker outputs) are chosen in each context (Sensorimotor Belief Network state and Working Memory state). Over time it learns to assume control of the most often-selected actions, freeing the Serial Decision Maker to focus on novel decisions. Its parallel nature makes it capable of driving multiple motor and working memory targets simultaneously.

Working Memory This component provides a discrete set of general purpose memory cells. Each cell's contents can be controlled by the Serial Decision Maker, which is then influenced by the Working Memory contents – a recurrent system, programmable via reinforcement learning. Working Memory contents also influence the Sensorimotor Belief Network state and provide an enhanced context for the Parallel Decision Memory.

3.3 Update Process

To remain as general as possible, all communication among components and with the external world is represented with real-valued arrays. On each update step we provide the system with new sensory data and external reinforcements, perform internal processing, and access new motor control signals. Internally, each update step performs component-specific processing and inter-component communication. Here we detail the update process from an external perspective (as seen from a program which uses the architecture) and internally within the architecture.

External Loop The following procedure illustrates the main update loop, i.e. how the system should be updated iteratively within some program. This assumes that the system is embodied, embedded within a simulated avatar or real robot.

1. The body's sensors capture an observation, the current state of the environment as viewed through the sensors. This observation is represented as arrays of real-valued data.
2. External reinforcements are computed (e.g., as defined by the programmer).

3. The sensory observation and external reinforcements are provided to the system.
4. The system performs internal processing, including learning from sensory data, learning from reinforcements, and computing new motor outputs.
5. The system's new motor outputs are applied to the body.
6. The environment (the real world or a simulation) is allowed to advance forward in time by some fixed interval.
7. Go to step 1.

As the system is expected to operate in real time, we assume a fixed update interval. For example, assuming an update rate of 10 Hz, at regular intervals of 100 ms the entire set of sensory inputs and external reinforcements are sampled, all internal processing is performed, and all new motor outputs are applied[5].

Internal Processing The following procedure illustrates the system's internal update procedure, which includes everything between providing new incoming sensory inputs and reinforcements to applying new motor outputs:

1. Update the Sensorimotor Belief Network's belief state:
   (a) Propagate the latest sensory inputs (Bayesian likelihoods) through the network.
   (b) Apply extra influences ("virtual evidence" in Bayesian network terminology) from Parallel Decision Memory (motor influence only) and Working Memory.

[5]In practice there is, of course, the issue of limited processing power. What if the internal processing takes too long, and each update iteration takes longer than the desired interval? In this case there are two options: find a faster computer, or decrease the system's processing requirements. For this reason our architecture includes a set of free parameters, described in a later chapter, which provide a speed/accuracy trade-off. This allows more accurate internal representations on faster computers, and lower accuracy (but faster performance) on slower computers. To help determine the optimal degree of speed vs. accuracy, implementations might include a built-in timing system which provides at runtime an estimate of the time needed for internal processing. Furthermore, an implementation might even run a benchmark test on start up and use the results to determine the optimal accuracy settings based on the computer's hardware limitations.

(c) Propagate the latest Sequential Memory predictions (Bayesian priors) through the network.

(d) Compute a new set of "beliefs" about the world (Bayesian posterior distribution).

(e) Train the kernel mixture models (conditional probability distributions) from the sensory input data.

(f) Compute internal curiosity rewards based on prediction/compression progress.

2. Update the Sequential Memory:

   (a) Access the latest input pattern (the current state of the Sensorimotor Belief Network multimodal root node).

   (b) Train the predictor neural network, whose activity still represents its last-step prediction of the current-step pattern, with the actual current input pattern.

   (c) Update the temporal trace array with the current input pattern.

   (d) Update the predictor neural network to produce a prediction for the next-step pattern (for the Sensorimotor Belief Network's multimodal root node prior).

3. Update the Serial Decision Maker:

   (a) Use the current system state (from the Sensorimotor Belief Network, Sequential Memory temporal pattern, and Working Memory contents) to update the value function and the policy (action/decision).

   (b) Train the value function based on the current reinforcements (external and internal).

   (c) Use the value function to compute a training signal for the policy.

4. Update the Parallel Decision Memory:

   (a) Use the current system state (from the Sensorimotor Belief Network and Working Memory contents) to compute a set of parallel actions (to influence Working Memory and the motor regions of the Sensorimotor Belief Network).

(b) Use the latest decision from the Serial Decision Maker to compute any errors in the current context-dependent outputs, and adapt them as necessary.

5. Update the Working Memory:

   (a) Use the current decisions (from the Serial Decision Maker and the Parallel Decision Memory) to update the read/write states of the memory cells.

   (b) Cells with "read" enabled pull data from areas of the Sensorimotor Belief Network.

   (c) Cells with "write" enabled influence areas of the Sensorimotor Belief Network.

4 Sensorimotor Belief Network

Here we describe the Sensorimotor Belief Network component of the Sapience architecture. This component is essentially an internal representation of the external world – a world model. In designing this component we utilize Bayesian inference as a unifying framework: at any given time this model provides a probabilistic estimate of the state of things based on current sensory data and prior experience. Furthermore, it learns continually from all incoming sensory data by using an adaptable kernel mixture model.

In this chapter we begin by summarizing various key elements of the sensory and motor regions of cerebral cortex. Then we describe our abstract Sensorimotor Belief Network component, including its unifying theoretical principle of Bayesian inference, an unsupervised learning kernel mixture model which converts raw data arrays into probability distributions, the usage of a hierarchical Bayesian network to address the curse of dimensionality, methods for computing prediction/compression progress for internal curiosity rewards, a method to produce motor control signals, and a reflex system for bootstrapping the learning of motor representations.

4.1 Inspiration from the Sensorimotor Cortex

In the mammalian brain the cerebral cortex[6] is a 2-4 mm thick two-dimensional sheet surrounding many other structures. As a general indicator of its importance, its relative size has increased significantly in higher mammals, corresponding to more complex perceptual abilities and behaviors.

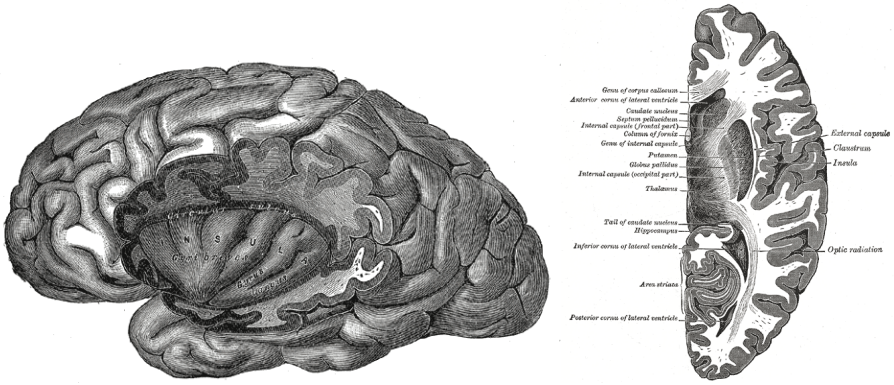


Figure 3: Cerebral cortex illustrations. The left image exposes the insular cortex located in the hidden interior walls of the two hemispheres. The cross section on the right clearly shows a distinction between the gray matter (cell bodies/processing) and white matter (neural fibers/communication). From Gray's Anatomy (public domain).

Here we look specifically at the sensorimotor cortex, by which we mean all regions of cerebral cortex except the prefrontal area (considered separately in a later chapter). A much more detailed account can be found elsewhere [27, 1, 12]; we mainly focus on a few key aspects that influence our architecture design. Briefly, we are interested in its large-scale connectivity (inputs from all kinds of sensory sources, outputs to many other brain regions), its columnar structure, and its apparently homogeneous, data source-agnostic functionality.

Anatomy The sensorimotor cortex (occipital, parietal, temporal cortex and parts of the frontal cortex) receives major inputs from a large variety of sensory sources via the thalamus, including:

[6]The word "cortex" is Latin for "shell" or "bark." The cerebral cortex is physically shaped like a shell around the cerebral hemispheres.

- Visual: color, motion, depth
- Auditory: frequency-domain patterns
- Somatosensory (skin-related): pressure, vibration, temperature, pain
- Somatosensory (muscle-related, i.e. proprioception): muscle length, muscle length changes/velocity, muscle tension
- Vestibular: head angle w.r.t. gravity, head acceleration vector
- Gustatory: flavors – chemical patterns on the tongue
- Olfactory[7]: scents – air born chemical patterns

There exist separate cortical regions for each data source. Typically, sensory data coming from the thalamus reaches a primary sensory region of cortex, where it is processed and sent to a secondary sensory region, tertiary region, etc., until finally reaching the multimodal association regions in parietal and temporal cortex. Alongside these "bottom-up" cortical connections (including thalamocortical and corticocortical) there are "top-down" connections from higher (e.g., multimodal) regions down towards lower primary regions.

In each of the sensory cortical regions it is common to find a topographic arrangement of sensory representations (e.g., retinotopic maps of visual patterns, somatotopic maps of skin surface patterns, tonotopic maps of audio frequencies, etc.) This topographic property becomes less apparent (or perhaps just harder to measure) in the higher-level multimodal regions. The benefit of maintaining this topographic relationship is that computational elements that deal with similar data are physically close to each other, enabling shared computation with short inter-element connections. This allows sensory data from various arrays of sensors (e.g., skin, retina, cochlea) to be mapped onto a two-dimensional sheet of cerebral cortex in a way that maintains the local relationships of the data points.

Most sensorimotor cortical regions can be considered purely sensory, with no direct influence on motor output signals sent to the muscles. However, the primary motor cortex (intimately connected with somatosensory areas) does possess such connections. Its layer V giant Betz cells form the "corticospinal" (or pyramidal) tract (Figure 4), a set of output connections which travel from

[7]Interestingly, olfactory data is not routed through the thalamus. One possible explanation is that the thalamus might act to maintain topographic organization during the transmission of data from the sensory organs to the cortex. Olfactory patterns do not seem to have a clear need for topographic representation.

motor cortex, down the spinal cord, and synapse directly on the muscles[8]. This pathway is crucial for voluntary movement.

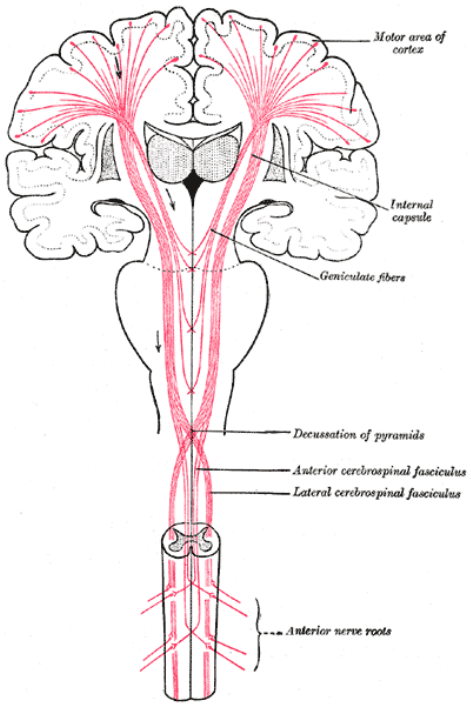


Figure 4: The corticospinal/pyramidal tract: motor output signals from the motor cortex to the muscles. From Gray's Anatomy (public domain).

The large-scale connectivity between the sensorimotor cortex and other brain regions is described in many sources [27, 1] which we summarize here. All regions of sensorimotor cortex (except primary visual and primary auditory) project to the striatum region of the basal ganglia (see [48] Figure 17.4). The basal ganglia have output connections back to the motor-related cortical regions (via the motor areas of the thalamus). Only the multimodal sensorimotor cortex projects to the hippocampus, which sends outputs back to the same multimodal areas (see [27] Figure 62-5). Nearly all areas of cerebral cortex project to the cerebellum's mossy fiber system ([1] p. 206), and, similar to the basal ganglia, the cerebellar nuclei send outputs back to the motor-related cortical regions ([1] p. 211) and possibly nonmotor areas like

[8]The premotor and supplementary motor areas of frontal cortex also contribute significantly to this pathway.

prefrontal cortex ([1], see discussion of nonmotor functions on pp. 218-219). Most sensorimotor areas have reciprocal connections with the prefrontal cortex region.

Within all regions of sensorimotor cortex, there appears to be a common columnar organization (reviewed in detail by Mountcastle [41]), resulting from a radial migration of cells during cortical development. A cortical column (or hypercolumn or macrocolumn) is roughly 0.5 mm in diameter and contains many minicolumns (or microcolumns) – on the order of 100 minicolumns per column. The entire human cerebral cortex contains roughly 2,000,000 columns ([26], Table 2).

Function The sensorimotor cortex is generally involved in representing information related to various sensory and motor modalities. More specifically, it appears that these representations are based on its columnar organization (again, see [41]). Each column appears to represent a single "variable," with each minicolumn representing one possible value of that variable. For example, in visual cortex the minicolumns appear to represent edges in visual space [23], organized topographically (each neighboring minicolumn represents a visual edge of slightly different angle). Similar representations exist for directions of visual motion (MT area of visual cortex), whisker deflection in rats (cortical "barrels" appear functionally equivalent to columns), audio intensity organized along isofrequency bands in auditory cortex, somatotopic maps (skin pressure and vibration, muscle length and tension), and also movement representations (direction vectors) in motor cortex [17]. In each case the sensorimotor cortex appears to be relatively agnostic to the type of data provided. Thus, we might assume a homogeneous function: similar circuitry across all sensory modalities implies a common operation in all cortical regions.

Concerning motor pathways, what do the corticospinal motor outputs actually represent? It appears that the primary motor cortex and nearby somatosensory cortex send signals to and receive signals from the muscles and tendons [48]. Signals coming from the muscles include the muscle length (encoded by muscle spindles) and muscle tension/applied force (encoded by golgi tendon organs). Signals going to the muscles include the "desired" muscle length (encoded by alpha motor neurons) and "desired" muscle tension (encoded by gamma motor neurons). Interestingly, the low-level feedback circuits present between the spinal cord and muscle/tendon provide a functional unit for each joint which can be viewed abstractly as a P controller

(simplified PD or PID controller). A P controller is a spring-like system whose inputs include an equilibrium position (desired state/angle) and gain (desired stiffness).

Primary somatosensory cortex (SI) contains four complete maps of the body (Brodmann's areas 3a, 3b, 1, and 2), each representing different types of information. Area 3a, which processes proprioceptive sensory inputs, is adjacent to the primary motor cortex (MI, Brodmann's area 4), which sends control signals to the muscles. The input and output data types in these nearby cortical areas appear to encode similar types of information.

Computational Models Several computational models have been proposed regarding cortical function. Most of them assume a hierarchical organization [49, 33, 16]. Some are based on the idea that unpredicted data (prediction errors) propagate up the cortical hierarchy [49, 16]; only data that cannot be modeled at one level is passed to the next, which presumably has a wider view of things and should be more capable of modeling complex patterns. Other models are based on a Bayesian inference framework [33], treating the cerebral cortex as a large Bayesian network [44] which computes "beliefs" (posterior distributions) about a large collection of variables.

4.2 Functional Abstraction

We assume that the sensorimotor cortex is organized as a hierarchy of functional units (columns) which can be represented abstractly as a Bayesian network[9] of "belief nodes." Each node in this network represents a variable concerning some aspect of the sensory space, and the node contains a discrete number of possible values/hypotheses (similar to minicolumns) for that variable. The task of the entire network is to compute the posterior probability distribution at each node with the intent of representing an estimate of the state of the world in a probabilistic way.

We use these general principles to design the Sensorimotor Belief Network (Figure 5), a component of the Sapience architecture which acts as a probabilistic representation of the external world (a world model). Each node

[9] We choose to focus on the Bayesian network representation, instead of prediction error propagation, because of its formalism and because the prediction error approach appears to provide the same types of messages between nodes as log-space Bayesian inference, described in [62].

(inspired by the cortical column) learns a symbolic representation from its given data. The nodes use kernel mixture models (inspired by cortical mini columns) to convert raw sensory input data into probability distributions. To break up the potentially huge, high-dimensional, multimodal sensory input space, the network is structured as a hierarchy: the bottom-level nodes model small patches of the input space, and the higher-level nodes model the results of the lower-level nodes.

The major output array is a concatenation of all nodes' posterior distributions – essentially a huge "belief vector." This array is used by various other components (e.g., as an input pattern for Sequential Memory predictions, as a state representation for the Serial Decision Maker, etc.) Its contents are more linearly separable (in a pattern recognition sense) than the input patterns because the posterior vectors provide sparse patterns (the kernel mixture model learning rules aim for minimal posterior entropy, i.e. very few active units at a time). This property enables better learning results for single-layer neural networks within other components that use the sensorimotor belief state as inputs.

The motor areas of the sensorimotor cortex are almost identical to the purely sensory areas except that they are able to control the variables they model (e.g., joint angles and motor force output). Therefore, in the Sensorimotor Belief Hierarchy the bottom layer of motor-related nodes sends prior (prediction) signals out to the body as control signals for its effectors.

4.3 Bayesian Inference

The main unifying idea behind the Sensorimotor Belief Network is Bayesian inference, as defined by Bayes' theorem:

$$P\left(C|E\right) = \frac{P(C)\,P(E|C)}{P(E)} \tag{1}$$

We have two variables: E is the "evidence" variable representing the input data, and C is the "class" or "hypothesis" variable, a latent variable which models the possible causes of the evidence. In general both E and C can be discrete or continuous. We focus on the case where E is a continuous vector variable and C is a discrete variable which takes one of a finite set of possible values.

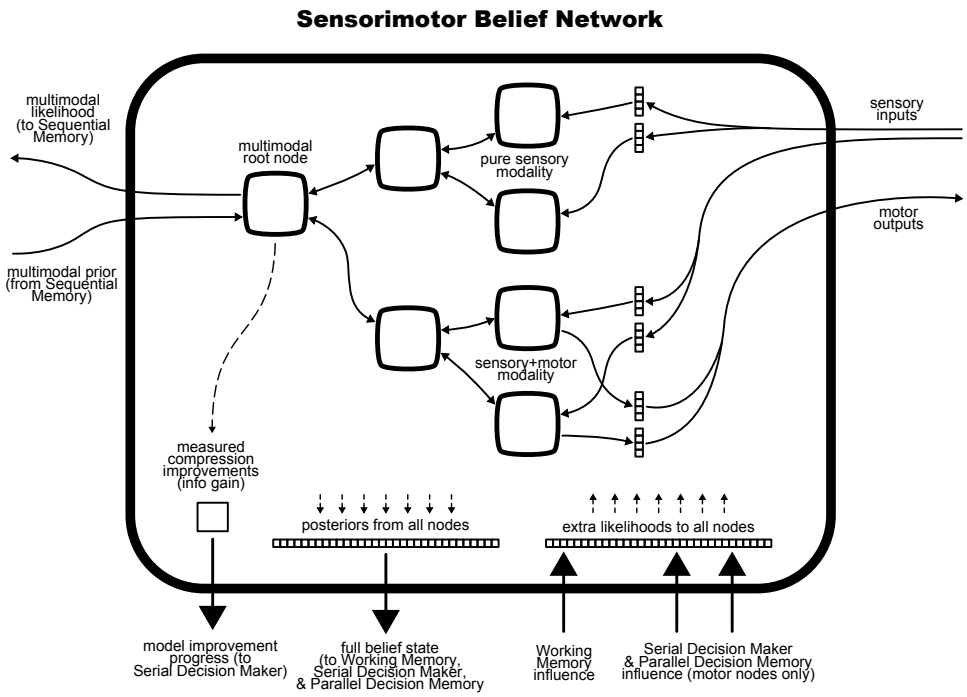


Figure 5: The Sensorimotor Belief Network component, a hierarchical Bayesian network representation of sensory and motor data. Sensory inputs are provided for all modalities, but only motor-capable modalities can send motor control signals back out (see right side of diagram). Large, high-dimensional input arrays are broken into smaller arrays (to address the machine learning "curse of dimensionality") which are then processed by multiple Bayesian nodes. The hierarchical arrangement converges onto a single multimodal root node. All inter-node connections are bidirectional: each node receives likelihood messages from its children and a prior message from its parent. The root node treats the external Sequential Memory component as its parent node. Influences from external components are applied as extra likelihood messages ("virtual evidence"), a standard Bayesian network procedure. The full state of beliefs (concatenation of all nodes' posteriors) is sent out to various components. Compression progress is also measured and sent to the Serial Decision Maker for internal curiosity rewards.

The general idea behind the theorem is that of combining new data with prior predictions to produce a better estimate of the variable in question. $P(C)$ is the prior distribution over the class variable – *before* seeing the data, what is the probability of each hypothesis being the case? $P(C|E)$ is the posterior distribution over the class variable – *after* seeing the data, what is the probability of each hypothesis being the case? $P(E|C)$ is the conditional probability of seeing the evidence given a hypothesis – if a specific hypothesis is the true cause, what is the probability of seeing the current data? ($P(E|C)$ is also known as the "likelihood" function which can be written $L(C|E)$.) $P(E)$ is the probability of seeing the evidence, which is difficult to compute accurately but simply acts as a normalizing constant and can be ignored. We refer to the three main terms as the posterior, prior, and likelihood. Informally, we may restate Bayes' theorem as:

$$posterior \propto prior \times likelihood \tag{2}$$

Before we can apply Bayes' theorem to a data source, however, we need some kind of explicit representation of the hypothesis variable – a discrete set of hypotheses. However, in real world problems we are usually only given samples from the evidence variable in the form of real-valued vectors. The problem is thus learning to represent the hypotheses solely from the data samples, which allows conversion from raw data into probabilities. Such a conversion is a critical component of any embodied, Bayesian inference-based intelligent system operating in the real world. Fortunately, we can learn this representation automatically from the data in an unsupervised manner.

4.4 Generating Symbols via Unsupervised Learning

Any embodied intelligent agent must deal with raw sensory data in the form of real-valued vector quantities. The primary challenge is converting these data vectors into some "useful" internal symbolic representation. Useful for what? Generally, the symbolic representation should be useful for making decisions (e.g., in a reinforcement learning framework). We want it to represent the external world in the most *informative* way, thus enabling well-informed decisions. Fortunately, the notion of information transfer between numeric representations is well-defined in terms of probability theory.

We assume that the external world contains a discrete set of data sources, or causes. As these causes interact with their surroundings, they generate pat-

terns of energy (electromagnetic, acoustic, etc.) with are then transduced through the sensory apparatus (e.g., a camera), resulting in a particular data sample (an array of real values). The learning system then converts these data samples into an internal symbolic representation, where the symbols represent hypotheses about which external cause produced the sensed sample. (The internal symbols can be viewed as hypotheses, classes, or categories.)

The learning system faces three issues: 1) determining the number of causes in the world to represent internally, 2) determining the most probable cause for each sensory data pattern, and 3) adapting the internal symbols to represent the causes better. Here we handle the first issue simply by assuming that the number of causes in the world will often be very large – more than we are able to model with a symbolic representation. For this reason we usually choose to use the maximum possible number of hypotheses to represent given limited computing resources[10]. (In the worst case some of the internal symbols will merely be redundant.) The second issue requires an internal *probabilistic* representation of the external causes, described next. The third issue, described later, involves a learning process which adapts this symbolic representation according to some learning criterion.

4.4.1 Symbolic Representation

Before we discuss the learning process, we start with the hypothesis representation. We use a kernel mixture model of the data. Kernel mixture models are comprised of a discrete number of parametric kernel functions which can be mixed together to model an arbitrary distribution. (They are non-parametric models in the sense that they model distributions without any clear parametric form; they can also be called semi-parametric since they are built from a set of simpler parametric models.) Each kernel represents a single class/hypothesis of a multi-valued variable within the Bayesian inference framework. These hypotheses are assumed to be exhaustive and mutually exclusive, which is required for Bayesian inference ([44], Section 2.1.5).

We use N kernels total. Our kernel function of choice is the d-dimensional spherically-symmetric Gaussian kernel:

[10] We might find a parallel principle in the mammalian brain: the cerebral cortex is as large as possible given certain constraints (metabolic requirements, birth canal vs. head size, etc.)

$$p(E = \mathbf{e}|C = c_i) = \frac{1}{\left(2\pi\sigma_i^2\right)^{\frac{d}{2}}} \exp\left(-\frac{\|\mathbf{e} - \mathbf{w}_i\|^2}{2\sigma^2}\right) \tag{3}$$

for d-dimensional evidence vector $\mathbf{e}$ and kernel/class c_i with d-dimensional center vector $\mathbf{w}_i$ and standard deviation σ_i. Here we have defined the kernel function as a conditional density, or likelihood function, $p\left(E|C\right)$ (using lower-case p to represent a continuous probability *density*, not a discrete probability mass function). This likelihood function can be plugged into Bayes' theorem:

$$P\left(C = c_i|E = \mathbf{e}\right) = \frac{P(C)\, p(E = \mathbf{e}|C = c_i)}{p(E = \mathbf{e})} \tag{4}$$

This computes the posterior distribution over the hypotheses given the new evidence $\mathbf{e}$. The prior class distribution $P\left(C\right)$ represents the prior probability (before seeing a data/evidence sample) that each hypothesis was truly the cause of the evidence/sample. $p(E = \mathbf{e})$ is the data density function, the probability density at any given evidence vector $\mathbf{e}$. (This can be computed as a weighted mixture of the likelihood functions, but we will ignore it here because it only serves to normalize the products in the numerator to unit sum.)

At this point we have defined a particular symbolic representation, a kernel mixture model, which may be utilized in a probabilistic framework to convert raw data samples into a discrete probability distribution (the posterior distribution over hypotheses) (Figure 6). Now the difficulty is to adapt the kernel centers and standard deviations based on the data.

4.4.2 Data-driven Symbol Learning Based on Infomax

Earlier we stated that an ideal symbolic representation is one that is maximally informative about the raw data. Now that we have defined the form of our representation, we must determine how the symbols will adapt to fit the data. One popular technique for doing so is the principle of maximum information preservation (commonly called "infomax") [36]. The goal of infomax-based learning is to produce a representation of the data such that the mutual information between the input data variable (E in our case) and the output/representation variable (C in our case) is maximized. When

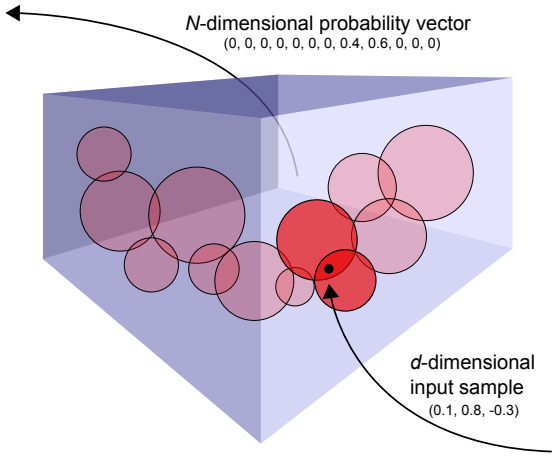


Figure 6: An adaptable kernel mixture model learns to represent the underlying data distribution. Here, d=3 and N is the number of kernels/hypotheses in the model. For each incoming data sample we can compute a set of component density/likelihood values, one for each kernel (see text). Then, given some prior distribution, we can convert the likelihoods to posterior probabilities, resulting in a discrete probability distribution (PMF) over the hypotheses. Essentially, this model learns a probabilistic symbolic representation from an arbitrary set of data samples.

this goal is achieved, we can assume that the learned representation is maximally informative, i.e. that the encoding processes loses as little information as possible.

In our case the mutual information between the evidence (continuous vector variable) and class (discrete variable) is:

$$I(E;C) = H(C) - H(C|E) \tag{5}$$

That is, the mutual information between E and C equals the entropy of C before seeing E minus the entropy of C after seeing E. Mutual information is symmetric ($I(E;C) = I(C;E)$) and always non-negative.

The Kullback-Leibler divergence between two probability distributions P and Q is defined as:

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \tag{6}$$

If we replace the distribution P with the posterior $P(C|E)$ and Q with the prior $P(C)$, then the mutual information between E and C can be defined in terms of Kullback-Leibler (KL) divergence as follows:

$$I(E;C) = \mathbb{E}_E\{D_{\mathrm{KL}}(P(C|E)\|P(C))\} \tag{7}$$

When viewed this way, the mutual information between E and C equals the expected value of the KL divergence between the prior and posterior. The KL divergence between these distributions represents the relative entropy between them, or the information gain after seeing a single sample from E. Thus, $I(E;C)$ is the expected information gain over all data samples in E.

As the goal of infomax-based learning is to maximize $I(C;E)$, it tries to maximize the expected information gain. Informally, it wants the prior and posterior distributions to be maximally different (as measured by KL divergence) on average, which is true when the prior distribution $P(C)$ is uniform and the posterior distribution $P(C|E)$ is totally concentrated on one value (a degenerate PMF). If this learning goal is achieved, each data sample provides maximal reduction in uncertainty/entropy concerning which class/hypothesis represents its true cause; the system receives maximal information gain per data sample.

In summary, by adopting the infomax learning principle, we focus on learning rules which do both of the following:

1. Generate a maximum entropy representation of C *before* seeing E – $H(C)$ – which corresponds to a uniform prior $P(C)$. For any given data sample, each class is equally probable.
2. Generate a minimum entropy representation of C *after* seeing E – $H(C|E)$ – which corresponds to a degenerate posterior $P(C|E)$. After seeing any given data sample, only one class is probable (minimal overlap).

In practice, these goals are difficult to achieve perfectly, but they still serve as an ideal learning target. Next we describe one specific learning algorithm which provides an approximation to the infomax approach.

4.5 A Robust Learning Kernel Mixture Model Algorithm

The kMER algorithm (kernel-based Maximum Entropy learning Rule), invented by Marc Van Hulle, is a recent method based the principle of equiprobabilistic kernel activation [24]. Equiprobabilism is the idea that each unit in a neural system should have equal probability of being active for any given data sample. Thus, equiprobabilism is a sort of maximum entropy goal. kMER implements equiprobabilistic learning by adapting binary thresholds on a set of neurons so that each neuron is activated by the same fraction of data samples. It essentially "tiles" the data space with a set of spheres, each containing the same number of samples. Then the resulting receptive fields can be treated as Gaussian kernels in order to compute probability densities/likelihoods.

Learning Rule Assume the input data space is d-dimensional. Each of N kernels K_i has both a d-dimensional center/weight vector $\mathbf{w}_i$ and a receptive field radius σ_i. For each kernel K_i there exists a "code membership function," which specifies whether the current input vector $\mathbf{v}$ is within its receptive field (whether it is "active"):

$$\mathbb{1}_i(\mathbf{v}) = \begin{cases} 1 & \text{if } \|\mathbf{v} - \mathbf{w_i}\| \leq \sigma_i \\ 0 & \text{if } \|\mathbf{v} - \mathbf{w_i}\| > \sigma_i \end{cases} \tag{8}$$

There also exists a "fuzzy code membership function":

$$\Xi_i(\mathbf{v}) = \frac{\mathbb{1}_i(\mathbf{v})}{\sum_{k=1}^{N} \mathbb{1}_k(\mathbf{v})} \tag{9}$$

which is essentially a normalized version of the code membership ($0 < \Xi_i(\mathbf{v}) < 1$ and $\sum_i \Xi_i(\mathbf{v}) = 1$).

kMER's kernels can be given a topographic influence in order to produce a topographic map, a relationship between the d-dimensional input space an a (usually 1- or 2-dimensional) "lattice space." This is achieved through the use of a set of lattice coordinates (i.e. in addition to each kernels d-dimensional center $\mathbf{w}_i$, it also has a 1- or 2-dimensional lattice position $\mathbf{r}_i$) and a neighborhood function (just like the classic SOM algorithm [28]), for example the Gaussian function:

$$\Lambda(i, j, \sigma_\Lambda) = \exp\left(-\frac{\|\mathbf{r}_i - \mathbf{r}_j\|}{2\sigma_\Lambda^2}\right) \tag{10}$$

where σ_Λ is the neighborhood range, which is usually set large initially, encompassing the entire lattice, and allowed to shrink to zero over time. The neighborhood function produces, for each pair of kernels, a value within $[0, 1]$ which represents the degree of closeness in lattice space. This function is applied in the center learning rule (see below) by "sharing" a kernel's activation with its neighbors in the lattice, ensuring that input samples which are nearby in the input space activate kernels which are nearby in lattice space. Figure 7 shows an example of topographic map formation. One major benefit of this topographic influence is much better initialization of kernel centers (e.g., along the directions of maximum variance), which tends to provide more consistent results at convergence.

kMER's incremental/on-line learning rules are as follows. For each input sample $\mathbf{v}$, update the center vectors $\mathbf{w}_i$:

$$\Delta\mathbf{w}_i = \eta \sum_j \Lambda(i, j, \sigma_\Lambda)\Xi_j(\mathbf{v})Sgn(\mathbf{v} - \mathbf{w}_i) \tag{11}$$

where η is the learning rate and $Sgn(.)$ is the (component-wise) signum function. The sum for each kernel i is taken over all N kernels j, a $O(N^2)$ computation. However, [24] (section 5.2.8) also provides an optimized version with much lower time complexity, which we use in our system.

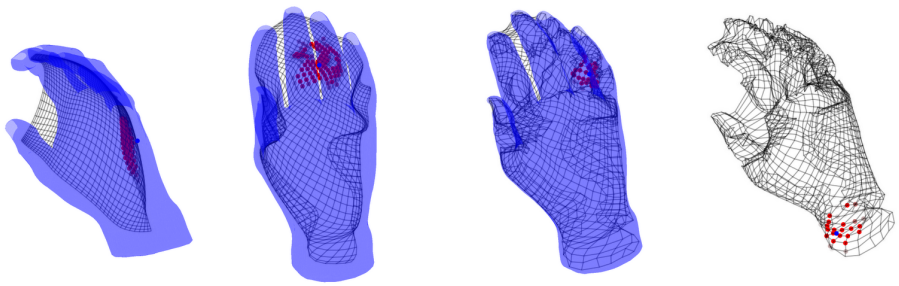


Figure 7: A sequence of images showing the development of a topographic map from 3-dimensional data space (samples from the hand surface) to a 2-dimensional lattice. Points that are nearby on the hand surface are nearby in the lattice. (Biologically speaking, nearby points on the skin are represented in adjacent locations within the 2-dimensional cortical sheet.) The final image shows the lattice alone without the hand data source.

Also for each input sample $\mathbf{v}$, update the receptive field radii σ_i:

$$\Delta\sigma_i = \begin{cases} -\eta & \text{if } \mathbb{1}_i(\mathbf{v}) = 1 \\ \eta \frac{\rho}{N-\rho} & \text{if } \mathbb{1}_i(\mathbf{v}) = 0 \end{cases} \tag{12}$$

where ρ is a scale factor that determines how many kernels will be active at once. At convergence, any given input sample will activate (fall within the receptive field of) ρ kernels. In other words, the probability that each kernel is activated by any given sample will be $\frac{\rho}{N}$. If $\rho = 1$, only one kernel will be active on average.

Essentially, each kernel center is moved towards the input sample in proportion to its fuzzy code membership Ξ_i. Since the fuzzy code membership values are normalized to unit sum, the active kernel centers must "share" the update strength. Each receptive field is increased in size by a small amount if the input sample is outside its radius but decreased by a large amount if the sample is within its radius. This leads to equal sharing of the data samples among kernels. (Intuitively, the kernels' receptive fields and data samples can be imagined as a set of funnels facing upwards, with marbles dropped from random locations above. The funnels try to move around, grow, and shrink in order to ensure that each one gets the same number of marbles.)

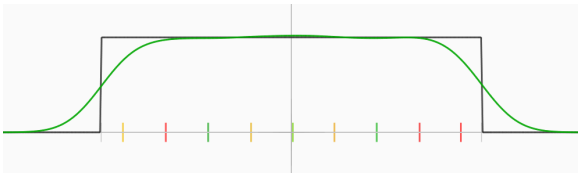


Figure 8: A kernel mixture model learning to represent a 1-dimensional uniform distribution. Both the actual data density and the learned density estimate are shown. The vertical bars represent the kernel centers.

kMER's learning rules are based on robust median statistics, so the resulting kernel centers converge on the median[11] of the input samples within its receptive field. This is accomplished through the use of constant direction factors (ignoring magnitude) in the incremental rules (note the use of $Sgn(.)$ in the center update rule).

Probabilistic Interpretation The learning rules above treat the kernel receptive fields as hyperspheres with radius σ_i. However, as described in [24] (section 5.4.5), it is possible to interpret these radii also as the standard deviations of hyperspherical d-dimensional Gaussian kernels (Equation 3). This produces a kernel mixture model, an estimate of the probability density function underlying the data. See Figures 8, 9, 10, and 11 for example results of applying kMER to various simple data distributions.

This completes the link between the kMER unsupervised learning rule and the probabilistic Bayesian framework. kMER learns to divide the input data samples evenly among a discrete number of receptive fields. We use the resulting receptive fields to compute Gaussian kernel functions which represent probability densities. Finally, we use the density estimates as likelihoods in Bayes' theorem.

Note that the prior distribution over the class variable, $P(C)$, is still undefined. Since we use a kernel to represent each possible class value/hypothesis, the prior probability $P(C = c_i)$ corresponds to the prior probability of any given data sample being caused by kernel i. Fortunately, rather than having to estimate these priors from the data (a relatively difficult problem), the

[11] As Van Hulle notes in [24], there is no unique definition of median for $d > 1$, so the median is defined per dimension: a given kernel receptive field contains equal numbers of input samples on either side of its center within each dimension (i.e. equal samples within $\mathbf{w}_{ij} - \sigma_i$ and $\mathbf{w}_{ij} + \sigma_i$ for kernel i and input dimension j).

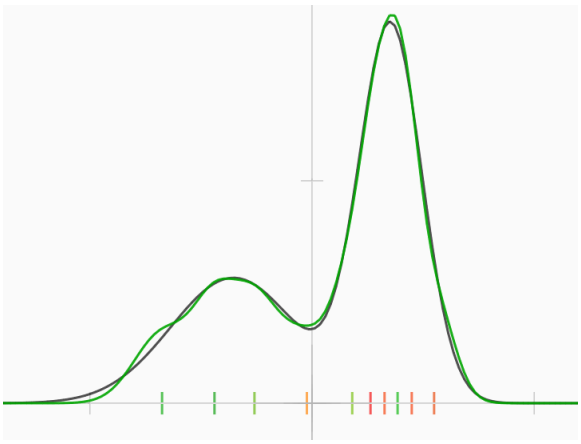


Figure 9: A kernel mixture model learning to represent a 1-dimensional distribution of two overlapping Gaussian sources. Both the actual data density and the learned density estimate are shown. The vertical bars represent the kernel centers.

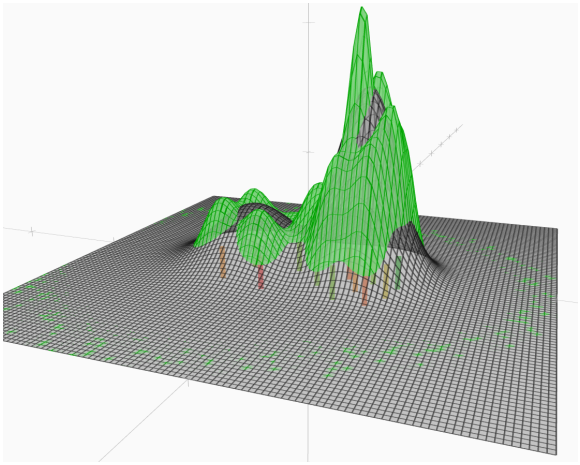


Figure 10: A kernel mixture model learning to represent a 2-dimensional distribution of two overlapping Gaussian sources. Both the actual data density and the learned density estimate are shown. The vertical bars represent the kernel centers.

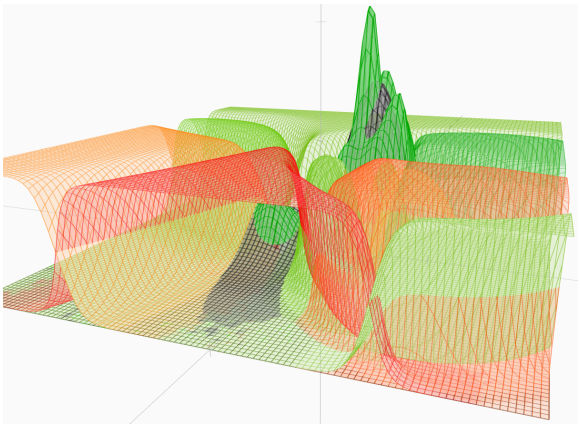


Figure 11: Visualization of the kernels' "probability surfaces," the probability that a data sample at a given location was "generated" by that kernel.

kMER learning rule automatically learns towards equal priors by design[12]. In general, assuming no other previous knowledge about the situation (given in the form of a prior distribution), we can assume that $P(C)$ is nearly uniform.

Recall from earlier the desire to find an infomax-based learning rule for our symbolic representation. kMER can be interpreted as an approximate infomax learning rule. Its equiprobabilistic basis achieves an (approximately) uniform prior distribution, so the random variable C has (approximately) maximum entropy. By setting the parameter ρ in Equation 12 so that the expected number of active kernels is one, the posterior distribution is degenerate (concentrated on one value), so that the random variable $P(C|E)$ has (approximately) minimum entropy. These two aspects of the learning rule make it useful as an approximation of infomax, generating a representation of C which extracts very high information content from each data sample.

Summary kMER has many attractive properties:

- kMER's focus on median-based learning endows it with a quality of robustness, making it much less affected by outliers in the input data

[12]However, note that the equiprobabilism principle is a heuristic for achieving a perfect equal prior, maximum entropy density model (Van Hulle, personal communication, 2009). Even though the receptive fields in the learning rule contain equal numbers of data samples, the actual prior probabilities in the kernel density estimate will not be equal in general.

compared to mean-based learning rules. In comparison, the weight learning rule in the SOM algorithm [28], which looks very similar to the kMER rule, does not use $Sgn(.)$ but instead uses the full direction and magnitude of the error vector between the input and weight vectors; its neurons learn toward the *mean* of the data samples that activate them.

- It allows topographic map formation. The use of a kernel lattice and neighborhood function constrain the learned kernel centers to a topographic map between the input and lattice space.

- kMER's binary spherical receptive fields make it robust in high-dimensional spaces. If we simply used spherical Gaussian kernels directly in the learning rules (i.e. if activation/code membership were based on the Gaussian function), the results get worse as d increases because most of the Gaussian volume shifts farther towards its tails[13]. This presents significant numerical problems because computing the d-dimensional Gaussian function is much less accurate in its extreme tails than near its center. In contrast, kMER's binary spherical receptive fields circumvent this issue. Even though high-dimensional hyperspheres also contain most of their volume in a thin shell near their radii, it is much simpler to compute stably whether a data sample is inside or outside of a sphere. This feature helps kMER scale better to high dimensions in comparison with other common algorithms, like the maximum likelihood-based expectation-maximization (EM) algorithm [14].

- The underlying principle of equiprobabilism ensures that each kernel takes an equal role in modeling the data. It reliably produces a weight density proportional to the input data density (a "faithful representation"), whereas the SOM algorithm [28] does not (for a 1-dimensional lattice, SOM's weight density $p(\mathbf{w}_i)$ is proportional to $p(\mathbf{v})^{\frac{2}{3}}$, assuming infinite neurons) ([24], Section 4.1). A related problem is that the SOM algorithm often produces "dead" units, wasted neurons which are initialized outside the data distribution and never get used.

- kMER's learning rules can be interpreted as an approximate infomax learning algorithm, and its learned kernel parameters can be used to compute probability density estimates/likelihoods, a probabilistic symbolic representation of the input data.

[13] For example, a 1-dimensional Gaussian contains 6% of its volume beyond 2σ, compared to 15% in 2 dimensions, 27% in 3 dimensions, etc.

The next section discusses how we may combine unsupervised learning with Bayesian inference in a way that scales to high-dimensional data spaces.

4.6 Hierarchical Empirical Bayesian Network

At this point we might assume that the combination of an unsupervised learning algorithm (like kMER) embedded within a probabilistic framework (Bayesian inference) is enough to provide a good representation of any sensory data. However, one practical issue remains concerning high-dimensional data: the "curse of dimensionality," a very general problem facing many machine learning algorithms.

Any kind of adaptable representation of a data space requires that the data distribution is sampled at a certain resolution. If the distance between samples is too great, some features in that intervening space will be missed. The "curse" is that as the dimensionality d of the data space increases, the number of data samples needed to fill its volume with a fixed sampling resolution increases exponentially. As an example, a learning algorithm might need 100 samples from a 1-dimensional space, 10,000 from a 2-dimensional space, 1,000,000 from a 3-dimensional space... However, it is almost always the case that the required number of samples is simply not available; instead, the sampling resolution suffers as d increases. There is simply too much space between the available samples, and learning an accurate representation becomes increasingly difficult.

We choose to handle this problem by maintaining a constant value for d by breaking up high-dimensional data spaces into several spaces of smaller dimensionality. For example, a 100 x 100 pixel image space (10,000 dimensions) can be split into 200 separate 50-dimensional spaces. Each of these smaller spaces can then be modeled with its own kernel mixture model. How then can we combine the results of these separate models? We treat each one as a node in a Bayesian network, organized hierarchically so that the results of many nodes, each focused on a small patch of the data space, converge onto a single root node which summarizes the entire space.

A Bayesian network is a graphical model containing a set of nodes, each representing a random variable, and a set of connections between nodes which define relationships between the nodes as conditional probabilities. It acts as a distributed representation of Bayes' theorem, where each node computes probabilities regarding a small part of the world. Within each

node, "likelihood messages" (new data/evidence) are received from child nodes, "prior messages" (prior distributions) are received from parent nodes, and the two messages are combined to compute the posterior distribution ("beliefs") for the node's variable, as in Equation 2.

Figure 12 shows a single node with two children and one parent. It contains a kernel mixture model which learns the conditional probability distribution, a mapping between the node's values (it's "belief space") and the child's values. The node's likelihood message is computed as a combination of the likelihood messages provided by the children, where the child likelihoods are concatenated and provided as input data to the node's kernel mixture model. Essentially, the node learns to model the belief space of its children. Priors to be sent to the children can then be computed as a weighted mixture of the node's kernel center vectors (weighted by the node's own prior from its parent).

Bayesian networks assume statistical independence among all variables that are not connected. This means that each node is assumed to represent a piece of the world which is causally unrelated to all other nodes except for its parent or child nodes (assuming a hierarchical network). We address this requirement by breaking up the input data space (assumed to be presented *topographically*) into small patches, where neighboring nodes are assumed to be modeling different causes simply because their data comes from physically separate sensors. This does not imply total statistical independence, but we assume that it is good enough for most purposes.

Why is it important for the data to be presented topographically? As described earlier, a topographic map is a transformation from one space to another (possibly of different dimensionality), where two points that are similar in one space are similar in the other space. Each patch of the full sensory array should represent a non-overlapping receptive field, the output of sensors which are nearby in physical space, so that a node which models that data can find statistical regularities in its data samples. (For example, in images sampled from a digital camera, it is much easier to find patterns in nearby pixels than from a collection of pixels chosen from random locations within the image.) Topographic representations make it easier to divide the data by their spatial arrangement into statistically independent variables, a crucial ingredient for Bayesian networks.

Biologically speaking, if we assume that each cortical column is "trying" to represent a set of input data different from its neighboring columns (i.e. representing non-overlapping receptive fields), then the columns should learn

to represent variables that are conditionally independent, given their common parent column, which is exactly what is required in a Bayesian network. Furthermore, if we assuming each column's set of minicolumns learns based on an infomax-like principle so that each minicolumn represents a non-redundant feature, then the minicolumns will be statistically independent of each other, which is also required for Bayesian network node (i.e. each of a node's values must be mutually exclusive).

To update our hierarchical network, we use Pearl's Bayesian Belief Propagation (BBP) algorithm for tree-structured networks ([44] Section 4.2). BBP is a standard Bayesian network updating algorithm with the following properties:

- Efficient storage and processing requirements: $O(n^2 + mn + 2n)$ real values and $O(2n^2 + mn + 2n)$ multiplications per node for a tree with m children per parent and n values per node.
- Network updates can be performed synchronously or asynchronously, enabling parallelized implementations for multi-processor computers.
- Any new information (new data/likelihood messages from below or new root prior messages from above) propagates through the network in a single pass, with the required number of updates proportional to the network's diameter.

If the likelihood messages for the bottom-level nodes come from the raw data, what provides the prior message for the root node? This question is largely unanswered in [44], which simply calls the root prior "the background knowledge remaining unexplicated." In other words, it must be estimated somehow.

One possibility is to use a simple naive prior, where the root node's prior distribution equals its previous-step posterior distribution[14]. By setting the root's prior probabilities based on the previous state of the world, this scheme implicitly assumes, naively, that things never change. However, given no other knowledge, this isn't the worst strategy. It is probably better than simply using a uniform prior distribution since it captures some prior knowledge about the world.

What would be the ideal source of root node priors? It would be a perfect predictor which, based on current beliefs, perfectly predicts the next-step

[14]Note that this is only allowed for the root node. All other nodes must get their priors solely from their parent nodes. (See [44] p. 164.)

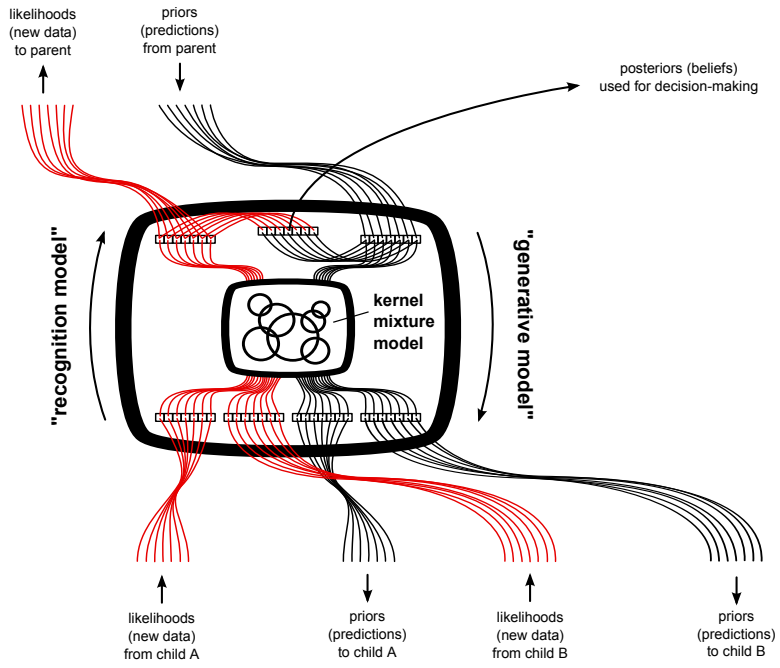


Figure 12: A single node in the belief network representing some variable. This node combines data from multiple child nodes with predictions from a parent node. The result is a set of beliefs about the state of its variable which can be used elsewhere for making decisions.

distribution. (This is in contrast to the naive prior which would predict that things stay the same as the current state.) This might not be possible in practice, but it should be possible to improve over the naive prior. We can use a sequential prediction generator for this role which learns from experience. This is precisely the purpose of the hippocampus-inspired Sequential Memory component described later.

4.7 Measuring Model Improvements

Earlier we defined the learning objectives for our system, one of which was improving the world model. How should model improvements be measured? Here we define two possible methods: data prediction error and information gain. The choice of which method to use will be decided after running experiments. It may become apparent that one method is more useful or even that they both represent the same measurement in different forms.

4.7.1 Data Prediction Error

One way to measure the model's prediction performance is the mean squared prediction error of its sensory data. At each instant, it is possible to compare the actual incoming sensory data with predictions generated by the Bayesian network. (Sensory predictions are simply the bottom-level priors sent out from the nodes nearest to the raw sensory data. Biologically, this would probably occur in the thalamus.) The mean squared error is then computed from the component-wise errors between these two arrays. This measure summarizes many aspects of the model's prediction performance, including simple reconstruction errors due to compression (e.g., looking at an unchanging visual scene) and sequential prediction errors (e.g., predicting the path of a moving object).

4.7.2 Information Gain

Within the Bayesian inference framework, we can measure the information gain provided by a data sample, a form of model improvement. Naturally, this measurement is provided as the KL divergence (Equation 6) between the prior and posterior distributions. This measure summarizes the information gained from seeing new data (likelihood array). This general approach

has been used by other researchers for experimental design [35], to motivate "reinforcement driven information acquisition" (optimal sequences of experiments) within learning systems [55], and to model human attention [25]. In the Sensorimotor Belief Network we can measure information gain at the root node (i.e. the difference between the multimodal node's prior and posterior) or averaged over all nodes.

Instead of using KL divergence, we might instead use Jensen-Shannon (JS) divergence [34]. The JS divergence between two probability distributions P and Q (assuming equal weighting) is defined as:

$$D_{\mathrm{JS}}(P\|Q) = \frac{1}{2}D_{\mathrm{KL}}(P\|M) + \frac{1}{2}D_{\mathrm{KL}}(Q\|M) \tag{13}$$

$$M = \frac{1}{2}(P + Q) \tag{14}$$

Intuitively, the JS divergence of two distributions is the mean KL divergence from the mean distribution. Similar to the mutual information between a data and class variable, JS divergence is the expected KL divergence between posterior and prior. It has several benefits over KL divergence: it is bounded to a finite range $[0, -\log(\frac{1}{2})]$ (arbitrary base), it is well-defined for all combinations of zero and non-zero probability values, it is the square root of a true metric, and it is symmetric: $D_{\mathrm{JS}}(P\|Q) = D_{\mathrm{JS}}(Q\|P)$.

4.7.3 Curiosity Rewards

We use the measured model improvements to motivate exploratory behavior. The measured improvements are sent to the Serial Decision Maker as internal curiosity rewards, reinforcing situations and actions that lead to further model improvements.

The combination of a powerful world model with a curiosity drive enables an interesting synergy. It enables a feedback loop where the entity is driven to explore new situations, exploration improves the entity's world model, and the entity becomes curious about even more complex situations[15]. The learner is continually driven to satisfy its curiosity in increasingly sophisticated ways.

[15]Interestingly, mammals with larger relative brain sizes (usually corresponding to larger cerebral cortices) tend to exhibit more advanced play behavior ([11], Table 8.1).

4.8 Producing Motor Outputs

So far we have discussed how our system processes incoming sensory data, but how does it generate outgoing motor control signals? To answer this question, we again look for inspiration from biology. Based on the biological evidence reviewed at the beginning of this chapter, we assume that any motor-capable region of the Sensorimotor Belief Network should sense and control similar variables. For example, if a motor-related belief node receives inputs regarding a joint angle from a robotic arm, it should also send out a control signal encoding the *desired* joint angle. If another node receives the current force output from a servo motor, it should send out the *desired* force output. This scheme allows the motor belief nodes to learn a representation of motor space which is also used directly as a basis for control.

4.9 Bootstrapped Motor Development via Reflexes

In order to learn a useful representation of motor space, it is important that the motor-related variables are sampled from a large range of possible values. If an embodied system (e.g., a humanoid robot) remains totally stationary, its sample distribution for all motor-related variables (like joint angles) will not be representative of the full motor space. This situation is highly detrimental to the decision making process (described in a later chapter) which must choose, at each instant, from among a set of learned motor gestures. If all motor configurations have, through learning, converged to a single gesture, the decision making process will become effectively stuck, always choosing from a set of equivalent motor configurations.

During the initial phase of development we help bootstrap the motor learning process with a set of built-in reflexes. In general we do not know the ideal set of reflexes for any given body, so we simply use a neural network with random connection weights mapping sensory inputs directly to motor outputs[16]. This reflex network takes full control of the body initially, pushing the effectors through their full range of motion. Over time we ramp down the effect of the reflexes, simultaneously ramping up the effects of the decision making system (described later). The idea is that an embodied system will begin its lifetime fully controlled by its reflexes and gradually gain

[16]For a given body, it might be better to use artificial evolution to evolve a good set of reflexes over the course of several generations. This might speed up the initial learning process. However, we assume that this is not strictly necessary in general.

the ability to make voluntary actions. This scheme has biological support: cat motor cortex does not initially influence spinal motoneuron activity (i.e. the corticospinal/pyramidal tract), only gradually becoming active over the course of several weeks after birth [10].

We now speculate briefly about bootstrapped motor development animals. It seems that a general distinction exists concerning two sources of motor control: reflex-based (genetically-determined, including central pattern generators, etc.) vs. voluntary control. Insects appear to be solely driven by reflex-like circuits with little or no adaptable voluntary control. Amphibians and reptiles heavily depend on reflex control but are also able to switch voluntarily between various automatic control systems. Mammals often begin life under full reflex control but learn to increase voluntary control after some time. The initial set of behaviors is often crucial within the expected environment after birth (for example, gazelles are able to stand and run almost immediately – there is no time to learn from scratch in the presence of predators), but voluntary behaviors are learned eventually. Humans, as an extreme example, require a relatively simple initial set of reflexes at birth because, presumably, our parents will protect and provide for us as we develop voluntary control. Perhaps many of the reflexes we have at birth are merely for bootstrapping voluntary motor development; seemingly random flailing drives our effectors through their full range, training our motor cortex to represent the widest range of configurations. Maybe the parental safe haven is necessary for such advanced learning of voluntary control; more precisely, it could be that the degree of voluntary control within a species is proportional to the amount of parental support available after birth.

5 Sequential Memory

After describing a hierarchical network for Bayesian inference, there is remains the outstanding question of where the root node's priors are produced. Indeed, the source of priors for Bayesian inference is a tricky issue in general. We propose to approach this problem by taking inspiration from the brain's hippocampus. Its connectivity indicates an intimate relationship with the cerebral cortex, possessing reciprocal connections with the multimodal cortical regions[17].

[17]Interestingly, various comparative studies with vertebrate brains hint that most of the cerebral cortex (6-layered "neocortex/isocortex") essentially appeared as an outgrowth of the evolutionarily older olfactory/hippocampal region (3-layered "allocortex") [12].

Functionally, this component in our architecture provides prior knowledge about what the Sensorimotor Belief Network should expect to see next. This includes sequential prediction capabilities, where past learned patterns can be recalled sequentially in order to bias expectations about future sensory data. For these reasons we call this component the Sequential Memory.

5.1 Inspiration from the Hippocampus

Here we look at the structure and function of the hippocampus[18] region in the brain. Its patterns of input and output connectivity provide some direction in designing the Sequential Memory component.

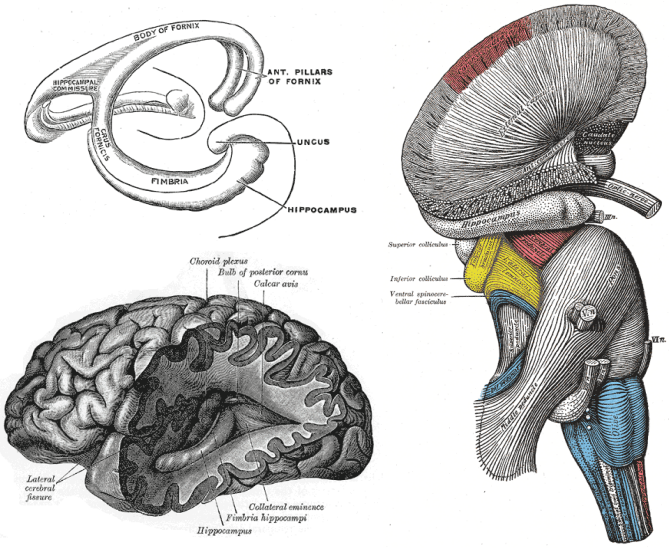


Figure 13: The hippocampus shown in isolation, hidden within the cerebral hemispheres, and in relation to the midbrain. From Gray's Anatomy (public domain).

Anatomy The hippocampus receives major inputs from and sends major outputs back to the entorhinal cortex, a region between the hippocampus and the multimodal areas of sensorimotor cortex. It also has a major output pathway called the fornix (from areas CA1 and CA3) which projects to

[18]The word hippocampus is Latin for "seahorse" due to its shape.

various external brain regions including the striatum of the basal ganglia (see [1] Figure 21-10). Internally, a region called the dentate gyrus forms a major part of the "perforant pathway" through the hippocampus. It is the largest hippocampal region by cell count, containing roughly 1,000,000 granule cells (vs. only 200,000 entorhinal cortex input cells) in the rat hippocampus [4]. Signals flow in from the entorhinal cortex, through the dentate gyrus granule cells, then to areas CA3/CA1, and to the subiculum before returning back to entorhinal cortex and out to the original external sources ([27] Figure 62-5).

Function The hippocampus is commonly known as being involved in forming and recalling episodic memories, such as sequential patterns. Much has been learned about its function from various memory disorders and lesions. It appears that the "content" of the memories are stored elsewhere in cerebral cortex, but the hippocampus acts to tie them together into a cohesive event or sequence.

Computational Models Besides the biological evidence already reviewed, it is helpful to study computational models by other researchers. Atallah et al [6] suggest that various brain regions are suited for different purposes based on computational trade-offs. The hippocampus is specialized for quickly learning specific details, i.e. "snapshots," while the posterior cortex slowly learns statistical regularities. The authors discuss a connection from hippocampus to the ventral striatum of the basal ganglia (possibly the fornix pathway).

Banquet et al [7] proposed a cognitive architecture for robot control based on the dentate gyrus and CA3/CA1 regions. They suggest a spectral timing model for the dentate gyrus in which each granule cell responds to its inputs with a different temporal response pattern based on some physiological variation within the cells. The result is a set of "spectral batteries" which provide a recent history trace, enabling other regions to learn precisely-timed responses based on their composite temporal pattern.

5.2 Functional Abstraction

We combine inspiration from the known structure and function of the hippocampus with various computational needs present in our architecture to

design a Sequential Memory component (Figure 14). Functionally, the Sequential Memory stores and recalls sequences of patterns present in the sensorimotor root node, and it provides a temporal state representation to the Serial Decision Maker for learning precise reward predictions.

The multimodal root node of the Sensorimotor Belief Network has no parent node to provide its prior distribution. This prior should be generated based on prior experience about the sequential patterns present in the multimodal root node's belief state. We see the connection between multimodal cortical areas and the entorhinal input/output area of the hippocampus as fulfilling this role.

Learning to predict sequential patterns can be accomplished in various ways [40]. Without regard to biological plausibility, we choose to handle this issue by generating a type of spectral timing system called a tapped-delay line memory (essentially a scrolling marquee of recent events), then training a neural network predictor based on this temporal pattern. This is similar to the spectral timing dentate gyrus model used in [7]. We essentially assume that the purpose of the dentate gyrus is to provide a short-term memory pattern of recent events (a distributed "clock"), that the CA3/CA1 areas convert this pattern to a more linearly-separable version, and that the connections back to entorhinal cortex learn to predict the next input pattern.

Besides sequential predictions, learning context-specific reward predictions (within the Serial Decision Maker component) also requires a temporal pattern of recent events. We assume that the fornix connection from the hippocampus provides a copy of its internal clock (dentate gyrus), enabling the ventral striatum to make temporally accurate reward predictions. In our system we send a copy of the Sequential Memory's temporal pattern array to the Serial Decision Maker for this purpose (Figure 14).

5.3 Dynamic Reconstruction Algorithm

Our approach to the sequential prediction problem is similar to the algorithm described in [19] (Section 13.11) for solving the "dynamic reconstruction" problem (i.e. learning to reproduce a time-varying signal). Namely, we use a delay-line memory representation of the recent history of inputs, plus a learning neural network one-step predictor (a single-layer neural network). This scheme, capable of learning relatively complex temporal patterns, is shown in Figure 14. Theoretically, this setup allows recursive prediction by

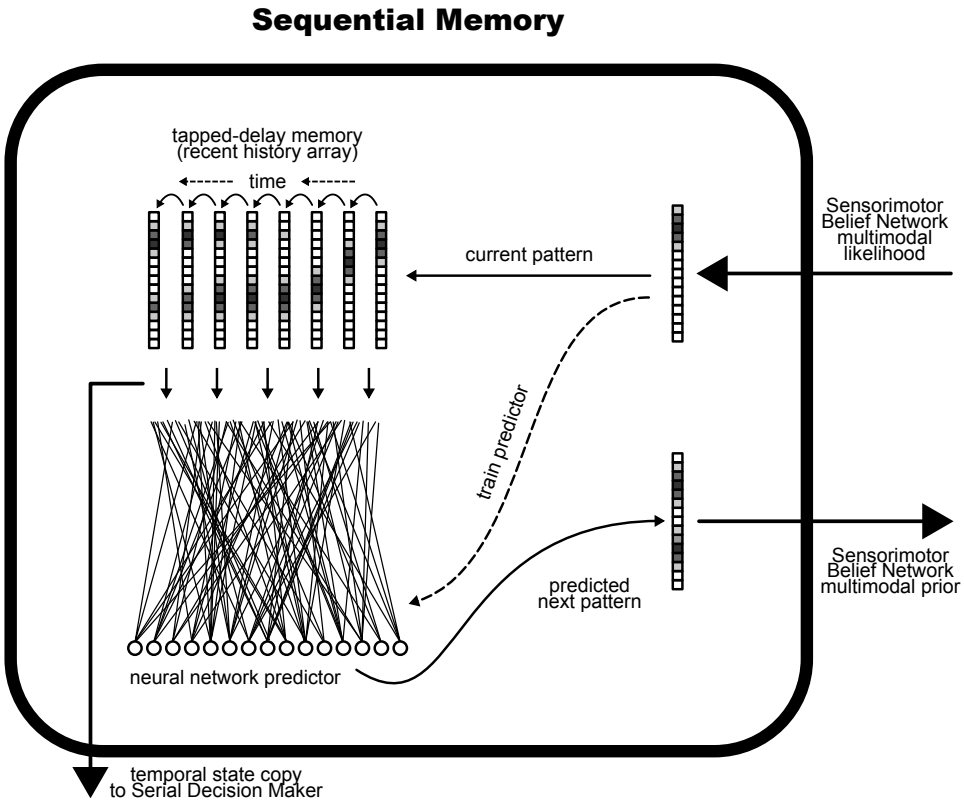


Figure 14: The Sequential Memory component. Takes the current multimodal belief state from the Sensorimotor Belief Network and encodes its temporal activity using a tapped-delay line array. This array functions like a scrolling marquee, holding a short history of recent activity. This temporal array is then used as input to a neural network predictor which tries to predict the next input pattern based on past activity. This neural network is trained with the actual input patterns. The predicted patterns are sent out as the prior for the Sensorimotor Belief Network multimodal root node. Also, a copy of the temporal array is sent out to the Serial Decision Maker to help make reward predictions.

feeding its predictions back as inputs, thus operating as a recurrent network.

6 Serial Decision Maker

Until now we have discussed components within the architecture that involve representing the world. This chapter concerns the decision making process: using the results of that representation of the world to select from a set of potential actions. This inevitably involves the concepts of reinforcement and value, which provide a basis for comparing actions in a given context. We can generally divide the decision making process into two separate procedures: learning the value/utility of a given situation, and learning the appropriate action to take in a given situation. In theoretical reinforcement learning these procedures correspond to learning the value function and policy. In the brain they both appear to occur within the basal ganglia. Indeed, there are interesting connections between the basal ganglia macro structure and so-called "actor-critic" reinforcement learning architectures, plus a strong similarity between the firing patterns of midbrain dopamine neurons and theoretical reward prediction error signals (e.g., temporal difference learning).

6.1 Inspiration from the Basal Ganglia

The basal ganglia are a collection of nuclei located deep within the cerebral hemispheres. Their gross anatomy and circuitry are relatively well known, and many of their functions have been characterized. However, much research remains to be done in determining precisely their internal mechanisms. Recent computational models of dopamine neuron activity have proven to be very influential in understanding their involvement in reward-based learning.

Anatomy The striatum region is the primary input pathway of the basal ganglia. Striatal inputs come from all regions of cerebral cortex (except primary visual and primary auditory), including sensorimotor regions and prefrontal cortex (see [48] Figure 17.4). The major output pathway is through the globus pallidus, which projects to the thalamus and on to most areas of the frontal cortex (including motor-related and working memory areas). Dopamine neurons in the midbrain also send connections to various basal ganglia nuclei.

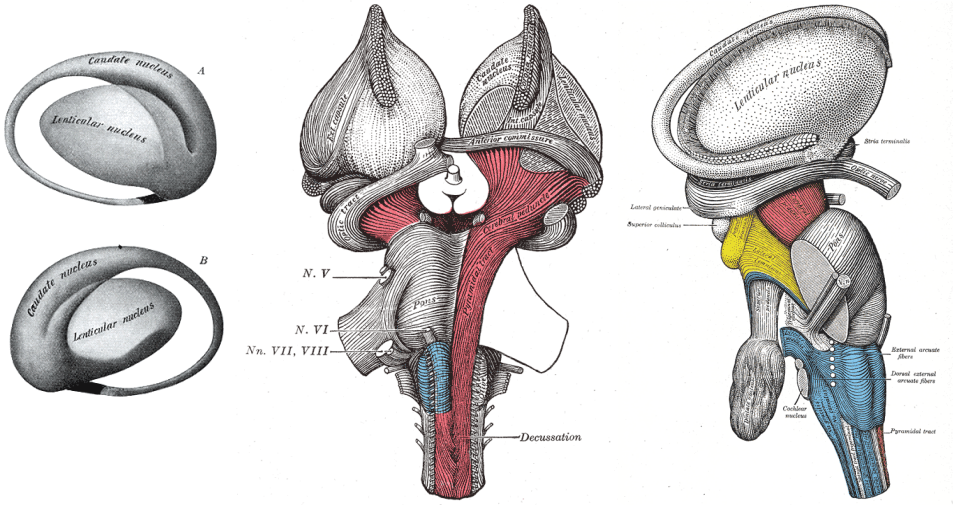


Figure 15: Illustrations of the caudate and lenticular nuclei, major pieces of the basal ganglia. The major input region is the striatum, comprised of the caudate and putamen (part of the lenticular nucleus). Major outputs come from the globus pallidus (another part of the lenticular nucleus). From Gray's Anatomy (public domain).

Function It appears that the functional roles of the basal ganglia fall into two categories: associative learning of context-dependent value, and selecting context-appropriate actions. The latter seems to depend on the former: proper action selection requires first learning the value of various actions. [50] discusses the general problem of action selection, describing the basal ganglia as the primary means of controlling access to various limited resources. The same authors also implemented a high-level basal ganglia model in software, embodied it within a mobile robot, and let it provide context-dependent action selection functionality to the robot [46].

Models In the reinforcement learning literature, an "actor-critic" architecture is a functional division of the reinforcement learning process into two components: an actor which chooses actions and a critic which provides instructive feedback (both to the actor and to itself) based on reinforcements. In [22] a parallel is made between such actor-critic architectures and the basal ganglia, where the striosomal modules within the striatum are treated

as part of a critic system, and the matrisome/matrix modules are treated as part of the actor.

One intriguing recent model involving basal ganglia function is the temporal difference (TD) model of midbrain dopamine neuron activity, reviewed in [56]. This model relates the purely theoretical TD learning algorithm [66] with the activity of dopamine neurons in monkeys which appear to encode reward prediction errors. This connection provides a basis for understanding reinforcement learning in terms of biological constraints.

A more recent model of dopamine neuron activity is the "primary value learned value" (PVLV) model by O'Reilly et al [42]. This model attempts to overcome limitations of the TD-based model by providing separate mechanisms for learning primary rewards (PV) and conditioned stimuli (LV).

6.2 Functional Abstraction

In designing an abstract decision making component (the Serial Decision Maker, Figure 16), we follow the general functional division of the basal ganglia into context-dependent value learning and context-dependent action selection (i.e. an actor-critic architecture). This component provides an action selection system that learns from reinforcement, including external and curiosity-based internal rewards. Its "actor" is a discrete serial switching mechanism (a competitive winner-take-all network) which enables at each instant one possible "action" out of a finite set. This set of actions includes two types: 1) motor actions, which influence motor-related beliefs in the Sensorimotor Belief Network, and 2) working memory actions, which update memory cells in the Working Memory component (i.e. enabling input/output gating of information to/from temporary storage). The "critic" is based on the PVLV model of dopamine activity, which trains the actor. Both actor and critic utilize single-layer neural networks to learn their context-dependent mappings.

The inputs to the Serial Decision Maker include the contents of various other components, including the Sensorimotor Belief Network state, Working Memory contents, and the Sequential Memory's internal temporal pattern. Note that the sensorimotor beliefs represent a more linearly separable version of the raw sensory data, which enables simpler learning in the linear neural networks here.

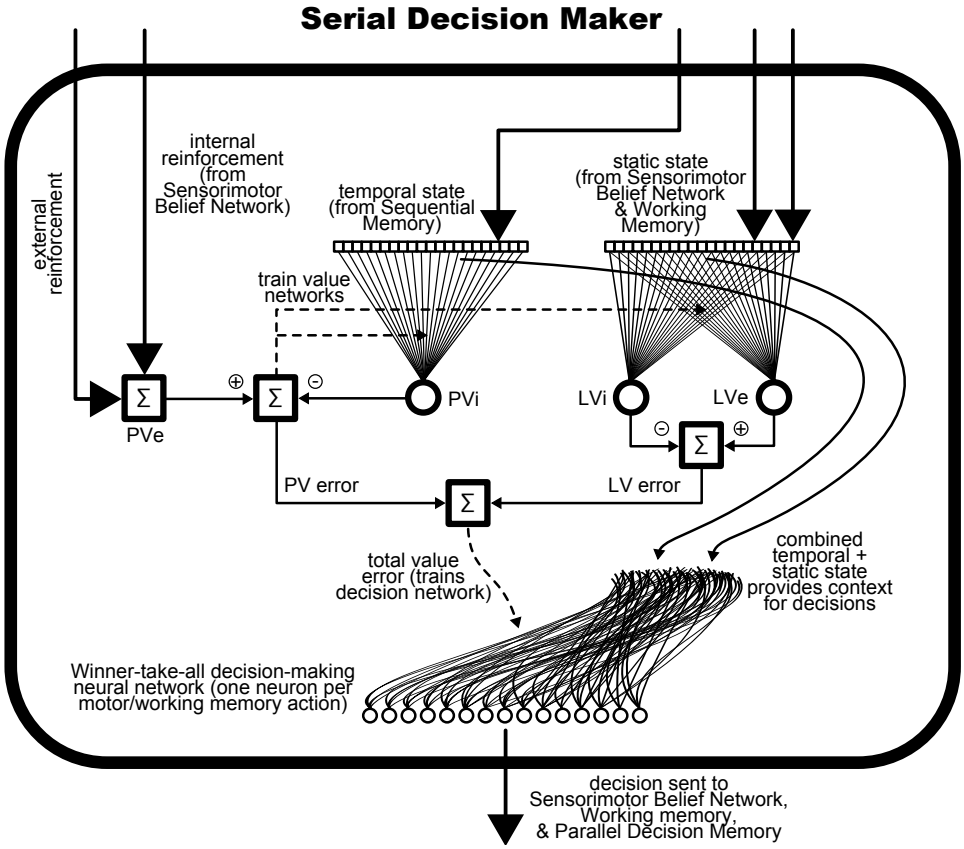


Figure 16: The Serial Decision Maker component. The action selection mechanism is the winner-take-all network at the bottom which chooses actions based on the entire input context. The rest of the components are part of the PVLV model of dopamine activity, whose output trains the action selection network.

7 Parallel Decision Memory

We assume that in most realistic environments, many of the Serial Decision Maker's actions will be frequently chosen in certain contexts, and therefore will constitute a significant waste of resources. As the Serial Decision Maker is a sort of bottleneck, focusing on only one decision at a time, it seems helpful (even essential) to store well-learned decisions elsewhere to be executed automatically. This is the role of the Parallel Decision Memory component. It constantly watches the Serial Decision Maker's actions and learns to take them over after many trials. If environmental demands change so that the desired action in a given context is different, the Parallel Decision Memory can adapt to fulfill the new requirements. Furthermore, it can drive many targets in parallel, enabling complex motor output patterns with little intervention from the Serial Decision Maker[19].

For this component we take inspiration from the cerebellum and its relationship with the cerebral cortex and basal ganglia.

7.1 Inspiration from the Cerebellum

The cerebellum contains half of the brain's neurons crammed into only 10% of its total volume. Despite its massive cell counts, its structure is very repetitive: a few well-characterized circuits run through the cerebellum, making it possible to study its overall function by focusing on a small number of neural pathways. It seems possible that the cerebellum performs a single, relatively simple function in parallel on massive arrays of data.

Anatomy Most cerebellar inputs come through the massive cortico-pontine-granule cell pathway (the mossy fiber system). A relatively smaller input pathway travels through the red nucleus-inferior olive-Purkinje cell route (the climbing fiber system). Multiple independent point-to-point loops run from the cerebral cortex through the red nucleus, inferior olive, cerebellar Purkinje cells, deep cerebellar nuclei, thalamus, and back to cerebral cortex.

[19]Such parallel operation seems necessary in many everyday situations. Imagine the complexity of many tasks we take for granted – maintaining upright posture (simultaneously controlling multiple leg and torso muscles) while using our arms for an unrelated task, like painting. Our serial decision making process cannot handle hundreds of motor decisions every second, and why should it? If it can offload its most common decisions to an external system (the cerebellum), it can instead focus on truly novel decisions.

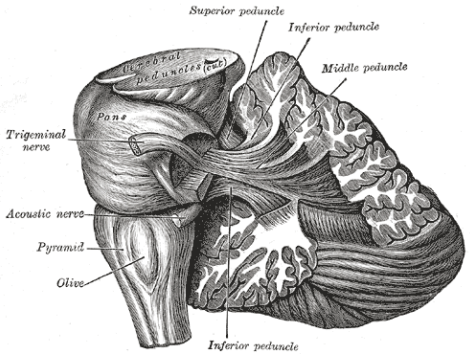


Figure 17: Cerebellum cross section. From Gray's Anatomy (public domain).

Essentially, only a few distinct circuits run through the cerebellum, but they are replicated millions of times.

Cerebellar inputs and outputs appear to be organized similarly to the basal ganglia ([1] Figure 15-24): both receive inputs from most of the cerebral cortex and send outputs back to the frontal cortex (motor and working memory areas).

Function The cerebellum exerts its effect by modulating motor cortical activity. It is involved generally in motor control and learning, but more specifically in conditioned response learning, temporally-specific associative learning, and timing of movement initiation and termination. Recently it has been implicated in cognitive tasks like word retrieval [1]. It appears to be well-suited to "take over" well-learned motor sequences [21].

Models The models of Marr [39] and Albus [2] interpret the mossy fiber input system as carrying a type of distributed context representation, and the climbing fiber inputs provide teacher signals based on errors between actual and desired motor performance (but see other interpretations of climbing fiber activity reviewed in [9]). The Albus model in particular describes the mossy fiber system in terms of a Perceptron-like pattern-recognition system. [45] provided a computational cerebellum model within a larger context, including the complete circuit through the basal ganglia, thalamus, and inferior olive.

7.2 Functional Abstraction

The Parallel Decision Memory component (Figure 18) enables parallel motor automation, a sort of shadow of the Serial Decision Maker. It observes commonly-chosen actions (by the Serial Decision Maker) and learns to perform them, offloading decisions from the action-selection bottleneck. In addition, whereas the Serial Decision Maker operates as a serial process, the Parallel Decision Memory provides parallel outputs. It is implemented as a massive single-layer neural network using simple supervised learning. It learns a mapping from the current context (Sensorimotor Belief Network and Working Memory state) to the desired control signal vector, and its teaching signal is the Serial Decision Maker's current choice. Both decision-making components essentially receive the same contextual inputs and exert their influence on the same targets.

8 Working Memory

In many realistic scenarios it is beneficial to be able to hold information in mind temporarily and later let that information influence action. For example, consider the task of hearing a telephone number and dialing it: the number is spoken, the sound of the spoken digits fades within milliseconds, but the listener can *actively* store and maintain those digits internally, then later *actively* allow that internal representation to influence action (e.g., dialing the phone number). Notice the two actions required: active storing and active retrieving. Essentially, working memory seems to be treated as a set of actions, similar to motor commands.

Although the prefrontal cortex is not as well-characterized as other brain regions, we can still gain much insight by studying its anatomy and assumed behavioral roles.

8.1 Inspiration from the Prefrontal Cortex

The prefrontal cortex, which has greatly expanded in humans relative to other mammals, appears to provide us with a host of executive functions, like planning, decision making, prioritization, and judgment, all of which are required for strategy formation and complex social situations. It is interesting to speculate about the evolution of cortical regions. It is as if the

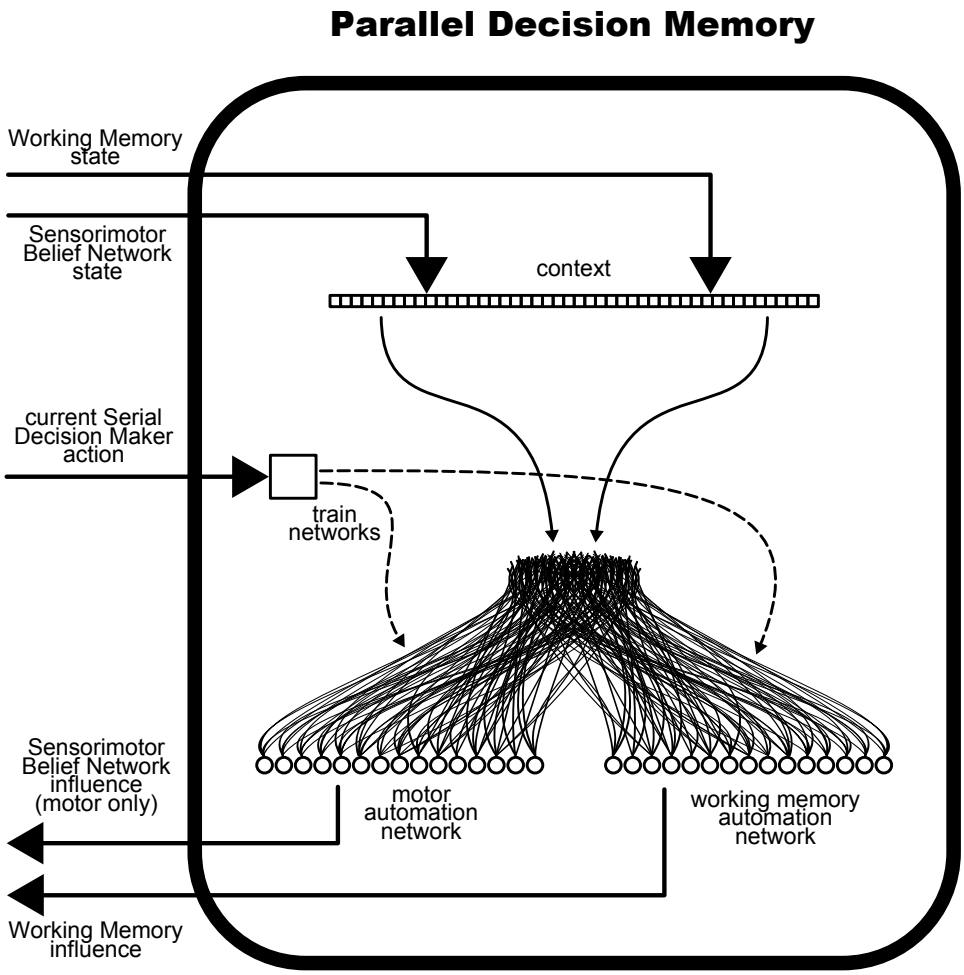


Figure 18: The Parallel Decision Memory component. This operates as a massive supervised single-layer neural network. Contextual inputs are provided to the network to compute its outputs. The weights are trained with the error between actual and desired outputs. However, only the weights related to the currently-chosen action (by the Serial Decision Maker) are trained. The resulting output array is sent out to influence motor control and working memory updating.

sensory cortex "grew out of" the early hippocampus (which started as an outgrowth of the olfactory system) [12], sensory cortex evolved an extension with motor outputs (the motor cortex), basal ganglia evolved connections to the motor cortical regions for action selection, and later the motor cortex evolved an extension (prefrontal cortex) which lost the motor outputs but maintained the basal ganglia selection mechanism. This intriguing structure provides inspiration for the Working Memory component of the Sapience architecture.

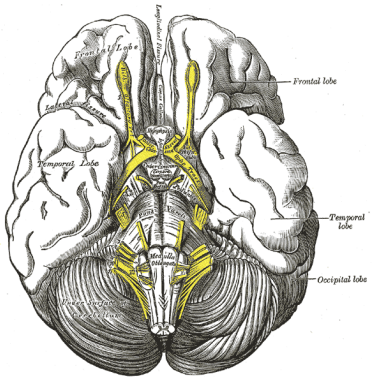


Figure 19: View of the underside of the brain showing the prefrontal cortex at the top. From Gray's Anatomy (public domain).

Anatomy In front of the premotor cortex, the prefrontal area makes up most of the frontal lobe in humans [1]. It maintains reciprocal connections with many other cortical regions and with the basal ganglia. Similar to the columnar organization of the sensorimotor cortex, the prefrontal cortex appears to be organized into a discrete set of "stripes," of which there are approximately 20,000 in humans [20].

Function Interestingly, unlike the more posterior areas of frontal cortex (primary motor, premotor, and supplementary motor), the prefrontal regions generally do not produce motor responses following electrical stimulation.

Many behavioral correlates of the prefrontal cortex can be inferred from damage to this area. Such damage may lead to deficits involving attention allocation, short-term/working memory, decision making, initiation of behavior, distractibility, impulsiveness, failure to complete tasks, or perseveration (failure to recognize when a task is complete) (see [1] pp. 250-251).

Models One recent model involving the prefrontal cortex is the PBWM (prefrontal cortex/basal ganglia/working memory) model by Hazy, Frank, & O'Reilly [20]. In this model the basal ganglia model is able to selectively update working memory "stripes" in prefrontal cortex with new information from the posterior/sensorimotor cortex. It must learn which stripes to update based on reinforcements – if a particular working memory representation tends to provide more rewards (or less punishment), it will be chosen more often. This scheme follows the basic reinforcement learning paradigm, applied to working memory actions instead of motor actions.

8.2 Functional Abstraction

In designing our Working Memory component (Figure 20) the core feature is a discrete number of memory cells. This follows the stripe-like organization of prefrontal cortex.We follow the inspiration of the PBWM model [20], namely that working memory units are treated as possible "actions" to be chosen by a basal ganglia-like decision maker. Working Memory cells can be selectively written to (with contents from the Sensorimotor Belief Network) or read from (allowing memory cells' contents to influence sensorimotor beliefs and also the decision making components). Fortunately, we have already defined our Serial Decision Maker to choose actions in a fairly general way. Now we can simply extend its set of actions to include read and write actions within a Working Memory component. In any given context, the Serial Decision Maker might choose to enable some motor action, write something to working memory (Sensorimotor Belief Network beliefs), or read something back out of working memory (influencing beliefs or actions).

The Working Memory component acts as a temporary memory storage area. Its implementation is relatively simple because its discrete operations are very similar to those of digital computers. Its intimate connection with the decision making components makes it essentially a type of "neural RAM" with distinct read/write operations. We distinguish between two types of memory cells: discrete cells and binary cells. When the "read" action is chosen on a discrete cell, it pulls in new sensorimotor contents (current posterior distributions). When the "write" action is chosen, its output gate is opened, and its contents influence sensorimotor beliefs (as an extra likelihood message, or "virtual evidence") and the decision making components. For binary cells the same read/write operations exist, but the cells are not connected to the Sensorimotor Belief Network – they simply act as general purpose

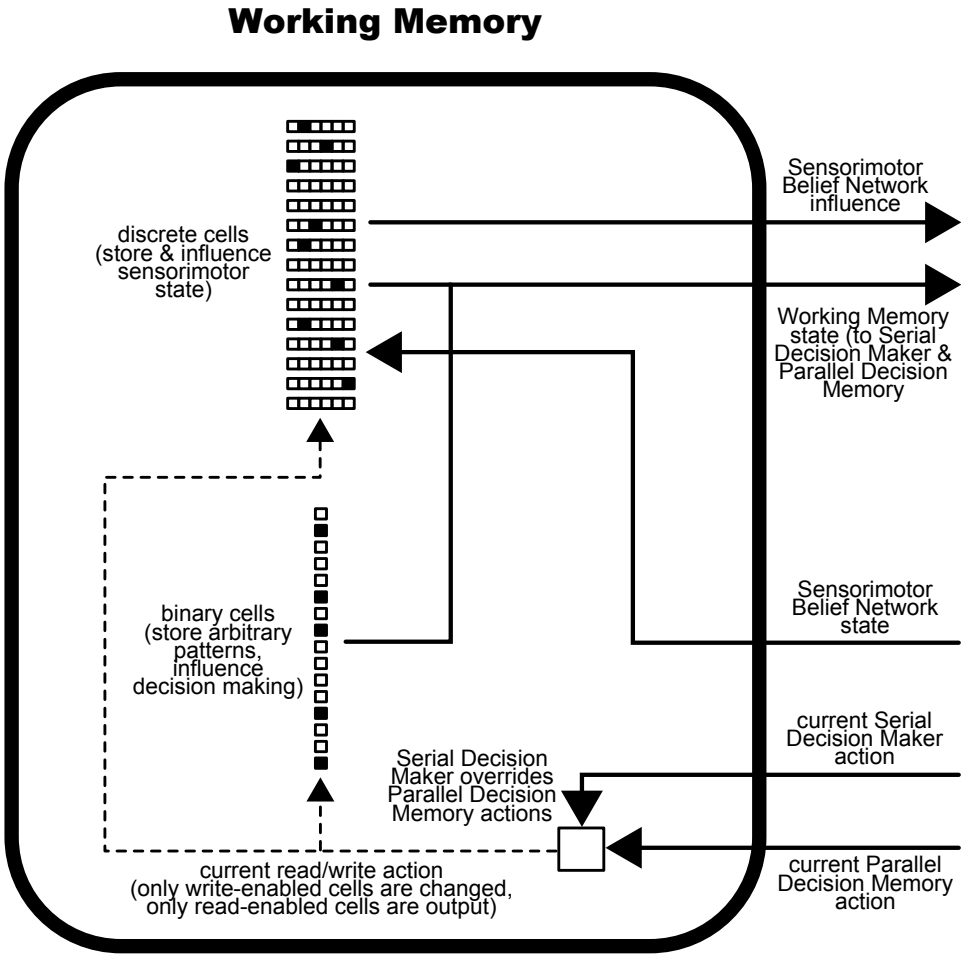


Figure 20: The Working Memory component is comprised of an array of discrete gated memory cells which write to/read from the Sensorimotor Belief Network. It also includes a set of binary cells, not influenced by sensorimotor beliefs, which simply influence the decision making components. Read/write operations are driven by the Serial Decision Maker and the Parallel Decision Memory.

storage for influencing future decisions.

Interestingly, the entire contents of this Working Memory array are included as input to the Serial Decision Maker; thus, a recursive relationship exists where Working Memory contents affect the next read/write action, which modifies Working Memory contents, which then affects the next read/write action, etc. This provides a foundation for learning arbitrary action sequences/motor programs. It might even be possible to demonstrate a correspondence between this Serial Decision Maker/Working Memory mechanism and a universal Turing machine, where the Serial Decision Maker acts as the "head" (choosing memory cells and performing read/write actions) and the Working Memory cells act as the "tape." This would be a powerful theoretical foundation for our Working Memory component and might provide an intriguing framework for studying basal ganglia/prefrontal cortex interactions.

9 Completed Work

This chapter covers the work completed so far, including implementations of the main architecture components, a probe tool for real-time introspection (plots and 3D visualizations) of the system's internal variables, and a simulation environment for running embodied experiments. Altogether, the current implementation, experimental programs, and auxiliary tools represent over 1 megabyte of source code (uncompressed C++ and Python text files).

9.1 Component Implementations

All five components described earlier have been implemented in C++. The Sensorimotor Belief Network has undergone extensive testing in isolation. The Sequential Memory has been tested successfully on a keyboard-based melodic sequence storage and recall program. The PVLV model within the Serial Decision Maker has been tested successfully on a basic classical conditioning program. The Parallel Decision Memory and Working Memory implementations have not yet been tested on such specific experiments.

The implementation includes various free parameters for balancing speed vs. accuracy. It also includes support for thread-based parallelization of Bayesian network updating, which should improve runtime performance on

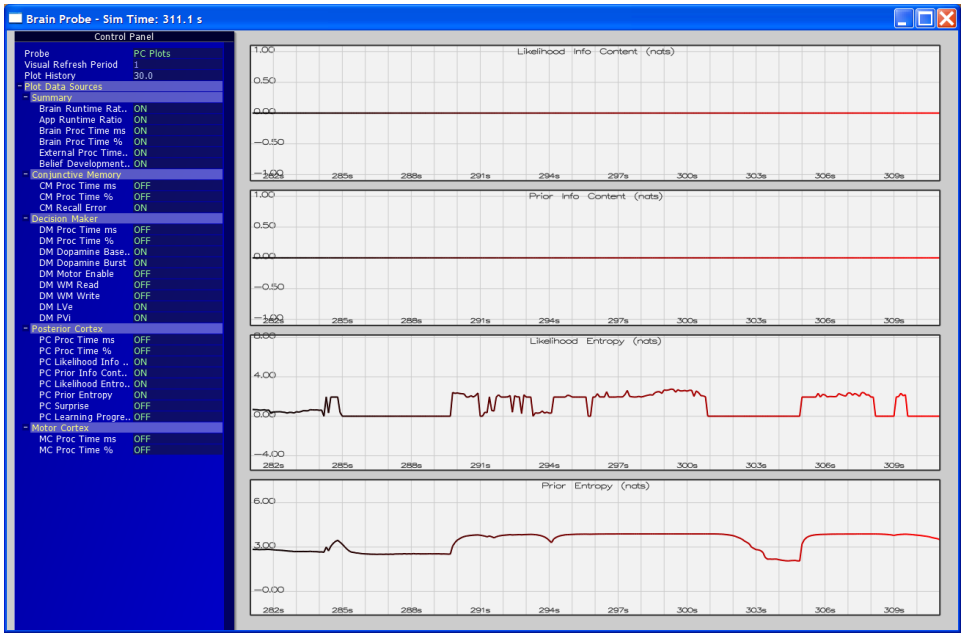


Figure 21: Probe window showing real-time plots of key internal variables.

multi-processor machines. Also, many test programs have been written to test small parts of the implementation (e.g., a real-time density estimation visualization to test the kernel mixture model, as shown in Figures 8, 9, 10, and 11, and a topographic map formation test, shown in Figure 7).

9.2 Architecture Probe Tool

Besides the architecture implementation, we have implemented a probe tool (see Figures 21 and 22). This tool accesses the Sapience architecture implementation and displays its various internal variables in a separate window, including many real-time plots. The information provided by this tool will be crucial in debugging any problems during experimentation.

9.3 Simulation Platform

In order to test our architecture implementation on various learning experiments, we have implemented a 21-degree-of-freedom simulated arm. The

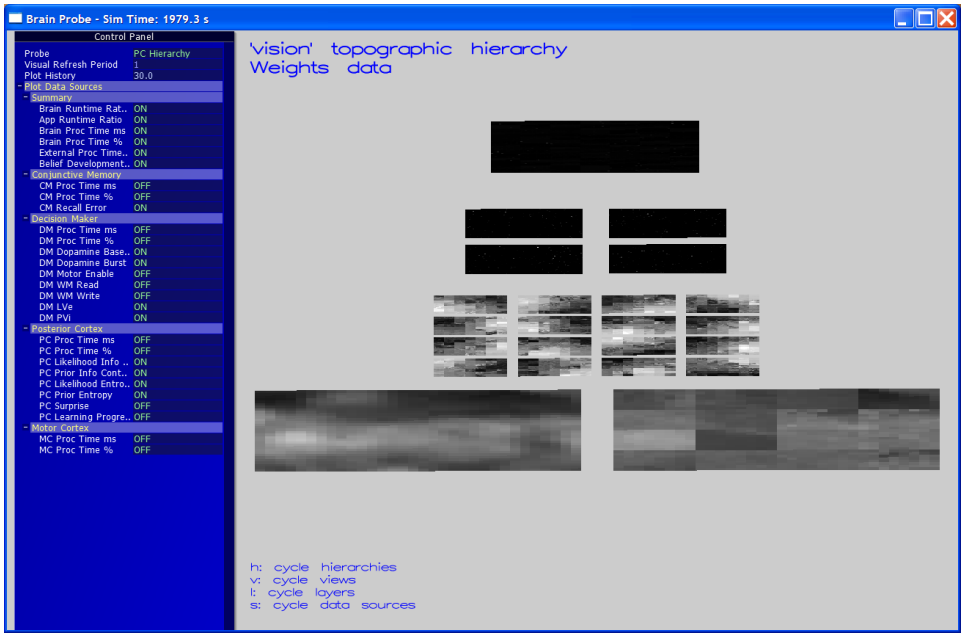


Figure 22: Probe window showing a visual representation of the Sensorimotor Belief Hierarchy. In this case the input data is coming from a set of natural images, and the various levels in the hierarchy display the learned kernel center/weight vectors (visual image filters).

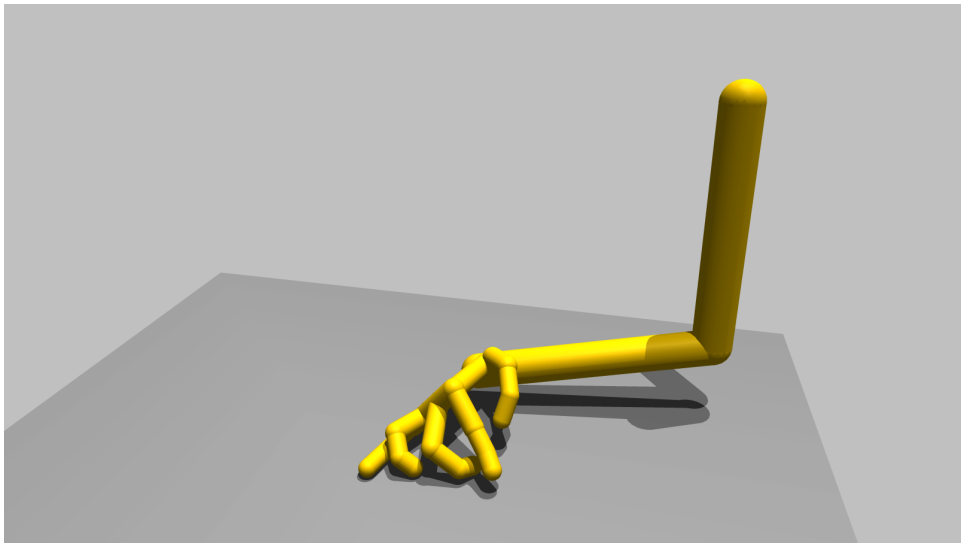


Figure 23: Screenshot of simulated arm.

simulation environment uses Ogre [65] to render 3D graphics and Open Dynamics Engine [57] with OPAL [64] to simulate collision detection and physics. The arm itself is fixed in space at the shoulder joint and positioned above a table-like surface which can be used for manual exploration of various simulated objects (Figure 23). Each joint degree of freedom (some joints have multiple DOFs) in the arm and hand uses pre-defined rotational limits roughly corresponding to range of motion present in the human arm and hand joints. Sensory inputs are provided from three main sources: proprioception (joint angles and motor force output), touch (simulated skin contact sensors), and vision (a single grayscale video stream). Motor outputs drive servo controllers present in each joint degree of freedom. Figure 24 shows the simulation again with visualization of the touch sensors and vision stream.

Proprioception Each joint degree of freedom provides two types of proprioceptive information: the current angle and the current force output, both normalized to [0, 1].

Touch Sensors The surface of the skin is covered with an array of touch sensors. These simply provide binary touch feedback as follows: when any object contacts the skin, the nearest touch sensor is made active. Touch

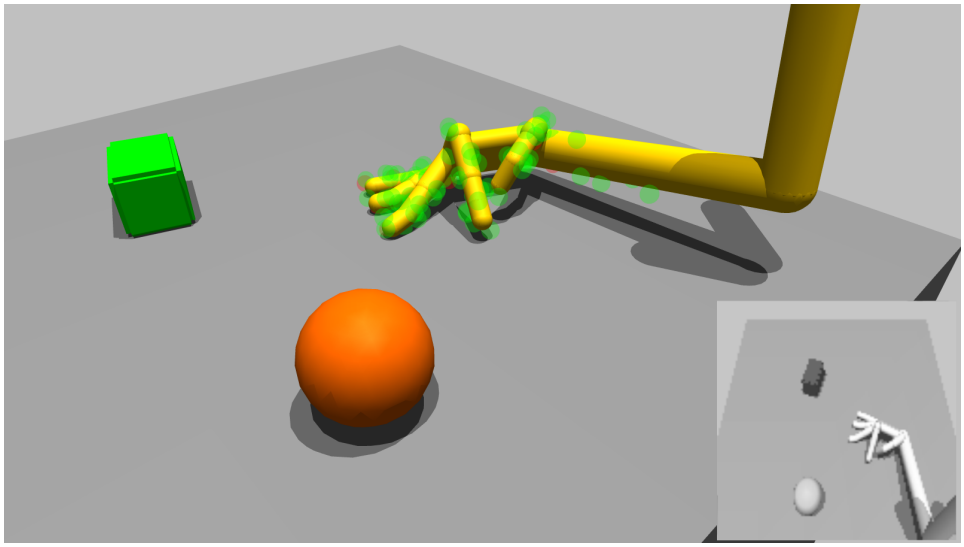


Figure 24: Arm simulation with sensory system visualization enabled. This shows the simulated eye (monochrome, monocular vision) as well as the touch sensor positions across the skin surface.

sensors without any nearby contacts remain inactive.

Vision To simulate vision we render the entire 3D scene from a single (monocular) viewpoint onto a 2D texture in memory. Color information is removed – we focus only on monochrome vision. The 2D texture data can then be sent directly to the Sapience architecture as sensory input. This texture can also be displayed on the screen for debugging, as shown in Figure 24.

Servo Motors Each joint degree of freedom utilizes a servo motor to control limb motion. The built-in feedback mechanism in these controllers mimics the spinal stretch reflex ([58], section 2.5), maintaining a desired angle (within certain force limits) despite external disturbances. This allows the higher-level controller (the Sapience architecture implementation) to focus on setting simpler parameters like desired angles without focusing on precise force values. The actual control mechanism is a "force-limited velocity constraint" (FLVC), which is similar to but more robust than a PID controller. Smith implemented FLVCs (joint motors) in the Open Dynamics

Engine software library. Other researchers [51] have also used FLVCs in simulations instead of PD/PID controllers, citing that they are simpler (fewer free parameters) and more numerically stable.

A FLVC takes a desired velocity (e.g., a rate of rotation) and a maximum force. The physics simulator's constraint solver then tries to reach the target velocity in one time step using up to the maximum force. (Instead, we wish to set the controller's desired joint *angle*, so we simply set the FLVC's desired velocity in proportion to the error between the current joint angle and the desired angle.) Thus, each joint DOF has a sort of equilibrium point (the desired angle) which can be set. Additionally, we wish to control the amount of stiffness present in a joint. For this we simply interpret the FLVC's maximum force value as the joint stiffness which can be set along with the equilibrium point.

The resulting motor system allows independent control of joint angle and stiffness, a very general control scheme for jointed systems (see [32] for biological evidence of this type of dual control). Critically, two related variables (actual joint angle and actual force output) are provided as proprioceptive sensory inputs, as described above. This enables the Sapience architecture's motor output signals to be expressed in terms of the sensory data space (proprioception) it has learned to represent.

9.4 Initial Runtime Performance Test

With all five Sapience architecture components enabled and connected to the arm simulation, we performed an initial runtime performance test of our implementation to ensure that, in practice, it can be updated at a reasonable rate. This test was performed on a computer with a dual 2.8 GHz Pentium D CPU, 3 GB RAM, and a GeForce 6800 video card. The system was given 42 proprioceptive inputs (joint angle and force output for each joint DOF), a 32x32 pixel visual input, no touch sensors enabled, and 42 motor outputs (desired joint angle and stiffness for each joint DOF).

The Sensorimotor Belief Network contained 60 nodes (17 sensory, 42 motor, 1 multimodal), with a total of 931 belief values (all posteriors concatenated). The Sequential Memory had 400 inputs and 400 outputs. The Serial Decision Maker had 1862 static inputs (sensorimotor + working memory), 553 temporal inputs (from the Sequential Memory), and 498 outputs (motor + working memory). The Parallel Decision Memory had 1862 inputs (senso-

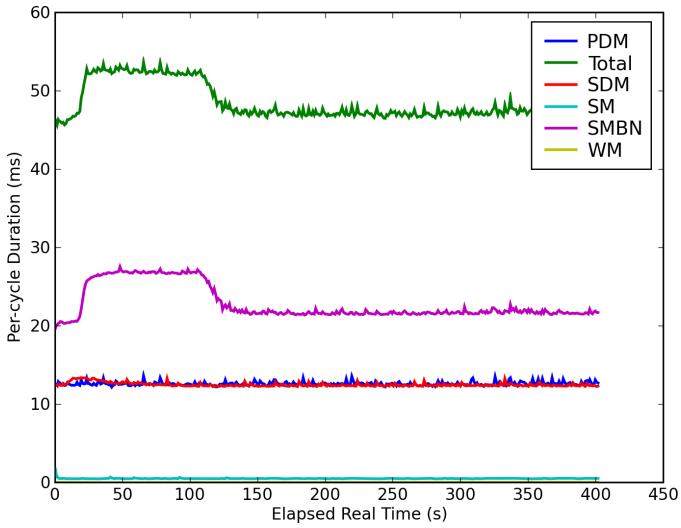


Figure 25: Initial runtime profiling results. The total update took less than 50 ms on average. Abbreviations refer to the five main Sapience components. The initial increase and later decrease in runtime for the Sensorimotor Belief Network (also reflected in the total) is due to an initial increase in topographic lattice connections (for the kernel mixture models) and subsequent removal as the neighborhood size shrinks. See text for more details.

rimotor + working memory), 378 motor outputs, and 60 working memory outputs. The Working Memory used 60 discrete cells, zero binary cells, and had 931 reciprocal sensorimotor connections. Multithreading was enabled for Sensorimotor Belief Network processing. We updated the Sapience implementation at a rate of 10 Hz, the physics simulation at 100 Hz, and the 3D graphics at roughly 50 Hz.

The resulting timing plot is shown in Figure 25. In summary, each Sapience update took about 50 ms, with the Sensorimotor Belief Network consuming about 22 ms per update, the Serial Decision Maker and Parallel Decision Memory each taking about 13 ms per update, and the Sequential Memory and Working Memory each taking about 1 ms per update. This initial estimate shows that our Sapience implementation, with modest input/output dimensionality, can run in real-time on a standard computer, leaving enough time to maintain the desired physics and graphics update rates. We expect the runtime performance to improve further with future software optimizations and potential support for GPGPU (general purpose graphics processor)

acceleration of the kernel mixture model algorithm (the most CPU-intensive part).

10 Proposed Work

We propose to perform a set of experiments in order to evaluate the Sapience architecture's effectiveness towards its learning objectives. The two main measurable objectives, defined earlier, involve external reward intake and world model improvements.

These experiments attempt to demonstrate fully autonomous motor development, driven by simple reinforcements, capable of handling high-dimensional sensory data and motor outputs. Our general strategy is to start with simpler tests, with most components disabled, and incrementally add more components. This will hopefully show the overall benefits of each component in terms of the learning objectives.

10.1 Experiment Set 1: Passive Data Compression

Initially we plan to start with only the Sensorimotor Belief Network and a naive Sequential Memory (i.e. the output priors simply equal the previous-step likelihood inputs). This minimal system will be given vision inputs only; there will be no motor-related inputs or outputs. The simulated arm will be driven externally by a simple reflex control system, providing continually changing visual inputs. No internal or external rewards will be involved.

We will plot the model improvement rate (reconstruction error and/or information gain methods) over time based on the visual inputs and no active control over the input distribution. This will provide a baseline measurement for comparison with later experiments.

We will perform the experiment three times: once with only a single belief node (no hierarchical division of input space), again with the hierarchical Bayesian representation to show the benefit of such a hierarchical representation, and once more with the full (non-naive) Sequential Memory enabled to show the benefit of its sequential prediction capabilities.

10.2 Experiment Set 2: Active Data Compression

This set of experiments builds on the previous set. Starting the arm simulation with the same motor reflex system as before, this time the control signals will gradually transition to be controlled by the Sequential Decision Maker. In addition to visual sensory inputs, proprioceptive inputs will be included. Internal curiosity rewards will be provided based on model improvements. Similar plots will be generated as before regarding the model improvement rate; however, this time the curiosity rewards should encourage targeted exploration, resulting in a noticeably better model improvement rate.

Optionally, it may be possible to generate a set of images of the simulated arm representing the learned motor gestures; this would be possible by artificially biasing the belief values in the multimodal belief node, letting this bias influence motor outputs, and visualizing the resulting arm configuration.

10.3 Experiment Set 3: External Reward Acquisition

This third set of experiments adds external rewards. A simple reaching task can be defined, where a target position is presented to the system with a visual indicator. When the hand intersects the target position, a reward is given, and the target is moved to a different position. This scheme can be repeatedly endlessly. A plot will be generated of the reward acquisition rate over time, with and without the inclusion of internal curiosity rewards. This should show the benefit of curiosity rewards for targeted exploration, leading to improved performance on externally-rewarded tasks (as demonstrated on a much simpler domain in [61]).

Optionally, it would be possible to generate a series of plots with the effect of curiosity rewards scaled relative to external rewards (e.g., curiosity rewards scaled to be 0.5X, 1X, and 2X as powerful as external rewards). There may be an optimal relationship between internal and external rewards which, for example, results in behavior that includes some amount of exploration but not so much that it interferes with externally-rewarded task learning.

10.4 Optional Experiments

These extra experiments will be performed if there is time remaining before the final defense.

10.4.1 Parallel Decision Memory Benefits

Enabling the Parallel Decision Memory component should demonstrate a transfer of motor control away from the Serial Decision Maker for well-learned tasks, similar to the computational cerebellum/basal ganglia model in [45]. This would be demonstrated within the same experimental environment as described above, where the task is to reach towards rewarding positions in space. However, by plotting the influence of both the Serial Decision Maker and the Parallel Decision Memory on the motor areas of the Sensorimotor Belief Network, we hope to show a reduction in influence by the Serial Decision Maker along with a simultaneous increase in Parallel Decision Memory influence. Further, we could show the benefits in terms of reward acquisition by adding this extra component, simply by plotting the external reward acquisition rate over time on the reaching task (vs. with the Parallel Decision Memory disabled). Finally, by switching the task definition partway through the experiment (e.g., by changing the visual indicator representing the goal position), it should be possible to demonstrate an abrupt increase in Serial Decision Maker control as the Parallel Decision Memory suddenly begins making more errors.

10.4.2 Working Memory Benefits

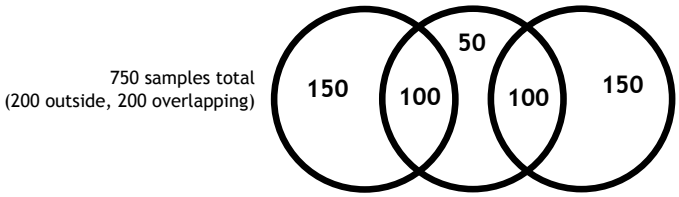
Enabling the Working Memory component should demonstrate the ability to learn tasks which require active internal maintenance of task-relevant information. This could be demonstrated with a simple cued response task: briefly display one of two visual indicators, wait a short period, and request one of two responses from the system. The earlier visual indicator would represent which of the final responses is appropriate. If the correct cued response is chosen, an external reward is given; otherwise, an external punishment is given. This type of task should be impossible to solve without the Working Memory enabled.

10.4.3 Improved Kernel Mixture Model Learning Rule

This experiment has an entirely different goal from the others. The kMER algorithm described earlier can be viewed as an approximation to the infomax learning principle. kMER's basis as an "equiprobabilistic" learning rule is a heuristic which might be improved, making it even closer to the info-

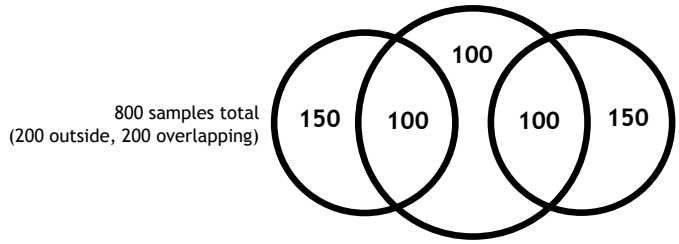
max ideal. Figure 26 represents one such improvement which may provide benefits over kMER. Briefly, instead of producing a set of receptive fields which contain the same number of data samples (regardless of overlap with other receptive fields), it may be possible to devise a modification of kMER where each receptive field contains the same number of "normalized" samples (e.g., a sample lying within two receptive fields only counts as half). The motivation behind this change is that by assuming the kernels represent data-generating processes, a data sample should be attributed to only one kernel, or partway to multiple kernels, but not fully to multiple kernels. Initial experiments, already performed, show that this type of learning rule is possible. It remains to be seen whether this new scheme provides actual benefits, as measured on standard density estimation and pattern classification tasks. (Note: I have been working on this problem via email with Marc Van Hulle who created the kMER algorithm.)

Equal code membership rule (kMER)



	kernel 1	kernel 2	kernel 3
Total # samples within radius →	250	250	250
# overlapping samples →	100	200	100
# samples after sharing →	200	150	200
Mean activation probability →	0.33	0.33	0.33
Mean fuzzy code membership →	0.36	0.27	0.36
Overlap fraction →	0.4	0.8	0.4

Equal fuzzy code membership rule



	kernel 1	kernel 2	kernel 3
Total # samples within radius →	250	300	250
# overlapping samples →	100	200	100
# samples after sharing →	200	200	200
Mean activation probability →	0.31	0.38	0.31
Mean fuzzy code membership →	0.33	0.33	0.33
Overlap fraction →	0.4	0.66	0.4

Figure 26: A visual representation of the major difference between the kMER learning rule and a new equal fuzzy code membership rule. In this hypothetical example we show the difference between two different radius update rules, assuming fixed centers. The numbers in the diagrams indicate the number of samples within each region. The size of each receptive field indicates the number of samples it contains. kMER produces an equal number of samples within each receptive field (ignoring overlap), while the equal fuzzy code membership rule shares samples more fairly by dividing them among overlapping receptive fields. Note that the major difference between these two rules concerns overlap; when there is no overlap, the results are the same.

References

[1] Adel K. Afifi and Ronald A. Bergman. *Functional Neuroanatomy: Text and Atlas, 2nd Edition*. McGraw Hill, New York, NY, 2005.

[2] James S. Albus. A theory of cerebellar function. *Mathematical Biosciences*, 10:25–61, 1971.

[3] James S. Albus and Alexander M. Meystel. *Engineering of Mind: An Introduction to the Science of Intelligent Systems*. John Wiley & Sons, New York, NY, 2001.

[4] David G. Amaral, Norio Ishizuka, and Brenda Claiborne. *Progress in Brain Research, Volume 83: Understanding the Brain through the Hippocampus*, chapter Neurons, Numbers and the hippocampal network, pages 1–11. Elsevier Science, Amsterdam, 1990.

[5] John R. Anderson. *The Architecture of Cognition*. Harvard University Press, Cambridge, MA, 1983.

[6] Hisham E. Atallah, Michael J. Frank, and Randall C. O'Reilly. Hippocampus, cortex, and basal ganglia: Insights from computational models of complementary learning systems. *Neurobiology of Learning and Memory*, 82:253–267, 2004.

[7] Jean Paul Banquet, Philippe Gaussier, Arnaud Revel, and Sorin Moga. Sequence learning and timing in hippocampus, prefrontal cortex, and accumbens. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1053–1058, Washington, DC, 2001. IEEE.

[8] William Bialek, Rob R. de Ruyter van Steveninck, and Naftali Tishby. Efficient representation as a design principle for neural coding and computation, 2007. arXiv:0712.4381.

[9] James R. Bloedel and Vlastislav Bracha. Current concepts of climbing fiber function. *The Anatomical Record*, 253(4):118–126, 1998.

[10] I.C. Bruce and W.G. Tatton. Synchronous development of motor cortical output to different muscles in the kitten. *Experimental Brain Research*, 40:349–353, 1980.

[11] Gordon M. Burghardt. *The Genesis of Animal Play: Testing the Limits*. MIT Press, Cambridge, MA, 2005.

[12] Ann B. Butler and William Hodos. *Comparative Vertebrate Neuroanatomy, 2nd Edition*. John Wiley & Sons, Inc., Hoboken, NJ, 2005.

[13] Peter Dayan and Larry F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, 2001.

[14] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39:1–38, 1977.

[15] Kenji Doya. Complementary roles of basal ganglia and cerebellum in learning and motor control. *Current Opinion in Neurobiology*, 10(6):732–739, 2000.

[16] Karl Friston. Hierarchical models in the brain. *PLoS Computational Biology*, 4(11):e1000211, 2008.

[17] Apostolos P. Georgopoulos. Arm Movements in Monkeys: Behavior and Neurophysiology. *Journal of Comparative Physiology: A. Sensory, Neural, and Behavioral Physiology*, 179(5):603–612, 1996.

[18] Richard Granger. Engines of the brain: The computational instruction set of human cognition. *AI Magazine*, 27:15–32, 2006.

[19] Simon Haykin. *Neural Networks and Learning Machines*. Prentice Hall, 2008.

[20] Thomas E. Hazy, Michael J. Frank, and Randall C. O'Reilly. Towards an executive without a homonculus: Computational models of the prefrontal cortex/basal ganglia system. *Philosophical Transactions of the Royal Society B*, 2007.

[21] O. Hikosaka, K. Miyashita, S. Miyachi, K. Sakai, and X. Lu. Differential roles of the frontal cortex, basal ganglia, and cerebellum in visuomotor sequence learning. *Neurobiology of Learning and Memory*, 70:137–149, 1998.

[22] James C. Houk, James L. Adams, and Andrew G. Barto. *Models of Information Processing in the Basal Ganglia*, chapter A Model of How the Basal Ganglia Generate and Use Neural Signals That Predict Reinforcement, pages 249–270. MIT Press, Cambridge, MA, 1995.

[23] David H. Hubel and Torsten N. Wiesel. Receptive fields of cells in striate cortex of very young, visually inexperienced kittens. *Journal of Neurophysiology*, 26:994–1002, 1963.

[24] Marc M. Van Hulle. *Faithful Representations and Topographic Maps: From Distortion- to Information-Based Self-Organization*. Wiley, New York, NY, 2000.

[25] Laurent Itti and Pierre Baldi. Bayesian surprise attracts human attention. In Bernhard Schölkopf, John Platt, and Thomas Hofmann, editors, *Advances in Neural Information Processing Systems 19*, pages 547–554. MIT Press, Cambridge, MA, 2006.

[26] Christopher Johansson and Anders Lansner. Towards cortex sized artificial neural systems. *Neural Networks*, 20:48–61, 2007.

[27] Eric R. Kandel, H. Schwartz James, and Thomas M. Jessel. *Principles of Neural Science, Fourth Edition*. McGraw-Hill, New York, NY, 2000.

[28] Tuevo Kohonen. *Self-Organizing Maps*. Springer, New York, 1997.

[29] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[30] Ray Kurzweil. *The Age of Spiritual Machines: When Computers Exceed Human Intelligence*. Penguin USA, New York, NY, 1999.

[31] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[32] Mark L. Latash. Independent control of joint stiffness in the framework of the equilibrium-point hypothesis. *Biological Cybernetics*, 67:377–384, 1992.

[33] Tai Sing Lee and David Mumford. Hierarchical bayesian inference in the visual cortex. *Journal of the Optical Society of America A*, 20(7):1434–1448, 2003.

[34] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[35] Dennis V. Lindley. On a measure of the information provided by an experiment. *Annals of Mathematical Statistics*, 27(4):986–1005, 1956.

[36] Ralph Linsker. Self-organization in a perceptual network. *Computer*, 21(3):105–117, 1988.

[37] Moshe Looks. *Competent Program Evolution*. PhD thesis, Washington University, Saint Louis, MO, 2006.

[38] Henry Markram. The blue brain project. *Nature Reviews Neuroscience*, 7(2):153–160, 2006.

[39] David Marr. A theory of cerebellar cortex. *Journal of Physiology*, 202:437–470, 1969.

[40] Michael D. Mauk and Dean V. Buonomano. The neural basis of temporal processing. *Annual Review of Neuroscience*, 27:307–340, 2004.

[41] Vernon B. Mountcastle. The columnar organization of the neocortex. *Brain*, 120:701–722, 1997.

[42] Randall C. O'Reilly, Michael J. Frank, Thomas E. Hazy, and Brandon Watz. Pvlv: The primary value and learned value pavlovian learning algorithm. *Behavioral Neuroscience*, 121(1):31–49, 2007.

[43] Randall C. O'Reilly and Yuko Munakata. *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*. MIT Press, Cambridge, MA, 2000.

[44] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[45] Charles Peck, Tyler Streeter, and James Kozloski. An integrated cerebro-cerebellar model demonstrating associative learning and motor control. In *Proceedings of the 10th Tamagawa-Riken Dynamic Brain Forum*, Hakuba, Nagano, Japan, 2007. Tamagawa University-Riken Brain Science Institute.

[46] Tony J. Prescott, Fernando M. Montes Gonzalez, Kevin Gurney, Mark D. Humphries, and Peter Redgrave. A robot model of the basal ganglia: Behavior and intrinsic processing. *Neural Networks*, 19:31–61, 2006.

[47] The Open Cognition Project. Opencog prime. http://opencog.org/wiki/OpenCogPrime, May 2008. Open source AGI design.

[48] Dale Purves, George J. Augustine, David Fitzpatrick, William C. Hall, Anthony-Samuel LaMantia, James O. McNamara, and S. Mark Williams. *Neuroscience*. Sinauer, Sunderland, MA, 2004.

[49] Rajesh P. N. Rao and Dana H. Ballard. Predictive Coding in the Visual Cortex: A Functional Interpretation of Some Extra-Classical Receptive-Field Effects. *Nature Neuroscience*, 2(1):79–87, 1999.

[50] Peter Redgrave, Tony J. Prescott, and Kevin Gurney. The basal ganglia: A vertebrate solution to the selection problem? *Neuroscience*, 89:1009–1023, 1999.

[51] Torsten Reil and Colm Massey. *Morpho-functional Machines: The New Species: Designing Embodied Intelligence*, chapter Facilitating Controller Evolution in Morpho-functional-Machines - A Bipedal Case Study, pages 81–98. Springer, 2003.

[52] Juergen Schmidhuber. Curious Model-Building Control Systems. In *Proceedings of the International Joint Conference on Neural Networks, Singapore, Volume 2*, pages 1458–1463. IEEE, 1991.

[53] Juergen Schmidhuber. Simple algorithmic theory of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. *Journal of SICE*, 48(1):21–32, 2009.

[54] Juergen Schmidhuber. Ultimate cognition a la goedel. *Cognitive Computation*, 1:177–193, 2009.

[55] Juergen Schmidhuber, Jan Storck, and Josef Hochreiter. Reinforcement driven information acquisition in non-deterministic environments. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 159–164, Paris, France, 1995. ICANN.

[56] Wolfram Schultz. Multiple Reward Signals in the Brain. *Nature Reviews Neuroscience*, 1:199–207, 2000.

[57] Russell Smith. Open dynamics engine. http://www.ode.org, April 2001. Open source rigid body dynamics library.

[58] Russell L. Smith. *Intelligent Motion Control with an Artificial Cerebellum*. PhD thesis, University of Auckland, Auckland, New Zealand, 1998.

[59] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[60] Tyler Streeter. Design and implementation of general purpose reinforcement learning agents. Master's thesis, Iowa State University, Ames, IA, 2005.

[61] Tyler Streeter. Curiosity-driven exploration with planning trajectories. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 1897–1898, Boston, MA, 2006. AAAI.

[62] Tyler Streeter. A hierarchical empirical bayesian model of cerebral cortex. http://www.agi-09.org, March 2009. Poster presentation.

[63] Tyler Streeter, James Oliver, and Adrian Sannier. Verve: A general purpose open source reinforcement learning toolkit. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 359–369, Philadelphia, PA, 2006. ASME.

[64] Tyler Streeter, Andres Reinot, Alan Fischer, and Oleksandr Lozitskiy. Open physics abstraction layer. http://opal.sourceforge.net, September 2004. Open source physics API.

[65] Steve Streeting. Ogre. http://www.ogre3d.org, February 2000. Open source 3D graphics engine.

[66] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.