Unifying Perception and Curiosity

by

Jonathan M. Shaw

Submitted in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Supervised by

Professor Dana H. Ballard

Department of Computer Science
The College
Arts and Sciences

University of Rochester
Rochester, New York

2006
To

my wife Joanna, for her support and love immeasurable
my daughter Ariana, for the joyous distraction she has been
my parents, for bending my will toward worthy things, and inspiring me to do them
my Lord and Savior Jesus Christ; may my work always be a sweet offering to Him
Curriculum Vitae

Jonathan Shaw was born in Portland, Oregon on June 22, 1978. He participated in a transfer program from George Fox University to Seattle Pacific University and received a Bachelor of Science in Applied Science from George Fox University in 1998 and a Bachelor of Science in Computer Science from Seattle Pacific University in 1999. He began graduate studies at the Oregon Graduate Institute in 1999 and received a Master of Science degree in Computer Science in 2001. He then moved to the University of Rochester and began research in predictive coding under the direction of Professor Dana H. Ballard. He received a Master of Science degree from the University of Rochester in 2003.
I would like to thank my father for showing me that the world is a fascinating place. When I was little, I watched him build a robot in our garage that had two legs and only pegs for feet. I think his work on making this robot balance and walk had a more direct influence on my interests and the subject matter of this dissertation than anything else in my life.

While my father provided wonder, my mother provided humor and courage. She taught me to always be cheerful, and to never be afraid of what anyone might think. One cannot get discouraged while being cheerful, and this has served as a source of strength over the years.

My sister has always been there for support and encouragement. When I first moved to New York from Oregon, when I was lonely and hadn’t made any friends, it was talking to Rachel on the phone that helped me through the hard times more than anything else.

Midway through my time in New York, I got married. Joanna has been an incredible source of joy and companionship, and has helped me keep work in its proper perspective. She has also replaced the disastrously messy apartment I kept with a beautiful oasis to return to every evening.

My wife has also given me a daughter. Ariana brings great joy to life, and reminds me that the world is a fascinating place. She is amused by the smallest things, as we all should be.

I would like to thank my advisor Dana Ballard for working with me over the past several years, even though it has not always been easy. His guidance has shaped my thinking about the brain a great deal and has led me to the particular area of research I have chosen.

I would like to thank Professors Chris Brown and Mitsunori Ogihara for helping me out in a difficult time when my advisor was away.

I would like to thank Professor Alex Pouget for giving me a perspective on the brain which is different from that of my advisor’s. Some day it will be known whether neurons are deterministic or follow Poisson processes, but it has been very valuable in my research to know that both are acceptable viewpoints.
I would like to thank Xipeng Shen and Manu Chhabra for many excellent conversations.

Finally, I would like to thank the Lord for giving me the talents, inspiration, motivation, determination, and everything else required to do this. But for his grace, none of this would ever have happened, or indeed have even been started.

This material is based upon work supported by a GAANN grant #P200A000306 from The Department of Education, grant # 5 RO1RR009283-11 from the National Institute of Health, and a National Science Foundation grant number E1A-0080124. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of above named organizations.
Abstract

There has been much research in recent decades aimed at discovering what the underlying principles are, if any, that drive the brain. As the cortex appears to be basically uniform, it seems that if there is an underlying principle, it is ubiquitous. However, the principles which have been proposed to explain the brain have largely been specialized principles, which each explain a particular aspect of the brain.

Principles such as efficient coding, predictive coding, and temporal invariance have been proposed to explain sensory coding, and have succeeded to some measure in reproducing the receptive field properties of neurons in the visual cortex. Bayesian surprise has been offered as an explanation of attention, and has enjoyed some success in modeling human saccades, while reinforcement learning and intelligent adaptive curiosity have been aimed at explaining how actions are chosen.

In this dissertation we propose a novel principle which we call predictive action. It is an information theoretic principle which unifies all of the above proposals. We show its relationship to each of the above proposals, and give several algorithms which approximate predictive action for specific environments. We hope that this principle will allow not only for a greater understanding of the brain, but also serve as a principled basis for the design of future algorithms to solve a broad range of problems in artificial intelligence.
# Table of Contents

**Curriculum Vitae** iii  
**Acknowledgments** iv  
**Abstract** vi  
**List of Tables** xi  
**List of Figures** xii

## 1 Introduction

1.1 Study of the Brain .................................................. 1  
1.2 A Global Optimization Principle ................................. 2  
  1.2.1 An Optimization Principle for Sensory Perception .... 3  
  1.2.2 A Global Optimization Principle for the Brain .......... 5  
1.3 The Thesis ............................................................. 5  
1.4 Dissertation Contributions ...................................... 5  
1.5 Organization .......................................................... 6

## 2 Background

2.1 Neuron Models ....................................................... 7  
  2.1.1 Integrate and Fire Model ............................... 7  
  2.1.2 Spike Response Model .................................. 8  
  2.1.3 Poisson Neurons ........................................ 8  
2.2 Sparse/Efficient Coding .......................................... 9  
  2.2.1 Kolmogorov Complexity/MDL ...................... 9  
  2.2.2 Independent Components Analysis .................. 10
2.2.3 Bruno Olshausen’s Model ........................................ 12
2.2.4 Laurent Perrinet’s Model ........................................ 13
2.2.5 Nonlinear Sparse Coding ...................................... 14

2.3 Predictive Coding ................................................. 15
  2.3.1 Predictive Information ........................................ 15
  2.3.2 Rao, Zhang, and Ballard’s Model ............................ 16
  2.3.3 Temporal Invariance ......................................... 16

2.4 Relating Perception to Action .................................. 17
  2.4.1 Philipona’s Naive Creature .................................. 17
  2.4.2 Oudeyer’s Curious Dog ...................................... 18
  2.4.3 Bayesian Inference ........................................... 19
  2.4.4 Reinforcement Learning ..................................... 20

3 Predictive Action Theory ........................................ 23
  3.1 Introduction ..................................................... 23
  3.2 Definitions ..................................................... 24
    3.2.1 Predictive Coding .......................................... 24
      Definitions of Predictive Coding ............................. 24
      Model Information Equals Future State Information ..... 28
      Information on Input Statistics can be Stored Up ....... 30
    3.2.2 Predictive Curiosity ...................................... 32
    3.2.3 Predictive Action .......................................... 36

  3.3 Relationships to Other Models ................................. 38
    3.3.1 Temporal Invariance ...................................... 38
      Introduction ................................................... 38
      World Model .................................................. 38
      Static Predictive Coding .................................... 39
      Bandwidth Constraints ..................................... 39
      Optimal Static Predictive Code .............................. 40
      Temporal Invariance ........................................ 41
      Generalization to More Output Dimensions ............... 43
      Conclusions .................................................. 43
    3.3.2 Bayesian Surprise .......................................... 44
    3.3.3 Intelligent Adaptive Curiosity ............................ 45
    3.3.4 Reinforcement Learning .................................. 47

  3.4 Conclusions .................................................... 48
# Coding Models

4.1 Introduction .............................................. 49
4.2 Matching Pursuit based models ........................... 49
   4.2.1 Matching Pursuit .................................. 50
   4.2.2 The Static Model .................................. 52
   4.2.3 Results for Static Images ......................... 53
   4.2.4 The Temporal Model ................................. 54
   4.2.5 Results on Temporal Data ......................... 56
4.3 Predictive Coding Network ............................... 56
   4.3.1 Algorithm Derivation ............................. 57
   Maximizing Information ................................. 58
   Training \( f \) and \( h \) ................................. 63
   The Choice of \( g \) .................................. 64
   Minimizing Spike Count ................................. 65
   4.3.2 Conclusions ........................................ 66

# Predictive Curiosity Models

5.1 Recognizing a Picture ................................... 68
5.2 Playing Mastermind .................................... 72
5.3 Near Optimal algorithm for an MDP ................... 73
   5.3.1 Reinforcement Learning ......................... 80
5.4 Conclusions ............................................. 82

# Conclusion

6.1 Dissertation Contributions ............................. 83
6.2 Future Research Directions ............................. 84
6.3 Concluding Remarks ................................... 84

Bibliography ................................................. 86

Appendices .................................................. 94

A Temporal Invariance and Predictive Coding ............ 94
   A.1 The Damped Brownian Motion Process ............ 94
   A.2 Linear Combinations of DBM are not DBM ......... 96
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.3 DBM is an Information Theoretic Stationary Point</td>
<td>97</td>
</tr>
<tr>
<td>A.4 DBM is a Temporal Invariance Stationary Point</td>
<td>104</td>
</tr>
<tr>
<td>A.5 Value of Predictive Coding Objective Function for a DBM Process</td>
<td>107</td>
</tr>
<tr>
<td>B Predictive Coding Network</td>
<td>109</td>
</tr>
<tr>
<td>B.1 Definition of “synchronous”</td>
<td>109</td>
</tr>
<tr>
<td>B.2 Analytical details of the predictive coding network derivation</td>
<td>110</td>
</tr>
<tr>
<td>B.2.1 Terms equalling zero</td>
<td>110</td>
</tr>
<tr>
<td>B.2.2 Dealing with neurons separately</td>
<td>111</td>
</tr>
<tr>
<td>B.2.3 Derivative of history</td>
<td>113</td>
</tr>
<tr>
<td>B.2.4 Definition of L</td>
<td>113</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Turing Test Questions ............................................. 24
5.1 Saccade Results .................................................... 71
5.2 MDP Exploration Results ........................................ 78
# List of Figures

3.1 Markov Process with Known and Unknown Transition Matrices . . . . 25  
3.2 Information Per Bit Using Exponentially Decaying Bit Frequency . . 32  
3.3 Definition of Curiosity . . . . . . . . . . . . . . . . . . . . . . . . . . 33  
3.4 One Dimensional Codes of a Two Dimensional DBM . . . . . . . . . 41  
3.5 Temporal Variance of Thresholded DBM vs. DBM . . . . . . . . . . 42  

4.1 Matching Pursuit . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51  
4.2 Static Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52  
4.3 Matching Pursuit Results . . . . . . . . . . . . . . . . . . . . . . . . . 54  
4.4 Spiking Network Results . . . . . . . . . . . . . . . . . . . . . . . . . 67  

5.1 Test Images . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70  
5.2 Saccade Filtering . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71  
5.3 Mastermind . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73  
5.4 Maze Exploration Without Wind . . . . . . . . . . . . . . . . . . . . . 79  
5.5 Maze Exploration With Wind . . . . . . . . . . . . . . . . . . . . . . . 79  
5.6 Many Time Steps With Wind . . . . . . . . . . . . . . . . . . . . . . . 80  
5.7 Reinforcement Learning . . . . . . . . . . . . . . . . . . . . . . . . . . 81
What is the chief end of man?
The chief end of man is to glorify God and to enjoy Him forever.

*The Westminster Shorter Catechism, 1642-1647*

To get rich.

*“The Revised Catechism”*
by Samuel Clemens (Mark Twain), 1871

## 1 Introduction

### 1.1 Study of the Brain

The human brain takes in data from a number of different sensory modalities, and outputs commands to muscles all over the body. The resulting behavior is the standard that intelligent behavior is measured against. Since the early days of artificial intelligence, researchers have been trying to create artificial systems which approach this level of intelligence. We are still trying.

Although the brain is the standard, much research has been directed at engineering a solution essentially from scratch. The rationale is often that some algorithm exists which performs intelligently, the brain is simply one hardware implementation of that algorithm, and studying the hardware implementation may do very little to elucidate the nature of the algorithm. This would be the case for studying a computer’s hardware to understand how the Linux kernel operates. One could in principle characterize the nature of transistors, study their connections inside the integrated circuits on the motherboard, figure out how the integrated circuits connected together, study the data inside the memory, put it all together, and figure out how the Linux kernel operates. However, the study of the hardware contributes very little to the understanding of the software; the software is at a completely different level than the hardware. The case could be made that studying neurons in the brain is like studying transistors in the computer processor, and similarly contributes very little to the understanding of the algorithms used by the brain to achieve intelligence.

While the above argument raises the *possibility* that neuroscience may shed little light on the algorithms used by the brain, it does nothing to close off the alternate possibility, that study of the brain could yield great amounts of information. To the contrary,
this latter possibility is much strengthened by two observations. First, a great deal has been learned already by studying the cortex, which is where most of the computation occurs that is of interest here. Van Essen [Essen et al., 1992] constructed a diagram showing which brain functions are performed by which areas of the visual cortex. Such maps directly show how the brain decomposes the problem of intelligence into subproblems. Some sections have been studied well, and some characteristics of their operation are known. For example, Hubel and Wiesel demonstrated that the primary visual cortex contains cells which are responsive to edges of particular orientations, and that these edge cells are organized in a columnar arrangement [Hubel and Wiesel, 1968] [Hubel and Wiesel, 1977]. Such observations seem to convey information about the algorithms used, and not just about the implementations used.

Second, the cortex is basically uniform [Stevens, 1989] [Jerison, 1991] [Ringo, 1991], so observations about one section of cortex tend to be applicable to the rest of the cortex. This is true not only within the brain, but across brains of different species as well. This lends support to the idea that there is really only one algorithm the brain is using, and it is reused by each area of cortex. If indeed there is such an algorithm which underlies the behavior of all of the cortex, then a good understanding of any piece of cortex will include an understanding of this algorithm. This is radically different from the case of the computer, where an understanding of the Linux kernel is distributed over all of the hardware and its global state. With the brain, the problem is simple, in that the solution can be found anywhere.

1.2 A Global Optimization Principle

So what does this algorithm do? The typical answer is that an organism has some criteria on which it judges its bodily state. It measures whether it is hungry, injured, enjoying its mate, or any number of other states of pleasure or pain. Based on this, it has some overall measure of what is called reward, and the purpose of intelligence is to maximize the organism’s reward over the course of its lifetime. In ethical terms, the intelligence of an agent is measured by how selfish it is: the more reward it can accumulate, the more intelligent it is.

Q learning [Watkins, 1989] and TD learning [Barto and Sutton, 1990] are algorithms designed to solve exactly this problem for a restricted world. At each time step, the algorithm is given one of a discrete set of inputs and a scalar valued reward, and it must choose one of a discrete set of actions. The reward is only based on the current state (although there are variations to this), and the current state is only based on the previous state and the previous action taken. For this case, Q learning or TD learning can be used to determine what action should be taken from each state in order to maximize the agent’s expected reward over time. Schultz [Schultz et al., 1997] argues that the neurotransmitter dopamine encodes the reward prediction error, which is the main quantity of interest in the TD learning algorithm. It is reasonable, then, to think that
the algorithm the brain is running is simply a larger scale and more advanced version of TD learning.

A consequence of this conjecture is that everything the brain learns must be learned purely because it is measurably useful to the organism. Cells in the primary visual cortex learn to pick out edges in the visual field, and this must be done because edges make the organism happy. In a way, this is true. Without picking out edges, the organism cannot recognize possible predators, food, mates, etc. But if higher levels of the visual cortex are not yet trained, then picking out edges is of no value at all. Recognizing a predator, or any other object, is a complicated task, as is evidenced by how poorly computer vision works. It requires a good deal of machinery, spanning many layers of cortex in the brain. In order for the visual system to be useful to the organism, much of it must do its job at once. If every cell in the primary visual cortex did its job, but the rest of the visual system did not, it would be of no use to the organism. Sifting through all of the possible ways the visual system could operate may require not only a prohibitive amount of time, but also reward data. We believe it is unlikely that any algorithm, given only reward data, even in fair quantity, will be able to learn to recognize objects without the benefit of great amounts of prior knowledge.

### 1.2.1 An Optimization Principle for Sensory Perception

So how does vision arise in the brain? One obvious answer is that it is simply hard coded. Linsker proposed a model of the visual cortex which caused the neurons to learn to detect edges purely because that was what the network did [Linsker, 1986c] [Linsker, 1986b] [Linsker, 1986a], and Piepenbrock et al. proposed one which did both orientation specificity and ocular dominance [Piepenbrock et al., 1997]. They could have been hard coded to detect these, except that some mechanism had to exist to ensure that they spanned the joint space of edges and ocular dominance. However, Sur rerouted half of the visual input in a ferret to the auditory cortex, and the consequence was that the ferret learned to “see” with its auditory cortex [Sharma et al., 2000] [von Melchner et al., 2000]. Not only did it develop edge detectors, it was actually able to make decisions based on what was in its visual field. This argues that the features which cells seek are not defined a priori, but are chosen based off of the input the cell receives. Somehow, there is something intrinsically interesting about edges in visual data, and the cells in the primary visual cortex figure that out.

For the moment then, let us ignore the larger problem of generating intelligent actions from sensory input, and instead concentrate on determining how sensory input should be processed. As mentioned before, if one piece of the cortex can be understood, it sheds light on the rest of the cortex. Therefore, if sensory processing can be understood, it should shed light on how actions are generated.

Field asked the question “What is the goal of sensory coding?” [Field, 1994]. His conclusion was that the goal is to pick out features which are sparse. A real valued
feature is sparse if its value is usually near zero, but occasionally deviates greatly from zero. Edges are sparse features, because if an edge detector is convolved with an image, then most values will be nearly zero, but a small number will be large. Olshausen [Olshausen and Field., 1996] created a model which took in image data and found the most sparse features of that data, and the resultant features were edge detectors.

A more information theoretic answer is that the goal is to convey as much information about the input as possible while keeping some coding cost fixed. This idea, now termed “efficient coding”, can be traced back to Barlow [Barlow, 1953], who argued that representations of sensory input should have as little redundancy as possible. The filters obtained by optimizing information transfer are similar to those obtained by sparse coding [Rao and Ballard, 1997], but efficient coding lends itself to more general application (see [Karklin and Lewicki, 2003] for an example of finding contours). Unfortunately, neither sparse coding nor efficient coding have a way to separate signal from noise, and they therefore are prone to coding arbitrary amounts of noise.

The solution to this is predictive coding. Predictive coding has its roots in speech processing [Atal and Schroeder, 1967] and states that the goal is to convey as much information about the future of the input as possible while keeping some coding cost fixed. Both efficient and predictive coding have an intrinsic notion of the complexity of a signal, and the object of the network is to convey as much information according to that measure as possible with a fixed coding cost. In the case of efficient coding, the complexity measure is the Kolmogorov complexity [Li and Vitányi, 1997]. In the case of predictive coding, the complexity measure is predictive information [Bialek et al., 2001b] [Bialek et al., 2001a].

Rao [Rao and Ballard, 1997] and Zhang [Ballard et al., 2001] argued that predictive coding is in good accord with the behavior of the visual cortex, but predictive coding has not yet received the attention we feel it deserves. While predictive coding has been used for video compression (e.g. [H.263]), Rao’s and Zhang’s models are the only attempts we know of to relate predictive coding to the behavior of the visual cortex. In contrast to this, each of the following papers presents a unique sparse or efficient coding model in the context of the visual cortex: [Foldiak, 1990] [Olshausen and Field., 1996] [van Hateren and Ruderman, 1998] [Olshausen and Millman, 2000] [Welling et al., 2002] [Grimes and Rao, 2002] [Sallee and Olshausen, 2003] [Perrinet, 2003] [Karklin and Lewicki, 2003] [Triesch, 2004] [Deneve, 2004] [Zemel et al., 2004]. It is worth noting that efficient coding models can be predictive, and some of the above models claim to be predictive coding models. However, if a network is designed to explain its data rather than predict its future data, we are classifying it as an efficient coding network. We believe the reason efficient coding is better studied is simply that predictive coding is harder, and efficient coding has generally not progressed to the point where it just codes noise. A model must have sufficient input noise and nonlinearity before this happens, and the preceding models either avoid this case or hard code a definition of what noise looks like.
1.2.2 A Global Optimization Principle for the Brain

Having found a principle which may explain perception, we now return to the global problem of intelligent behavior. If the sensory cortices are performing predictive coding, then what does that indicate about the motor cortices? Because of the basic uniformity of the cortex, it seems that the task of all cortices are likely to follow the same goal. We argue that this goal is to understand the world. To the sensory cortices, this means that they should try to predict the world, and in doing so, they will learn about the interesting aspects of the world. To the motor cortices, this means that they should try to explore their world, and in doing so, they will gather information about all knowable aspects of the world. The actions of the sensory and motor cortices may thus be governed by a common goal.

Before continuing, it is worth pointing out that this does not deny that an agent can also have a drive to pursue reward. Perhaps remarkably, the algorithms described later in this dissertation which implement the above principle, which I will hereafter call predictive action, also contain a simple means of incorporating other drives into the system. It is thus possible for an agent which does predictive action to also seek reward. However, the predictive action drive must come first, because it is how the agent learns the characteristics of the world it exists in. Once it knows how to navigate the world, it has the tools it needs to maximize any desired reward.

1.3 The Thesis

The chief end of an intelligent agent is to understand the world around it.

1.4 Dissertation Contributions

The contributions of this dissertation can be divided roughly into two parts: theoretical and algorithmic. The theoretical contributions of this dissertation include reformulating predictive coding in terms of Shannon information, developing a general definition of curiosity, merging these two definitions into one single principle, and showing the relationship of this principle to other important principles in machine learning. The algorithmic contributions include a novel efficient coding algorithm, a novel predictive coding algorithm, and a novel algorithm to which demonstrates curiosity on a Markov Decision Process world. This last algorithm is demonstrated on stochastic maze problems, and in addition to investigating how well it explores the maze, we use it to demonstrate a link between curiosity and reinforcement learning.
1.5 Organization

The rest of this dissertation describes the predictive action principle in detail and demonstrates aspects of its behavior. Chapter 2 describes the prior work which is related to this thesis. Chapter 3 describes the theory of predictive action, including a reformulation of predictive coding and a definition of predictive curiosity, and its relation to existing principles. Chapter 4 describes two neural networks which perform sensory processing. Chapter 5 describes three examples of algorithms performing predictive curiosity. Finally, chapter 6 describes the conclusions reached by this dissertation and the work which is left to do.
2 Background

This chapter provides the backdrop of ideas against which the rest of this dissertation is set.

2.1 Neuron Models

2.1.1 Integrate and Fire Model

One of the simplest models of neuron behavior is the Leaky Integrate and Fire (LIF) model [Hill, 1936] [Stein, 1965]. This model states that there is an input current $I$ to the neuron. The neuron has a potential given by:

$$\frac{dV(t)}{dt} = -\alpha V(t) + \beta I(t)$$

where $\alpha$ and $\beta$ are tunable constants. When the potential reaches a threshold $T$, the neuron fires, and $V$ is set back to zero.

Often, $I$ is considered to arrive in spikes from other neurons, weighted by some synaptic weights. To express this, let us discretize time. Let the spike train be denoted by $S$, where $S_{i,t} \in \{0, 1\}$ and $S_{i,t} = 1$ exactly if neuron $i$ is firing at time $t$. Let $V_{i,t}$ be the potential of neuron $i$ at time $t$, and $w_{i,j}$ be the synaptic weight that neuron $i$ assigns to a spike arriving from neuron $j$. $V$ and $S$ are then given by:

$$V_{i,t} = \begin{cases} 
(1 - \alpha)V_{i,t-1} + \beta \sum_{j \neq i} w_{i,j} S_{j,t-1} & \text{if } S_{i,t-1} = 0 \\
0 & \text{else}
\end{cases}$$

$$S_{i,t} = \begin{cases} 
1 & \text{if } V_{i,t} \geq T \\
0 & \text{else}
\end{cases}$$
There are a number of modifications which have been done to this model to make it fit real data better (e.g. [Brette and Gerstner, 2005]) but this suffices to explain the general concept.

### 2.1.2 Spike Response Model

An alternate deterministic model is the Spike Response Model (SRM) [Gerstner and van Hemmen, 1992] [Gerstner and van Hemmen, 1993] [Gerstner et al., 1993]. The most important difference between the LIF model and the SRM model is that the SRM model is not described by a differential equation. Instead of integrating over presynaptic spikes, the neurons convolve their presynaptic spikes with kernel functions. Instead of resetting $V$ to zero after a spike, a function is subtracted for each spike to simulate the refractory period. If we denote the kernel function $K_{i,j,t}$ and the refractory function $R_t$, then the system can be described by:

$$
V_{i,t} = \sum_{j \neq i} \sum_{\tau < t} K_{i,j,t-\tau} S_{j,\tau} - \sum_{\tau < t} S_{i,\tau} R_{t-\tau}
$$

$$
S_{i,t} = \begin{cases} 
1 & \text{if } V_{i,t} \geq T \\
0 & \text{else}
\end{cases}
$$

Note that if $K$ and $R$ are exponential decays, then the SRM can be made equivalent to the LIF. However, typically $K$ and $R$ are chosen to be time bounded functions so that $V$ can be computed given a finite amount of history. One common choice is $R_t = \infty$ for $t < t_r$, and zero otherwise, where $t_r$ is the absolute refractory period. This ensures that the neuron cannot fire twice in $t_r$ time steps. Also, $K_{i,j,t}$ is usually independent of $i$ and $j$. We allow it to be general here because it is used in its general form in [Keat et al., 2001].

### 2.1.3 Poisson Neurons

Poisson processes [Snyder and Miller, 1991] have also been used to model neurons. In the previous two models, the neuron spiked when $V$ crossed the threshold $T$. In Poisson models, $V$ is taken as an instantaneous firing rate. That is to say that instead of defining $S$ as

$$
S_{i,t} = \begin{cases} 
1 & \text{if } V_{i,t} \geq T \\
0 & \text{else}
\end{cases}
$$

consider time as continuous again, and let $S$ denote a set of tuples such that $(i, t) \in S$ means that neuron $i$ fired at time $t$. A Poisson process is defined such that

$$
\forall t. \quad \lim_{\Delta t \to 0} \Pr [\exists \tau. \quad t < \tau \leq t + \Delta t \land (i, \tau) \in S] = V_i(t)\Delta t
$$
where $V$ may be defined as in either of the preceding sections, and $\wedge$ denotes a logical and.

## 2.2 Sparse/Efficient Coding

### 2.2.1 Kolmogorov Complexity/MDL

Suppose that one has a single chunk of data of unknown origin, and one would like to know as much about its structure and complexity as possible. If there are several available models which could have generated this data, then Occam’s razor provides an intuitive answer to this question: the simplest solution is probably the best.

This principle has been rigorously codified into what is known as Kolmogorov complexity [Li and Vitányi, 1997]. The space of models considered is the space of computer programs. If a given computer program generates the given data, then it is a candidate model. The shortest computer program which generates the data is the best model, and the complexity of the data is the length of that program.

To see why this works, let us consider a few examples. First, suppose that the data is 1000 bits of binary data, starting with a zero, and every bit is different from the previous bit. The simplest computer program which generates this data is a program which counts from zero to 999, and which outputs the low order bit of the counting index. Such a program is very short. On the other extreme, consider 1000 bits of data which was generated by tossing a fair coin and writing a zero for heads and one for tails. Although it is possible that the data sequence could be simple to generate, as in the previous case, it is vastly more likely that the best computer program to generate this data just has the sequence hard coded. The length of this program is 1000 plus a little bit to dump the data out. The interesting cases usually lie in between these extremes.

Suppose that the data is the alternating bit sequence as generated by the first program, except that each bit is flipped with probability 1/100. While some such sequences are simpler (such as the sequence where no bit was flipped), the best program to generate the sequence would probably hard code the set of bits that were flipped and assume the rest are generated normally. If the set of bits that are flipped is encoded efficiently, then this program should be fairly short.

Programs to generate an unknown block of data can be thought of as having a program segment and a hard coded data segment. The program segment encodes the probability distribution of the data, and the data segment encodes the actual sample, compressed using the distribution coded in the program segment. Occam’s razor could be rephrased to state that the simplest explanation which explains the data reasonably well is the best. The “simplest model” means short program segment, and “explains the data reasonably well” means short data segment, and their lengths are simply summed up to determine how good the model is.
One possible goal of sensory coding is to efficiently code the sensory input. If one were to make a good model of visual input, it would probably be some form of object and lighting models, rendered onto the retina, with a little noise added. If this is the model which best codes the visual input, then it seems reasonable that the brain might use a model of this kind purely because it best codes the visual input. Therefore, in order to duplicate the brain’s amazing perceptual abilities, one simply has to write a program that figures out how best to code the input it is given.

Unfortunately for Kolmogorov complexity, it is in general uncomputable to find the shortest program which generates the given data. At least part of this is because there exist programs which run forever without ever outputting anything, and it can be impossible to know whether a given program will do this. So if one is to search the space of programs, then one may end up waiting forever to determine whether a given program is better than another.

Fortunately, the Minimum Description Length principle approximates Kolmogorov complexity in a way which is tractable. MDL states that if one has a chunk of data and a set of models, then the model which explains the data best is the best model. The measure of how well a model explains its data is the length of a short program implementing the model plus the compressed length of any leftover data needed to explain the data chunk. The chief difference between Kolmogorov complexity and MDL is that MDL considers only a set of models, and typically the set of models considered are all tractable models. For example, if the data chunk consists of 1000 samples of Gaussian data, then the set of models might be Gaussians of any mean and variance. Given some prior distribution on the mean and variance, the value assigned to any hypothesis would be the coded length of the mean plus the coded length of the variance plus the coded length of the 1000 samples, compressed using the coded mean and variance. Often using MDL requires the specification of arbitrary priors, such as the priors for the mean and variance. Still, it has proven useful despite its weaknesses.

2.2.2 Independent Components Analysis

Algorithms which attempt to perform efficient coding usually try to find a factorial code for their input. A factorial code is a deterministic translation from input vectors to output vectors, such that the elements of the output vectors are independent. In concert with this, one often puts a prior distribution on the output vectors. If the coded input is indeed made of independent elements and each element follows the given distribution, then the input can be optimally coded [Olshausen, 1996]. One only has to know how to code variables distributed according to the given distribution, and then each element of the output is coded separately using this code.

One method of coding an input as independent components is Independent Component Analysis (ICA) [Bell and Sejnowski, 1995]. ICA assumes that there are several independent scalar variables from the world, and that they are linearly mixed to form
The input. The mixing matrix must be square and nonsingular for ICA to work properly. The goal of ICA is to learn from this mixed data a demixing matrix, such that when the input is run through the demixing matrix, the result is the original world vector (except for a possible reordering and scaling of the components). Let us call this reconstructed vector $Y$. To find the demixing matrix, ICA minimizes the Kullback-Liebler divergence between the distribution of $Y$ and the product of the marginal distributions of the elements of $Y$, which is:

$$\tilde{p}(\vec{y}) = \prod_i p(y_i)$$

If the elements of $Y$ are indeed independent, then the Kullback-Liebler divergence between $p(\vec{y})$ and $\tilde{p}(\vec{y})$ will be zero. Thus minimizing this Kullback-Liebler divergence causes the algorithm to construct a demixing matrix that makes $\vec{y}$ equal to the original independent world vector, up to scaling and permutation of the elements.

In order to test whether efficient coding provides an explanation of the behavior of the brain, Van Hateren and Ruderman gave ICA whitened natural images as input [van Hateren and Ruderman, 1998]. They whitened the images first, because that is essentially what the lateral geniculate nucleus does to the image before passing it to the primary visual cortex [Atick, 1992] [Atick and Redlich, 1992]. ICA was thus given input analogous to the input to the primary visual cortex. The learned demixing matrix could then be viewed as a bank of image filters, because each output element from ICA is the dot product of one row or column of the matrix (depending on whether the matrix is left or right multiplied to the input) and the input image. When these filters were plotted, they resembled the Gabor filters. Individual cells in the primary visual cortex also appear to act as image filters, and when these filters are measured, they also look like Gabor filters [Hubel and Wiesel, 1968].

While this result is promising for the case that the brain does efficient coding, the standard ICA algorithm has weaknesses. Its two major weaknesses are that it is linear and that the representation it forms is complete. The first is a weakness because it means that hierarchical application of ICA accomplishes nothing. If the output vector of ICA is taken as the input vector to another instance of ICA, then the second instance will learn a demixing matrix that only scales and permutes the input. It cannot make the input more independent, because its output is the original input to the world multiplied by both demixing matrices, and multiplication by two matrices is no more powerful than multiplying by only one.

That ICA is complete is a weakness, because overcomplete representations are useful for making efficient codes [Lewicki and Sejnowski, 2000]. To see this, consider a two dimensional dataset, which when plotted, appears as three noisy lines emanating from the origin, and separated from each other by 120 degrees. For any complete representation (so the output is also two dimensional), at least one of these three spokes will generate high values in both dimensions. However, one could imagine a system which could take this two dimensional output and map it into three dimensions, so that
each spoke mapped to a dimension, and the distance along each spoke from the origin
determined the value of the corresponding element of the output. In such a representa-
tion, only one output element would be very large at a time. If each output were coded
according to its marginal distribution, this would be a much tighter encoding of the
input than any complete representation could do.

2.2.3 Bruno Olshausen’s Model

Olshausen developed several models which allow the representation to be over-
complete [Olshausen and Field., 1996] [Olshausen and Millman, 2000] [Sallee and
Olshausen, 2003]. Each of these algorithms place a (possibly parameterized) prior dis-
tribution on the output elements. In the case of the first model, which we will focus on
for the rest of the section, the output elements were assumed to be independent and each
distributed according to a Cauchy distribution. The model is generative, in the sense
that it describes a way to generate pseudo-data. To make an input sample, first generate
an output sample, according to the distribution of outputs. Next, take each output as
a specification of how much of its corresponding filter to add to the input, and add up
the proper amounts of each filter. Finally, add a little Gaussian noise. To put this in
rigorous terms, let $\vec{x}$ be an $N$ dimensional input image, $\vec{y}$ be the $M$
dimensional output values (where $M$ does not have to equal $N$), $\vec{n}$ be an $N$
dimensional sample of Gaussian noise, and $B$ be an $N$ by $M$ matrix such that $\vec{b}_i$
is the basis vector corresponding to output $i$. Then an input sample can be represented as is given by:

$$\vec{x} = \vec{n} + B\vec{y}$$

For any given input $\vec{x}$, there is a nonzero probability that it was generated by a given $\vec{y}$.
Since the object of the network is to generate an efficient code for $\vec{x}$, some particular
value of $\vec{y}$ has to be chosen. For each $\vec{x}$, the network chooses the $\vec{y}$ which is most
likely to generate $\vec{x}$. If one views the world as a reflection of the network, and the input
therefore as being generated by a $\vec{y}$ in the world, then the output of the network $\vec{y}$
is a MAP estimate of $\vec{y}$. There is a distribution of $\vec{x}$ when generated by the model, and there
is a distribution of $\vec{x}$ when generated by the world. The learning goal of the network
is to minimize the Kullback-Liebler divergence between these two distributions. If the
network could bring the divergence to zero, then the network has succeeded in learning
the distribution of inputs, and could code the input efficiently.

Because the output of the network is characterized by a Cauchy distribution, and
samples from a Cauchy distribution are rarely far from zero (but when they are far, they
are very far), the model is called sparse coding. The model just described was the first
sparse coding model, and there are a number of others with variations in the distribution
used, the noise model, etc. All of the sparse coding models, when applied to whitened
natural images, develop Gabor filters.
A weakness of these models is that their outputs are characterized by real numbers. In the brain, the output of a layer of cortex is a spike train. While these models show that efficient coding may be a plausible explanation of the brain’s operation, they say little about how the brain might actually use efficient coding.

### 2.2.4 Laurent Perrinet’s Model

Perrinet provides a model of the cortex which uses spiking neurons to perform efficient coding [Perrinet et al., 2001a] [Perrinet et al., 2001b]. His model assumes that the image is presented to the retina at some known start time, and that following this presentation, there is a wave of spikes to signal the image. This wave is called a volley. The volley could be triggered by a saccade, or it could be from a 50 Hz clock that has been observed in the primary visual cortex in some experiments.

The information in the volley is coded by the order in which the neurons fire. The neurons are integrate and fire neurons, so the more similar the input is to their receptive field, the faster they fire. Thus the neurons whose receptive fields match the input the best fire first, which is why rank conveys information. When two neurons have similar receptive fields, the firing of one inhibits the firing of the other by an amount related to the correlation between their receptive fields. This nicely allows for overcomplete representations, since two neurons can have very similar receptive fields, and when one fires it almost completely inhibits the other. The reconstructed input is a linear sum of the basis functions of each neuron that fired, and the amount of each basis function used is determined by the rank of the neuron. This algorithm is identical to the matching pursuit algorithm [Mallat and Zhang, 1993] except that the amount of each basis vector is coded by the rank instead of by an analog quantity.

Unlike most other sparse coding models, the input to Perrinet’s model is not analog. Instead, he took analog images and simulated the activity of integrate and fire on-center and off-center LGN neurons. The outputs of the simulated LGN fed his simulation of V1. The V1 neurons update their receptive fields by a Hebbian update rule that increases the weights of afferent neurons that fire right before the neuron fires and decreases the weights of afferent neurons that fire right after it does. Again, the receptive fields become Gabor filters.

In an efficient coding of a spike train, the number of bits required to code the train will increase monotonically with the number of spikes. Therefore, the efficient coding principle, when applied to spike trains, is that the input should be coded using as few spikes as possible. In this model, the result of learning is that the number of spikes in order to code the image to some desired accuracy does decrease over time.

Again, the receptive fields learned look like Gabor filters. However, while this may look like a plausible model of what neurons are doing in the brain, there are still a few difficulties to work out. First, Perrinet’s papers do not appear to specify a local learning rule for how much each neuron should inhibit other neurons. It would appear that this
learning requires non-local knowledge, which real neurons do not have. Second, while this method does sparsely code static images nicely, it leaves something to be desired when it comes to video. In order to code video, it must separate video into frames and have one frame per volley. However, for a spike train to have maximal entropy, and therefore be able to convey maximal information, it must be Poisson. Neurons which operate in volleys are clearly not Poisson, and therefore are not coding the information as efficiently as possible. Finally, and most important, the reconstruction of the input from the output is still a linear operation, which means that the network cannot learn more interesting things by being layered.

2.2.5 Nonlinear Sparse Coding

There has been some work on developing nonlinear efficient coding algorithms. Hyvärinen and Hoyer [Hyvärinen and Hoyer, 2000] suggest one means of doing this. Their approach assumes a two layer network, where one layer corresponds to simple cells and the other layer corresponds to complex cells. The complex cells are wired up to disjoint sets of four simple cells, and they square the outputs of the simple cells, sum the squares, and take the square root. The goal is to find a set of filters for the simple cells such that two simple cells which are connected to different complex cells have independent outputs, and the outputs of the complex cells are sparse. They provide a learning rule to accomplish this, and show that it does result in cells similar to complex cells. However, the approach seems extremely restricted, because it is specifically tuned for a type of regularity they expected to find in the data. One would therefore not expect this to work on different types of data.

A more general approach comes from Karklin and Lewicki [Karklin and Lewicki, 2003]. In this approach, they begin by running ICA on the input images. Next they consider the joint distributions of the coefficients of the ICA filters for each image patch. If the filters truly had independent outputs, then the joint distribution of those coefficients would just be the product of the marginal distributions for each filter. However, since the filters are linear, there exist residual dependencies among the filter outputs, so there is structure in the joint distribution. To model this, a particular form of the structure is assumed, and it is parameterized by a set of coefficients. They assume that these coefficients can be represented by a basis set with a sparse distribution. The key is then to model the residual structure among the coefficients of the ICA filters using a sparse basis set. By doing so, they end up finding textures and similar such perceptually salient image features. Unfortunately, to date this model is restricted to a two layer network.
2.3 Predictive Coding

2.3.1 Predictive Information

Kolmogorov complexity specifies the complexity of an arbitrary chunk of data in the absence of any other information about it. However, if something is known about the data, then other measures can be placed on it. One such measure is predictive information [Bialek et al., 2001b] [Bialek et al., 2001a].

The setting of predictive information is that one is given a distribution of data in advance, and the data consists of a possibly infinite number of samples in a temporal sequence. If one has observed all of the samples up to time $t$, then what can be said about the remaining samples? The answer is just the Shannon information $I(X^t; X_{t+1})$. This measure is called predictive information.

Unlike Kolmogorov complexity, predictive information is a curve, since it is a function of $t$. Surprisingly, this curve will asymptotically approach only one of three forms: constant, logarithmic, or power law. It will be constant if a finite number of previous samples is sufficient to convey all information about the future. An observable Markov process with a known transition matrix falls into this class. The last sample conveys everything that can be known about the next sample, which in turn conveys everything that can be known about the one after, and so on. Therefore, knowledge of the last sample is sufficient to convey everything that can be known about the entire future sequence, so the predictive information is constant with time.

Another different case occurs when the model is parameterized by a finite number of unknown parameters. In this case, the information grows logarithmically. The simplest example of this is a coin of dubious fairness. If the probability of heads is evenly distributed between 0 and 1, then the more samples one has observed, the better one can predict the future samples. However, if one knows the total number of heads and the total number of tails that have been observed, then knowing the actual sequence of them contributes nothing to the distribution of future sequences. The number of bits in these counts grows logarithmically with the number of samples, so it is not surprising that the information about the future also grows logarithmically.

The third case occurs when there are an infinite number of unknown parameters in the model. In this case, the amount of information one has about each parameter may grow logarithmically, but the sum of the infinite collection of logarithmic curves converges on a power law with an exponent strictly between zero and one. Models with this property must have interactions on a range of time scales, and the interactions must be different. Fractional Brownian motion, for example, has interactions on all timescales, but if one characterizes the correlation between a point at time $t$ and the points at time $t - 1$ and $t - 2$ completely, then the fractional exponent can be determined, and interactions on all other timescales are known. Therefore, the predictive information of fractional Brownian motion with an unknown exponent only grows logarithmically. In
order to achieve power law growth, the correlations on a finite number of time scales must not determine the correlations on the other timescales.

It has been suggested that the goal of sensory coding in the brain is to choose the model which minimizes the minimum description length of the input. Another principle which has been suggested is that the output of a layer of neurons should be chosen to maximize the information between the chosen output and the future of the input. If this were done perfectly, then the information conveyed would equal the predictive information of the input. Therefore, predictive information may be a fundamental quantity in understanding sensory coding.

2.3.2 Rao, Zhang, and Ballard’s Model

Rao and Ballard proposed a predictive coding model using two layers of neurons which could output continuous values [Rao and Ballard, 1999]. The first layer was analogous to the LGN, and the second layer was analogous to V1. The idea was that the activity of V1 inhibited the activity of the LGN, so that the LGN only represented what had not been predictive by V1. This is similar to Perrinet’s model, except that the cells inhibited the input instead of each other.

Zhang and Ballard extended this model to work with spiking neurons [Ballard et al., 2001]. As in Perrinet’s model, the spikes come in volleys. However, instead of using the rank in the volley as the indicator of the correlation between the input and the neuron’s receptive field, their model used Poisson neurons, where the probability of a neuron firing in a given volley was related to the correlation of the receptive field to the input.

The model is an improvement over previous models in that it does predictive coding instead of efficient coding, but it suffers from most of the same difficulties as Perrinet’s model. The reconstruction is still linear, and the use of volleys diminishes the amount of information that neurons can convey.

2.3.3 Temporal Invariance

There is another unsupervised learning principle which attempts to dictate optimal representations of data. Temporal invariance claims that the input should be represented by a set of values which change as slowly as possible. The intuition is that all of the things in the real world which we care about (who we are talking to, where they are, even what expression is on their face) all change much slower than the actual inputs we have from the world (the pressure on our eardrums, the visual input to a particular retinal photoreceptor). Therefore, if we extract properties of the world which change slowly, we are likely to extract the properties of the world we actually care about.

In addition to reconstructing receptive field properties well, there exists a very nice way to train nonlinear mappings from the input to the output using this principle [Wiskott and Sejnowski, 2002]. In chapter 3 of this dissertation, we argue that
this principle is actually an implementation of predictive coding using a certain set of assumptions. Therefore, all of the success of temporal invariance is also success for predictive coding.

\section{Relating Perception to Action}

The goal of intelligence is more than just to perceive the world. Action must be taken. Regardless of what the goal is that drives the agent, there are certain things that an agent needs to be able to do in order to act intelligently. First, the agent must be able to determine the general characteristics of its own body and of the world. Then it must be able to explore the world and learn how it works. It must be able to combine information from multiple modalities in order to determine what the state of the world is at a given moment in time. Finally, it must be able to use that information in order to plan actions which help it to achieve its goal. This section will cover the current approaches to these problems.

\subsection{Philipona’s Naive Creature}

An organism has a brain which receives input and generates output. If the organism has no knowledge at all about how these inputs and outputs relate to each other, or about its own body, or about the nature of the world that it exists in, then is it possible for the organism to learn these things? Philipona \textit{et al.} argue that it is indeed possible for an agent to find answers to these questions without any prior knowledge [Philipona \textit{et al.}, 2003] [Philipona \textit{et al.}, 2004]. This is quite useful, as it implies that very little needs to be hardcoded into the mechanism of an agent’s brain in order for the agent to function intelligently.

In [Philipona \textit{et al.}, 2003], they consider the case of an agent whose body consists of an arm permanently fixed to the ground. It has a few joints, and then an articulated hand at the end. On the end of each finger, there is an omnidirectional light sensor, which may also have a controllable aperture. The environment consists of floating lights which drift around. The input to the brain is two fold. There are the inputs from the light sensors, and there are proprioceptive inputs to indicate the positions of the arm and hand joints and the controllable apertures. The output of the brain is interpreted as controls for the arm, hand, and apertures.

The goal of the agent is to learn to describe the relationship between its inputs and its output using few parameters (much like efficient coding). The agent begins its learning by separating proprioceptive input from “exteroceptive” input (input from the environment). This separation takes place on the basis of the fact that proprioceptive input is assumed to be completely deterministic, whereas exteroceptive input is not. Therefore, if the agent keeps all of its outputs at zero, then only exteroceptive input can
change. Separating these types of input means it can learn the relationships between its output and the proprioceptive input independently of the exteroceptive input. Once this is done, then it has implicitly developed a separate notion of “self” and “world.”

Next, it finds movements it can make which undo movements in the world. For example, it can move one of its fingers to keep the light level constant. By performing such movements, it can determine the number of degrees of freedom and the structure of the world.

They simulated this system and used PCA to analyze the results. They found that their algorithm was able to correctly identify all of the degrees of freedom in the input and output. Their algorithm is limited to smooth continuous spaces, and its characterization of the world consists mostly of the number of degrees of freedom inherent in it. While this is far from a complete description of the agent’s input, it seems a promising beginning to the problem.

### 2.4.2 Oudeyer’s Curious Dog

Oudeyer et al. study the problem of making an agent curious about its environment [Oudeyer and Kaplan, 2004] [Oudeyer et al., 2005]. They use a Sony Aibo robotic dog to explore a playground. The dog can look around, bite an object, or bash an object, and these motions are parameterized by five continuous values. The robot can perceive object identity, distance to an object, and whether it is biting something, and these are discrete quantities. The robot is given no information about its sensors or its actions, and it is set in a playground consisting of a number of objects, some of which can be bitten and others can be bashed.

The robot’s brain consists of two modules. One attempts to predict future input based on current input and output. The second module attempts to predict the error in the predictions from the first module, as a function of the current input and action. Since the prediction error is measured in a task dependent fashion, the second module effectively measures how well the robot can predict the result of certain actions in certain circumstances. The goal of the robot is in each circumstance to choose the action for which the prediction has been decreasing the most rapidly. Therefore, the robot does whatever task it thinks it can learn the most by doing. In this fashion, it is curious.

The result is that the robot goes through several learning stages. First, it just tries random stuff. Then it just looks around, and often not even at objects. Next, it goes to bashing and biting, and notices that this sometimes has results. Then, it starts to look towards objects, and tries to bash and bite them. Finally, it learns which objects can be bashed and which can be bitten. Therefore, the robot’s curiosity has succeeded in causing it to learn about the world.
2.4.3 Bayesian Inference

Consider now an agent which knows how its own sensors and body are controlled, knows how the world behaves, and knows how its actions affect the world. Unfortunately, the world may not behave deterministically, and the agent’s sensors might have noise, and the agent’s body might also not perform deterministically. How can the agent deal with such uncertainty? The answer is Bayesian inference.

Bayes rule is a well known theorem from probability theory which states the following [Freund, 1992]:

\[ p(s|d) = \frac{p(d|s)p(s)}{\sum_{s' \in S} p(d|s')p(s')} \]

To see how this can be used, suppose an agent has data \( d \) from its sensors and wishes to know the world state \( s \). In general, it cannot know what \( s \) is with certainty because of sensor noise. However, it can know the probability of each state given the data \( p(s|d) \), provided that it knows two things. First, it has to know the probability of a particular state generating the given data \( p(d|s) \). Second, it has to know the prior probability of each state \( p(s) \). If it knows these, then in principle it can use Bayes rule to figure out the probabilities of each state given the data. This is called Bayesian inference.

Bayesian inference is often difficult to do in practice for several reasons. First, one frequently does not know these distributions in advance, so they must be inferred from data. Second, these distributions are often continuous, so they can only be approximately learned. For better approximations, more data will be required. Third, for continuous distributions, the denominator becomes an integral which is often analytically intractable. For some applications, such as just choosing the most likely world state given the data, the denominator can be ignored, but for many applications, it must be dealt with somehow.

There have been several methods proposed for performing Bayesian inference using networks of spiking neurons. Pouget, Deneve, and Latham [Deneve et al., 2001] [Deneve and Pouget, 2003] [Latham et al., 2004] propose one such model which uses population codes to represent head direction, eye direction relative to the head, and absolute eye direction. Each neuron in the code uses a bell shaped tuning curve centered at a neuron dependent location. Head location, for example, is then represented by a bell shaped curve along the set of neurons which are encoding it. Ideally, the curves for all three variables would be clean curves, but in reality, they will be noisy. The network has a basis function layer which is connected to all three population codes in a recurrent fashion. The network functions as a line attractor, where the stable states along the line are states where the population curves are clean bell curves with the proper relationships between them (namely that the absolute eye angle is the sum of the head angle and the eye angle relative to the head). The operation of the network is such that, when given noisy inputs, it makes a maximum likelihood estimate of the three variables and
causes their population codes to converge on clean bell curves centered at the proper locations.

Another method of doing Bayesian processing using neurons is proposed by Barber, Clark, and Anderson in [Barber et al., 2003]. This method consists of assuming that there exists basis functions for the marginals of quantities of interest, and that the neurons signal the coefficients of those basis vectors given the input. Therefore, the representation is a fairly explicit representation of the probability of the variables of interest given the input. Unfortunately, the theory appears only to deal with the marginals of the variables of interest. In reality, these variables have a joint distribution that one would like to reason about. However, if the neurons tried to represent the joint distribution using such a basis function approach, then they would require an exponential number of basis functions relative to the number of variables of interest.

Rao proposed a method of using Bayesian inference to keep track of the log probabilities for different states of a hidden Markov model given the observations [Rao, 2004]. In this case, the number of states is discrete, and Bayesian inference can be done exactly. However, Rao concludes that it cannot be done exactly using plausible spiking neurons, and therefore resorts to introducing a fairly serious approximation into the process. Deneve [Deneve, 2004] and Zemel [Zemel et al., 2004] also propose models of Bayesian inference in spiking neurons which essentially assume that the input is generated from a hidden Markov model. However, their Markov models are made of binary values which occasionally flip between zero and one in a fashion independent of each other, and this is even more limited than the case Rao considers.

While work on spiking neurons performing Bayesian inference has been generally set in the context of sensory integration, it is well to note that the purpose of sensory integration is to be able to act better. As a clear example of this, Körting et al. gave subjects a task where they were required to move their hand along a line through a series of force fields (provided by a robotic arm) [Körting et al., 2004]. To do the task well, they had to learn a prior distribution of force fields and, when partway down the line in a given trial, they had to use the forces they had already observed and the prior distribution they had already learned in order to guess what the forces were going to be along the rest of the line. The study concluded that the behavior of the subjects was consistent with the subjects using Bayesian inference for this task.

\subsection{Reinforcement Learning}

At this point, our agent has learned to distinguish itself from the world. It knows how to control its body, and it knows how its actions will affect the world given whatever state the world is in. It knows how to judge the world state optimally from its sensors. But now what? The traditional answer is that the agent must have something it considers rewarding, and it should seek this reward. To this end, reward is defined as a function of world state, or sometimes as a function of world state and the action of
the agent. The agent should take the actions which maximize its expected reward over time. This is called reinforcement learning.

The majority of the work done on reinforcement learning has been in the context of Markov decision processes (MDPs). In this context, the world state is fully observable, and the next world state is stochastic and conditioned only on the current world state and the current action. In order to maximize reward over time, the agent must not only seek to maximize its reward at the present moment. A greedy policy such as this could lead the agent to take an action which results in immediate reward, but leads to a state from which the agent can never get any more reward. Such behavior certainly does not lend itself to maximizing expected reward over time.

Watkins proposed a method for solving the reinforcement learning problem on MDPs called Q learning [Watkins, 1989]. To solve the problem just mentioned, which is usually referred to as the problem of delayed rewards, he introduced a function $Q(s, a)$. For every state $s$ and every action $a$, $Q(s, a)$ equals the expected amortized reward over time for taking that action from that state, and following the policy from there out. To understand this, suppose that for a given stretch of time, the agent gets reward $r_t$ at time $t$. Its amortized reward $\tilde{r}_t$ is given by

$$\tilde{r}_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau} = r_t + \gamma \tilde{r}_{t+1}$$

Thus if at time $t + \Delta t$, the agent receives reward $R$ and it receives no other rewards, then $\tilde{r}_t = R\gamma^{\Delta t}$.

Of course, the agent does not know at time $t$ what is going to happen in the future, because the world is stochastic. Further, the reward the agent gets is dependent on its policy. To handle this, suppose that the agent has a fixed policy to always take action $A(s)$ from state $s$. Under this policy, there is an expected reward from any given state and action given by:

$$Q(s_t, a_t) = \langle \tilde{r}_t \rangle$$

It can be learned because

$$Q(s_t, a_t) = \langle \tilde{r}_t \rangle = \langle r_t \rangle + \langle \gamma \tilde{r}_{t+1} \rangle = \langle r_t \rangle + \gamma \langle Q(s_{t+1}, A(s_{t+1})) \rangle_{s_{t+1}}$$

As the agent progresses according to policy $A$, it receives samples of $r_t$ and of $s_{t+1}$, and it can take the expected values of these online. Over time, the $Q$ values will converge on optimal estimates of $\langle \tilde{r}_t \rangle$. In order to find the optimal policy, the agent should perform the action $a$ from state $s$ for which $Q(s, a)$ is a maximum. Of course, as the policy changes, the $Q$ values change. Fortunately, for a reasonable set of conditions, this is guaranteed to converge on the optimal policy.
Reinforcement learning on MDPs is often solved using Q learning or TD learning [Barto and Sutton, 1990] (a modification where the Q values propagate faster), and, if one is willing to wait a bit longer, it can be solved nearly optimally in polynomial time using an algorithm due to Kearns and Singh [Kearns and Singh, 1998]. However, while the MDP case is more or less solved, no such algorithm exists for reinforcement learning on partially observed Markov decision processes (POMDPs). In this case, each world state probabilistically emits an observation to the agent, so the agent in general does not have enough information to know for certain what the state of the world is at any point in time. Some work has been done on worlds such as this. Sondik showed that for finite horizon worlds, the optimal policy can be represented with a finite number of parameters [Sondik, 1971]. Cassandra provided an optimal algorithm for solving POMDPs [Cassandra et al., 1994], but it is computationally intractable. Chrisman proposed the perceptual distinctions approach [Chrisman, 1992] which tries to build a model of the underlying world based on predicting future inputs. This approach is general, but often requires too much data to be practical. McCallum proposed the utile distinctions approach [McCallum, 1995] which tries to build a model of only the aspects of the underlying world which predict reward. It is good at handling worlds where only a fairly small number of past observations are needed in order to determine the optimal action. Parr [Parr and Russell, 1995] proposed an approximate method for solving arbitrary POMDPs, but it is not clear how effective the algorithm is. Weiring proposed a method for solving large POMDPs, but it only works if the POMDP has some unknown but very simple structure [Weiring and Schmidhuber, 1996].
3 Predictive Action Theory

3.1 Introduction

In 1950, Alan Turing proposed what has become known as the Turing test [Turing, 1950]. The goal of the Turing test is to determine whether or not a human judge, who may communicate with two contestants using only text, is able to decide which of the contestants is the human and which is the machine. The human is assumed to act human, and the question is whether or not the machine can act sufficiently human to fool the judge.

Suppose that we restrict the setting even further. Instead of allowing communication between the judge and the contestants to be text, suppose that all communication takes on the form of binary strings which have no prearranged meaning. How can the judge decide whether or not a contestant is human? There are at least two possibilities: the judge may attempt to draw on knowledge which only a human is likely to have (which would be difficult to do in this domain), or the judge may test the intelligence of the contestants.

Suppose the judge chooses to test the intelligence of the contestants. The judge’s questions and the contestant’s responses could be as shown in Table 3.1A. The judge can conclude from this that the contestant can recognize prime numbers, since his questions are prime numbers in a unary encoding, and the contestant’s responses are the next prime numbers in unary encoding. Of course, the contestant did not know what was going on for the first question, and made a sensible guess for the second question.

Another test the judge could do might be as shown in Table 3.1B. In this case, the contestant showed that he can learn to manipulate the judge’s question. The judge’s question consists of one single bit on, and if the contestant’s response has only an adjacent bit on, then the judge’s next question is the contestant’s previous response.

These are two very different tasks, but they both demonstrate the intelligence of the contestant. The single heading of “intelligence” thus seems to encompass both perceptions and actions. In this chapter we attempt propose a rigorous definition of intelligence which properly encompasses both of these aspects. To do so, we will begin
by giving a rigorous definition to intelligence in the sensory domain, which amounts to a rigorous definition of predictive coding. Next, we will give a rigorous definition to intelligence in the domain of choosing actions, which is essentially a rigorous definition of curiosity. Finally, we will merge these two principles to define intelligence in general.

### 3.2 Definitions

#### 3.2.1 Predictive Coding

**Definitions of Predictive Coding**

While the idea of predictive coding has been around for some time, it appears to still lack a standard definition. Previous predictive coding models have attempted to minimize the following goal within some bandwidth constraint:

\[
\left\langle \left[ S_{t+1} - f(A_t) \right]^2 \right\rangle_t
\]

where \( S_t \) is the sensor input at time \( t \), \( A_t \) is the action the system outputs at time \( t \), \( f \) is a function which reconstructs the next input given the current output. This is to say, specify some way to reconstruct the next input from the current output, and choose the current output to minimize the variance of the reconstruction. This definition assumes
several things. First, it assumes that there exists a subtraction operator defined in the stimulus space. In the case of spike trains, there is no such standard operator. Second, it assumes that the future does not branch much. If the current stimulus could be followed by two very different stimuli, then there will be no way to make the variance small. Third, it assumes that information can always be conveyed in the short term. A simple counterexample to this would be a video stream which was made by interleaving the frames of two different video streams. In this case, no frame contains any information about the next frame. Another, more pertinent example, is the case of spike trains again. Since time is continuous, the time step should be infinitesimal. However, in a given segment of time, there are only a finite number of times when all neurons are not silent. Therefore, the current input conveys no information about the next one.

For all of these reasons, we reject the above formulation of the predictive coding goal. Bialek has argued that predictive coding should be done using Shannon information, although he has, to our knowledge, not specified exactly what this looks like. However, he defines predictive information as

\[ I(S^{<t}; S^\geq t) \]

Unfortunately, this is a curve, as opposed to a single number. Perhaps the obvious way to fix this is to make the goal of predictive coding to maximize

\[ \lim_{T \to \infty} I(A^{\leq T}; S^{> T}) \]

Figure 3.1: A) A four state markov process where the state changes often from top to bottom and back, but stays on the left or right side for a period of time. B) A four state markov process with unknown transition probabilities.

To understand the behavior of this goal, consider the Markov process shown in Figure 3.1A. Suppose that a predictive coding system is designed to maximize the above goal under the bandwidth constraint that it is allowed to output one bit at each time step.
The system could use its bit to indicate whether the stimulus is one of the top two states or one of the bottom two states. However, since this feature changes with probability 0.5 on each time step, it conveys no information about the future states. On the other hand, if the system used its bit to indicate whether the stimulus is one of the left two states or one of the right two states, then it conveys some information about the future. Regardless of how large $T$ is, this will always be the optimal solution.

While the goal works well for the process shown in Figure 3.1A, it works poorly for the process shown in B. This process is again a four state Markov process, but the system does not know at the outset what the transition probabilities are. It must learn them by observing the input. Now, there are two things about the future that can be conveyed: the state of the process, and the transition matrix which governs the process. If the system always conveys information about the current stimulus, then the value of the goal will be constant as $T$ grows to infinity. To see why, suppose that by observation the system learned that the process actually is what is shown in A. As $T$ grows to infinity, the system will learn that the most information it can convey about the state is horizontal position. At time $T$, the system will convey the horizontal position of the state, and none of the previous outputs up to time $T$ are important anymore because the system is Markov. Therefore, the system conveys at most one bit about the future.

On the other hand, suppose that the system conveyed information about the transition probabilities. There are sixteen transition probabilities for a four state Markov process, so the system could partition its output into blocks of sixteen time steps. On each time step, it will convey one bit about one of the transition probabilities. As a simple example of how the system might do this, suppose that at time $t$, the system is supposed to convey a bit about the transition from state 0 to state 1. The system could build a model of the transition matrix give the data it has seen, and then choose a state to follow state 0, according to the probabilities in its internal model. If the new state is a 1, then it outputs a 1, otherwise it outputs a 0. The amount of information that the internal model conveys about the future grows logarithmically with $T$ [Bialek et al., 2001b] [Bialek et al., 2001a], and it is not hard to argue that the amount that the system conveys using the above policy will also grow logarithmically with $T$. So by conveying information about the transition matrix instead of the state, the system’s objective function becomes infinite as $T$ goes to infinity, instead of staying less than 1. This is clearly a better policy.

Unfortunately, such a policy has missed the point of predictive coding. The original idea for predictive coding is that a system picks out the useful features of the data, and conveys these to some part of the brain that takes actions based on them. The policy just described conveys a lot of information about how the input behaves, but never conveys any information about the current state of the input. Since decisions need to be based on current world state, such a system is useless for planning actions.

The problem with the above goal is that it is only concerned with the distant future. We propose a new goal which balances interest in both the present and the future. A
predictive coding system should maximize the following:

$$\sum_{t=0}^{T} I(A^t; S^{>t} | A^{<t})$$

(3.1)

for a possibly infinite horizon $T$. In the infinite horizon case, this objective function will typically become infinite, so some form of normalization may be needed. Under this goal, the optimal strategy for coding input from the model in Figure 3.1A is again to just code whether the state is on the left or on the right. Doing this will cause the objective function to grow linearly with $T$.

In the case where the model is unknown, the optimal policy is more interesting. If the system conveys only information about the transition probabilities, then the total information it can convey by time $T$ is logarithmic in $T$. However, consider a system which allocates some output bits to state and some output bits to transition probabilities. If amount of time between two successive bits allocated to transition probabilities grows exponentially with the number of bits that have been sent, then there is a constant amount of information that can be conveyed by each transition bit. This is an important point which we will demonstrate later on in this chapter.

The amount that it can convey about the future state of the system, as opposed to its transition matrix, is less that one bit per time step on average, since nothing can be conveyed about the vertical position, and the horizontal position only changes infrequently. The amount that can be conveyed about the transition matrix builds with time. The system can therefore beat the strategy of only coding state by instead coding state when there is more information available about state than transition, and coding transition when there is more information about transition than state. Since the average information about the state is constant, it should choose the time between transition bits so that the information it can convey about the transition at each time it sends a transition bit is larger than the amount it could have conveyed about the state at that time. The net result is that this system conveys more information about the future than one which only codes state and than one which only codes transition, and it does so by conveying information about transition exponentially less and less. Similarly, if the input were characterized by a process where the predictive information grew as a power law, then the number of bits used to code the system model would decrease as a power law. Therefore, we argue that this new definition of predictive coding properly balances its resources to convey information both about the current input state and about the statistics of the input.

Before moving on, there is one other definition which is worth mentioning. The definition in Equation 3.1 is specific to temporal codes of the input. At each moment in time, the code is attempting to add as much to what it has already said as possible. However, there may be occasions where it is useful instead to make the output at time $t$ alone convey as much about the future input as possible. In this case, the goal at time
$t$ would be to maximize

$$I(A^t; S^{>t})$$ (3.2)

where $A^t$ is a function purely of $S^t$. This framework seems inappropriate to neural signaling, since for most $t$, no neuron is firing, and therefore $A^t$ conveys nothing about $S^{>t}$. However, this may be useful for a task like finding a good representation of images. Suppose that we consider any image to be one frame in a movie, and we do not have access to the other frames. One could argue that the best representation of a single image is the feature set which is most predictive of the future of the movie. As bandwidth becomes tight, a coding of this nature would toss out pixel noise, then details of object positions for objects that might move frequently, and so forth. These are the same sorts of features which get ignored by Equation 3.1. Thus if we assume that human perception really is based on Equation 3.1, then the optimal coding of an image should be given by Equation 3.2.

**Model Information Equals Future State Information**

There is another formulation of the predictive coding goal which was presented in the previous section. This alternate formulation is equivalent under a broad class of circumstances, and can be easier to work with. It states that a predictive coding agent should maximize the following:

$$\sum_{t=0}^{T} I(A^t; M^t | A^{<t})$$ (3.3)

where $M^t$ is a statistical model of the future input. Before proving this claim and stating the conditions for which it is true, we must spend a moment defining what we mean by a model.

Consider the task of modeling a sequence of bits generated by flipping a biased coin with an unknown bias. One way to model this would be to say that we have a prior distribution of biases for the coin, which we will assume is uniform for this example. Before the first toss, we pick a bias from our distribution of biases, and then we use this bias to generate the result of the toss at each time step. This strategy will allow us to assign a probability to any data sequence.

Another option is more iterative in spirit. We could start with our distribution of biases, pick a bias, and toss the coin based on that bias. Then, we could update our distribution based on the results of the toss, and repeat this process. Equivalently, we could generate each toss based on the expected bias given the results of previous tosses. We will call this the Bayes model. According to it,

$$p(H|N_H, N_T) = \frac{N_H + 1}{N_H + N_T + 2}$$
where $H$ means heads, $N_H$ is the number of heads so far observed, and $N_T$ is the number of tails so far observed. Now we have a single model of the data, which is fully known beforehand. We can again assign a probability to any future sequence of tosses, and the probability we assign using this model will always be identical to the probability we assign by picking the bias before the first toss as previously discussed. However, in the first view, we have a distribution of models, and since each coin toss reduces the entropy of our model, we can talk in a meaningful way about the information that the coin toss conveys about our model. In the second view, we have a single model, and since it never changes no matter what data we see, no coin toss ever conveys information about the model. Despite this difference, any given sequence of coin flips will have the same probability under the distribution of models as it will have under the Bayes model.

The distinguishing factor between these is that if we restarted the experiment a large number of times, then one model from the distribution (the one with the true bias) will faithfully tell the probability of each sequence of coin flips. The Bayes model will not, since the coin would have a different bias each time. Even though the two systems produce identical probabilities for a single set of coin flips, their results differ when the system can be reset. Even though it may not be possible to reset the system, this distinction is important, and from here out, when we talk about a model of the data, we mean a model which would work even if the system were reset.

Having clarified what we mean by “model”, we now prove the claim above under the following condition:

$$\exists f. \quad p(M^t|S^{\geq t}) = \delta [M^t, f(S^{\geq t})]$$

(3.4)

where $M^t$ is a model of the future input, and $\delta$ denotes the dirac delta function. This is to say that at time $t$, the system has seen stimulus $S^{\leq t}$, and $M^t$ carries all the information required to predict the probability of any input $S^{>t}$ from the current moment. Further, if the system were reset back to the current moment an arbitrary number of times, $M^t$ would properly give the distribution of futures from the current moment. This condition intuitively means that the future data is sufficient to determine the model completely. In the case of coin flipping, if the sequence of coin flips is arbitrarily long, then the bias can be determined with arbitrary accuracy. From any point in time, there is an arbitrary amount of data left, so $p(M^t|S^{>t})$ is indeed a dirac delta function. Another way of viewing this condition is that it states that everything learned is useful, in the sense that everything learned is predictive of future input. If instead the sequence of coin flips were cut short, then the model would convey less about the distribution of future input, and what was learned about the model would be less useful. Anything learned on the last coin flip would be useless, because there is no future input left to predict.
Having established the necessary definitions and conditions, the proof is as follows:

\[
I(A^t; S^{\geq t} | A^{< t})
\]

\[
= - \sum_{S^{\geq t}, A^{\leq t}} p(S^{\geq t}, A^{\leq t}) \ln \frac{p(S^{\geq t} | A^{< t})}{p(S^{\geq t} | A^{\leq t})}
\]

\[
= - \sum_{S^{\geq t}, A^{\leq t}} p(S^{\geq t}, A^{\leq t}) \ln \frac{\sum_{M^t} p(S^{\geq t} | M^t) p(M^t | A^{< t})}{\sum_{M^t} p(S^{\geq t} | M^t) p(M^t | A^{\leq t})}
\]

\[
= - \sum_{S^{\geq t}, A^{\leq t}} p(S^{\geq t}, A^{\leq t}) \ln \frac{\sum_{M^t} p(M^t | S^{\geq t}) p(M^t | A^{< t}) / p(M^t)}{\sum_{M^t} p(M^t | S^{\geq t}) p(M^t | A^{\leq t}) / p(M^t)}
\]

\[
= - \sum_{S^{\geq t}, A^{\leq t}} p(S^{\geq t}, A^{\leq t}) \ln \frac{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(M^t | A^{< t}) / p(M^t)}{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(M^t | A^{\leq t}) / p(M^t)}
\]

\[
= - \sum_{S^{\geq t}, A^{\leq t}} p(S^{\geq t}, A^{\leq t}) \ln \frac{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}
\]

\[
= - \sum_{S^{\geq t}, A^{\leq t}, M^t} p(S^{\geq t} | M^t) p(M^t | A^{\leq t}) \ln \frac{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}
\]

\[
= - \sum_{S^{\geq t}, A^{\leq t}, M^t} \frac{\delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}{p(M^t)} p(M^t | A^{\leq t}) \ln \frac{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}
\]

\[
= - \sum_{S^{\geq t}, A^{\leq t}, M^t} \frac{\delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}{p(M^t)} p(M^t | A^{\leq t}) \ln \frac{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}
\]

\[
= - \sum_{S^{\geq t}, A^{\leq t}, M^t} \frac{\delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}{p(M^t)} p(M^t | A^{\leq t}) \ln \frac{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}{\sum_{M^t} \delta[M^t, f(S^{\geq t})] p(S^{\geq t}) / p(S^{\geq t})}
\]

\[
= - \sum_{A^{\leq t}, M^t} p(M^t | A^{\leq t}) \ln \frac{p(M^t | A^{< t})}{p(M^t | A^{\leq t})}
\]

\[
= I(A^t; M^t | A^{< t})
\]

Thus if everything that can be learned is useful, then it doesn’t matter whether the agent uses the amount it has learned about the model or the amount it has learned about the future.

**Information on Input Statistics can be Stored Up**

We now return to the question of whether or not the information a system has learned about the world can be stored up and signalled in compressed format. We present an empirical demonstration of this in practice, using a simple illustrative example. Suppose that the system is observing the results of tosses of a biased coin, with an
unknown bias randomly chosen from a uniform distribution. A simple upper bound on the amount of information the system has at time $t$ about the future is the number of bits required to represent the number of heads observed at time $t$ plus the number of bits required to represent the number of tails at time $t$. Since the observer of the system can keep track of $t$ for himself, the system only needs to signal the number of heads. Thus the information that can be conveyed is bounded by $\lg t$ (we use $\lg$ to denote $\log_2$). Now suppose that the system outputs one bit of information at times $t_n$, where $t_n = \alpha^n$ for some $\alpha > 1$. A reasonable guess for the information the system gains between times $t_n$ and $t_{n-1}$ is $\lg t_n - \lg t_{n-1} = \lg \frac{t_n}{t_{n-1}} = \lg \alpha$, which is constant. Since the system gains a constant amount of information between each bit it outputs, it seems reasonable that it should be able to carefully use that bit to actually signal a constant amount of information.

We tested this conjecture empirically. As stated in the previous section, the bias of a coin becomes known with arbitrary accuracy as the number of coin flips goes to infinity. Therefore, the theorem proved in the previous section is applicable to the case of coin flipping.

We can place a lower bound on the amount of information which can be conveyed about the bias of a coin using an exponentially decaying number of bits. To do so, we chose a particular signalling strategy and measured how much information it conveyed on average. The strategy works by assuming that there is a prior distribution on the coin’s bias, and this distribution is known to both the coding system and the receiver of the code. At $t = 0$, this prior is assumed to be uniform. Whenever the system gets to send a bit, it looks at the fraction of heads that it has received since the last bit it sent. If this fraction is greater than the expected number from the prior distribution, then it signals a 1; otherwise it signals a 0. The prior distribution is then updated based on whether a 1 was sent or a zero. The difference between the model entropy before the bit was sent and the model entropy after the bit was sent is the amount of information the bit carried about the model, which is also the amount of information the bit carried about the future input.

Using $t_n = \alpha^n$ would allow for the system to output multiple bits at $t = 1$ when $\alpha < 2$, which is not allowed. To fix this, the schedule we actually used for the test was $t_0 = 0$, $t_1 = 3$ and $t_n = t_{n-1} + \alpha(t_{n-1} - t_{n-2})$. A bit is not actually sent at $t = 0$, since the system has not yet observed anything. The coin’s bias was binned into 2000 values, $\alpha$ ranged from 1.2 to 2, and $n$ reached a maximum value of 15. For each value of $\alpha$, 20,000 different biased coins were randomly chosen and tried, and the average information per bit is shown in Figure 3.2. The important points from this graph are that the amount of information per bit flattens out to a constant value, and that this value is dependent on $\alpha$. This demonstrates that the system can to some degree store the information it has learned and then convey it in a compressed format to the receiver. We conjecture that this is a universal phenomenon; any system which is observing an input and learning about the future only needs to spend a number of bits commensurate with how much it has learned in order to convey what it has learned to the receiver.
3.2.2 Predictive Curiosity

The goal of predictive coding is to convey as much information about the future input as possible. This requires that the system learn as much about its future input as possible, and then choose its outputs to convey as much of what it has learned as possible. Since predictive coding is concerned only with passive worlds, the problem of learning as much as possible is greatly simplified. However, if the agent’s actions actually affect its input, then it has some control over how much it learns. If we consider the case where the agent can take any actions it likes and also convey everything it learns, then how should the agent act? We propose a novel answer to this question, which we call predictive curiosity.

In contrast to the development of predictive coding, we will develop the notion of curiosity starting with the limited version and then generalize it. The limited version is similar to the version of predictive coding given in Equation 3.3, and is equivalent to the full version under the same condition (Equation 3.4) that the two versions of predictive coding are equivalent.

The limited version of predictive curiosity states that the agent should choose its policy to maximize

\[
I(S^{\leq T}, A^{\leq T}; M) \equiv \sum_{t=0}^{T} I(S^t, A^t; M|S^{<t}, A^{<t})
\]

(3.5)
where $M$ denotes the model of the data, with the same understanding of “model” as was discussed for predictive coding. This goal states that the agent should learn as much about the world model as possible in the time it has. An agent which chooses its actions for this purpose can reasonably be described as curious. One might wonder why the actions are included in this equation. Since the goal is to choose the policy to maximize this objective function, and the policy is known in deciding how to update the model distribution given the input, then why is the action history important? The answer is that if the policy is deterministic, then it doesn’t matter at all. However, if the policy is stochastic, then knowing which actions were taken is crucial to interpreting the input.

As with the limited version of predictive coding, this definition works well so long as all information is useful. In many cases, this is just fine; if the input is an ergodic Markov decision process, then this definition is optimal. However, if the agent can take actions which cause irrevocable results, then this definition will start to break down. For example, suppose the agent has some marginally interesting MDP world it can interact with, and it also has a large set of coins it can flip. These coins can each be flipped exactly once, and they have a bias which makes them come up heads with certainty or tails with certainty. In this case, the agent learns one bit about the world model for each coin it flips, but this information is totally useless to the agent, since it cannot affect anything about the agent’s future. The agent should spend all of its time learning about the MDP, but instead it will spend its time flipping one shot coins.

![Figure 3.3: Different regions of entropy between between the future, the present, and the world model. All of these entropies are assumed to be conditioned on the past input and actions taken already.](image_url)
The general definition fixes this problem, and is perhaps best understood in terms of Figure 3.3. The three circles in this figure represent the entropy of the future input, the present input and actions, and the model of the input, each conditioned on the past input and actions. Since the hope is to learn something about the future input (which is why we call it predictive curiosity), the regions of interest are regions 1, 2, 3, and 4. It is only the input we are concerned with here, and not the actions, because the agent should not be choosing its actions to maximize information about future actions. The present moment contains no information about regions 1 and 4, so only regions 2 and 3 are of interest. Predictive coding trades the information about these two regions, but curiosity is not interested in what is about to happen so much as how the world behaves in general. The intuitive difference between areas 2 and 3 can be seen in the case of a Markov process. Area 2 is information about the state, and area 3 is information about the state transition matrix. If curiosity were about area 2, then the system would be happy to watch a completely known Markov process. However, a curious agent would get bored once it knows everything that can be known. Therefore, the predictive curiosity goal is to maximize the entropy in area 3. This area represents the amount of information the agent gets at each time step that tells about how the world will behave in the future. This goal can be written in many ways, including each of the following:

\[
\sum_{t=0}^{T} H(S^{>t}|S^{<t}, A^{<t}) + H(S^t, A^t|S^{<t}, A^{<t}) + H(M|S^{<t}, A^{<t})
- H(S^{>t}, S^t, A^t|S^{<t}, A^{<t}) - H(S^{>t}, M|S^{<t}, A^{<t})
- H(S^t, A^t, M|S^{<t}, A^{<t}) + H(S^{>t}, S^t, A^t, M|S^{<t}, A^{<t})
\]  

(3.6)

\[
\sum_{t=0}^{T} I(S^t, A^t; S^{>t}|S^{<t}, A^{<t}) - I(S^t, A^t; S^{>t}|S^{<t}, A^{<t}, M)
\]  

(3.7)

\[
\sum_{t=0}^{T} I(M; S^{>t}|S^{<t}, A^{<t}) - I(M; S^{>t}|S^{<t}, A^{<t})
\]  

(3.8)

\[
\sum_{t=0}^{T} I(M; S^{>t}, S^t, A^t|S^{<t}, A^{<t})
\]  

(3.9)

Equation 3.6 shows the relationship between this generalized form of mutual information and the intersection of sets. Equation 3.7 is interesting because the first term is much like the predictive coding goal, making the relationship between curiosity and predictive coding more explicit. Equation 3.8 is interesting because it clarifies the relationship between this general definition of curiosity and the restricted form given in Equation 3.5. Under the condition given in Equation 3.4, the information between the
future and the model is just the entropy of the model. Equation 3.8 then becomes

$$\sum_{t=0}^{T} H(M|S^{<t}, A^{<t}) - H(M|S^{\leq t}, A^{\leq t}) = \sum_{t=0}^{T} I(M; S^t, A^t|S^{<t}, A^{<t})$$

which is Equation 3.5. The last equation is just a more convenient notation.

The unrestricted version presented here seems to have a bias towards deterministic policies. To see this, suppose that the agent has two biased coins it can flip, where the biases are unknown. On each time step, it chooses a coin to flip, flips that coin, and gets back as input either “heads” or “tails”. The optimal policy is for the agent just to alternate between flipping one coin and then the other. If it does this, then all information is useful, and the amount it learns according to the unrestricted equation will equal the amount it learns according to the restricted equation. However, rather than strictly alternating, it could choose at each time step to randomly decide which coin to flip. In doing so, it will end up learning nearly the same amount about the biases in a given period of time. However, the unrestricted version will measure much less information learned than the restricted version. We consider this a weakness of the unrestricted version at present, and will in the future study using

$$\sum_{t=0}^{T} I(S^{>t}, A^{>t}; S^t, A^t; M|S^{<t}, A^{<t})$$

This is equivalent to the restricted version for a slightly weaker definition of useful:

$$\exists f. \ p(M|S^{>t}, A^{>t}) = \delta [M, f(S^{>t}, A^{>t})]$$

In the case of the stochastic coin flipping policy just mentioned, all information is useful according to this definition, but not the original one. We suspect this definition of curiosity is superior to the one given above, but have not studied it as much, and have not considered how to merge it with predictive coding.

The generalized multivariable form of mutual information used here was first proposed by McGill [McGill, 1954]. It has the curious property that it can be negative. Bell provides a simple and somewhat instructive example of this [Bell, 2003]. Consider three random one bit variables, $A$, $B$, and $C$. $A$ and $B$ are chosen randomly, but $C$ is the exclusive or of the other two. In this case, $I(A; B) = 0$, but $I(A; B|C) = 1$, so the generalized information is -1 bits. A slightly less instructive case can occur for the predictive curiosity. Suppose that the world generates four bits, one at a time. There are two possible world models. In both, three of the four bits are random. In the first model, the third bit is a copy of the first, and in the second model, the fourth bit is a copy of the first. The predictive curiosity principle gives a value of about -0.4 for the first time step for this setup. This deserves more study; at present we do not know the import of the objective function being negative.
3.2.3 Predictive Action

The uniformity of the cortex suggests that there may be one single principle which is ubiquitous throughout. Having defined a principle for perception and a principle for action, we now define a synthesis of these two principles, which we call predictive action. If the world is passive, then a system which implements predictive action should follow exactly the predictive coding policy. If the system has sufficient output bandwidth, then it should follow exactly the predictive curiosity policy.

To accomplish this, consider again the formulas for predictive coding (Equation 3.1) and predictive curiosity (Equation 3.7), shown below.

\[
\sum_{t=0}^{T} I(A^t; S^{>t}|A^{<t}) - I(S^t, A^t; S^{>t}|S^{<t}, A^{<t}, M)
\]

These two principles can be combined to form:

\[
\sum_{t=0}^{T} I(A^t; S^{>t}|A^{<t}) - I(S^t, A^t; S^{>t}|S^{<t}, A^{<t}, M)
\] (3.10)

which is the predictive action goal. In the case where all information is useful, this can be equivalently expressed as:

\[
\sum_{t=0}^{T} I(A^t; M^t|A^{<t}) - I(S^t, A^t; M^t|S^{<t}, A^{<t}, M)
\]

If the world is passive, then the optimal policy for predictive action is indeed the predictive coding policy. This is because in the case of a passive world, the network has no control over the second term \(I(S^t, A^t; S^{>t}|S^{<t}, A^{<t}, M)\). No matter what policy it chooses, this information will be the same, since its actions are ignored by the world. To maximize the predictive action goal, then, the network only has to maximize the first term, which is the predictive coding goal.

If the network has sufficient output bandwidth, then the optimal policy for predictive action is indeed the predictive curiosity policy. To see this, note first that if the network has outputs which do not affect its input, then the use of them is governed purely by the drive to maximize the predictive coding goal. Thus, if the network has some outputs which cause actions and some which do not, then the network will use the ones which do not for predictive coding. If these outputs have enough bandwidth, then they will
code everything about the future that can be coded, which implies that

\[ I(A^t; S^{>t}|A^{<t}) = I(S^t, A^t; S^{>t}|S^{<t}, A^{<t}) \]

In this case, the objective function for predictive action will equal the one for predictive curiosity, and the optimal policies for these two goals will be the same.

Like predictive curiosity, the predictive action objective function can be negative. This will happen, for example, when the network has no output bandwidth (and the world is passive by construction). In this case, the predictive coding term will be zero, but the correction term will not be. Even if we restrict ourselves to worlds where the predictive curiosity goal is always positive, it is not surprising that the predictive action goal can be negative. The original statement of the problem was that predictive action should behave like predictive curiosity when there was sufficient bandwidth. The goal we propose actually matches the objective functions under this circumstance. Having chosen to do so, the function must decrease as the output bandwidth becomes more limited. If it did not, then the network should not use all of its bandwidth, which would violate the predictive coding goal. However, if less bandwidth means that the objective function falls below the objective function for predictive curiosity, then the objective function will be negative whenever there is nothing left to learn about the world and there is any constraint on the bandwidth.

Having proposed an objective function which has the desired properties, we will now speculate for a moment on what the behavior would be of a neural network which implemented this principle. Suppose, for simplicity, that we considered a purely feedforward network, which had raw sensory input coming in at the bottom, and muscles connected to the neurons at the top. While this is not the case in the brain, it is illustrative of the properties of predictive action. At the bottom of the network, there is a great deal of data coming in, which tends to make bandwidth tight. Further, the output of the bottom layers are somewhat removed from the actions taken by the agent. Therefore, the bottom layers should perform predictive coding. This is in keeping with the agreement of experimental data in V1 with the predictions of models which are based on predictive coding.

The middle layers of the network lose information which is not used for taking actions. If in the input, symbol A is quickly followed by symbol B, the the lower layers will keep representing A because it predicts B, but stop representing B because it predicts nothing. In higher layers, the input is just A, which predicts nothing, so it also is dropped. However, if the layer signals A and this is used for making decisions, then it will affect future input. A is then an action that can be taken in an active world, and should not be dropped. Thus the middle layers fade from a representation of the world to a set of commands to take.

Because the middle layers of the network have dropped all of the information which will not be used in making decisions, the top layers of the network do not have as much data coming in. As a result, they have all the bandwidth they need, so they implement
predictive curiosity. As a result, the agent chooses its actions to learn about the world. In this fashion, the principle of predictive action can be used both to design intelligent agents and to explain receptive field properties throughout all of the cortex. As the principle is novel, much remains to be done to determine whether this it can live up to its promise.

3.3 Relationships to Other Models

3.3.1 Temporal Invariance

Introduction

Temporal invariance has been a candidate explanation for how the brain learns to recognize important things in its input for many years now [Mitchison, 1991] [Stone, 1996]. Efficient coding can be traced back to Barlow’s work on frogs [Barlow, 1953], and predictive coding can be traced back to Atal’s work on speech signals [Atal and Schroeder, 1967]. However, despite the general notion that predictive coding and temporal invariance might have some vague commonality at heart, no formal relationship between them has been established. In this paper, we will establish such a relationship between the temporal invariance principle as used in the slow feature analysis algorithm [Berkes and Wiskott, 2002] and static predictive coding, as defined in Equation 3.2.

World Model

Predictive coding and temporal invariance learn the same code for a particular class of input closely related to Brownian motion. Brownian motion is a well known process which generates data in a Markov fashion by adding at each time step Gaussian noise to the previous value. While this is a fair model of many different kinds of data, it poses difficulties for our problem because it does not have a finite variance. If the first sample is zero, then the variance from zero of the \( n^{th} \) sample is proportional to \( n \), so Brownian motion does not converge to a well defined stationary distribution. To resolve this, we will instead use damped Brownian motion (DBM). DBM generates data using the following rule:

\[
x_{t+1} = \gamma x_t + \epsilon n_t
\]

where \( 0 < \gamma < 1 \) determines how damped the system is, \( \epsilon \) controls the temporal variance, and \( n_t \) is zero mean unit variance Gaussian noise. With properly chosen \( \gamma \) and \( \epsilon \), this process has a Gaussian stationary distribution with any desired temporal variance. For a desired temporal variance of \( \sigma^2 \) and zero mean and unit variance for the stationary distribution, \( \gamma \) and \( \epsilon \) should be chosen as:
\[
\gamma = 1 - \frac{\sigma^2}{2}
\]
\[
\epsilon = \sigma \sqrt{1 - \frac{\sigma^2}{4}}
\]

Proof of this is given in Appendix A.1.

Suppose that data is generated using a static invertible nonlinearity \( \tilde{g} \) of a set of independent damped Brownian motion processes \( \tilde{x} \), so that \( \tilde{f}(\tilde{g}(\tilde{x})) = \tilde{x} \), where \( \tilde{f} = \tilde{g}^{-1} \). We use the word “static” here to mean that \( \tilde{g} \) is a function only of the current sample, and not all previous samples. Unlike standard Brownian motion, the sum of two DBM processes is no longer DBM if they have different temporal variances (the proof is in Appendix A.2). For this reason it seems very likely that any invertible nonlinearity is learnable from the data.

We argue in the following sections that static predictive coding and temporal invariance coincide when the input is an invertible nonlinearity of DBM processes. The optimal \( k \) dimensional code for \( \tilde{g}(\tilde{x}_t) \) is proportional to \( f_{1..k}(\tilde{g}(\tilde{x})) \), where \( x_1 \) has lower temporal variance than \( x_2 \) and so forth.

**Static Predictive Coding**

Let us now develop a predictive coding algorithm for coding static images. Our criterion for such an algorithm is that it should optimize Equation 3.2, which is

\[
I(A^t, S^{>t})
\]

where \( A^t \) is a function only of \( S^t \). Since we will be dealing with two inputs, we will switch our notation, so that \( S^t \) is now represented by \( x_t \) and \( y_t \). Also, we will let the output \( A^t \) be represented by \( z_t \).

**Bandwidth Constraints**

The kind of inputs and outputs used for predictive coding is fairly arbitrary, but for the purpose of comparing predictive coding and temporal invariance, let us assume the input at time \( t \) is a vector and the output at time \( t \) is also a vector. The form of bandwidth constraint used is also fairly arbitrary, but since temporal invariance fixes the variance of each of its outputs to 1, we will use this for the bandwidth constraint for predictive coding as well.

As an aside, it is interesting to compare this framework to that of sparse coding [Olshausen and Field., 1996]. In both frameworks, a continuous vector is coded by a continuous vector, and a constraint is put on the output bandwidth. However, whereas
the interesting properties which are learned in sparse coding come from the fact that the output is given a sparse prior, the interesting properties learned from static predictive coding come from the constraint that the output should be predictive of future input. There is no constraint that the output should be sparse, and in fact the constraint of unity variance implies that the optimal use of bandwidth is to have a Gaussian distribution, which is clearly not sparse. Sparsity thus takes the role of a trick used to approximate finding predictive features of the input without actually using time series data.

**Optimal Static Predictive Code**

Since DBM is Markov and the nonlinearity applied to it is static and invertible, the input to our coding system is also Markov. This fact allows us to use a simpler version of Equation 3.2. Since all of the information that \( z_t \) conveys about the future is summed up in what it conveys about \( x_{t+1} \), our objective function can be rewritten as

\[
I(z_t; x_{t+1})
\]

Suppose that the input to the system is two dimensional data generated by two independent DBM processes \( u \) and \( v \) and transformed by some invertible nonlinearity \( g \) into inputs \( x \) and \( y \). The system must apply some nonlinearity to its input in order to maximize \( I(z_t; x_{t+1}, y_{t+1}) \), where \( z_t \) is the system’s noisy one dimensional output. If \( z_t = f(x_t, y_t) + \sigma n_t \), where \( \sigma \) is the standard deviation of the noise added to \( z_t \), and \( n_t \) is zero mean unit variance Gaussian noise, then we argue that the optimal function \( f \) is \( f(x_t, y_t) = a g_{1}^{-1}(x_t, y_t) = au_t \), where \( a = \sqrt{1 - \sigma^2} \) and \( u \) is assumed to have a lower temporal variance than \( v \). In this case, the output of the system is again DBM. Proof of this is equivalent to proving that the optimal code of a two dimensional DBM system is a scaled value of the slower component. Appendix A.3 contains a proof that this coding function is a stationary point for the static predictive coding objective function under the unit variance constraint. This is to say that if the input can be transformed back to the original DBM process, then we can consider this a preprocessing step, and determine what the optimal code is of \( u \) and \( v \). If the code used is \( au_t \), then no small change in this code will change the value of the objective function. Therefore, this code is either a locally best code, worst code, or an inflection point of some sort.

We argue without proof that it is a maximum. To see why, consider Figure 3.4, which shows a two dimensional DBM process whose temporal variance in the horizontal direction is greater than that of the vertical direction. At time \( t \), the input is the black dot. Part A) shows the case where \( z_t = y_t \) (we will consider \( \sigma = 0 \) for this illustration), and the distribution of \( x_{t+1}, y_{t+1} \) is shown as the box in B). On the other hand, part C) shows the case where \( z_t = x_t \), and the distribution of \( x_{t+1}, y_{t+1} \) is shown as the box in D). Notice that the box in B is much smaller than the box in D, because the dot doesn’t move vertically very fast. The information conveyed about the future is the entropy of the future before knowing \( z_t \) minus the entropy of the future after knowing \( z_t \). Since
Figure 3.4: A two dimensional DBM process, where the horizontal temporal variance is greater than the vertical temporal variance. A) A code which picks out the vertical position of the current input. B) The range of possible next inputs, given the code in A. C) A code which picks out the horizontal position of the input. D) The range of possible next inputs given the code in C.

the coding function can’t affect the first entropy, the goal of the coding function is to minimize the second entropy, which is graphically shown as the size of the box. Since the horizontal line box is smaller than the vertical line box, $z_t$ should code $y_t$.

To extend this argument a bit further, suppose that $z_t = f(x_t, y_t)$. In this case, any given $z_t$ specifies a curve or curve segment instead of a horizontal or vertical line. Any section of the curve which has a nonzero slope will clearly contribute to the box being larger than if it had zero slope. Therefore, the optimal curve is the horizontal line, and the best code is $z_t = ax_t$.

Temporal Invariance

Suppose that instead of maximizing $I(z_t; x_{t+1}, y_{t+1})$, we minimize the following:

$$\langle (z_{t+1} - z_t)^2 \rangle_t$$

under the constraints that $z_t$ has zero mean and unit variance. We call this function the temporal variance of the process, and argue that the optimal behavior of a system minimizing temporal variance is again to recover the slower component of the original DBM. The argument begins with the theorem proven in Appendix A.4, which states that the function $f(x_t, y_t) = ag_1^{-1}(x_t, y_t) = au_t$ is a stationary point for the temporal invariance objective function as well. We argue without proof that this function is a minimum. Appealing again to Figure 3.4 A) and B), the current input falls on the horizontal line $z_t$, and the code for the next input falls on the horizontal line $z_{t+1}$.
expected distance between these lines is the temporal variance of the code. Clearly, the expected difference is smaller for the code in A) and B) than it is for the code in C) and D).

As in the predictive coding case, if \( z_t = f(x_t, y_t) \), then a given \( z_t \) specifies a curve segment, and any point on the curve with nonzero slope will widen the box, causing the temporal variance to increase. So once again, the optimal code is the horizontal line, where \( z_t = ay_t \).

Although the above argument is fairly convincing, there is a function which still might be worth considering for the temporal invariance case. Suppose that instead of using \( z_t = ay_t \), the system used \( z_t = H(y_t) \), where \( H \) is the threshold function. For this function, the temporal variance of the input would be zero almost all the time, but occasionally would be very large. On the graph, this corresponds not to a curve, but to two sections. The top half of the graph corresponds to \( z_t = 1 \), and the bottom half of the graph corresponds to \( z_t = -1 \). This function still causes \( z_t \) to have zero mean and unit variance in noiseless case (it can be scaled to account for noise in order to make the variance unity).

The results of applying a threshold to DBM are shown in figure 3.5. The graph displays the temporal variance of the thresholded DBM process as a function of the temporal variance of the original DBM process. Since the maximum temporal variance that a unit variance DBM process can have is \( \sqrt{2} \), the graph ends there. The dashed line represents the temporal variance of the original process, or equivalently, the process under the identity transformation.

![Figure 3.5](image)

Figure 3.5: Temporal variance of the thresholded DBM is always greater than the temporal variance of the original DBM, except at the extremes, which are both meaningless. The dashed line represents the variance of the DBM under the identity transformation.
The key observation is that the thresholded DBM has a greater temporal variance than the normal DBM except when the temporal variance is 0 or $\sqrt{2}$. The temporal variance is only zero as a limiting case; if it were truly zero then the system would have a constant value. Since the system is constrained to have zero mean, this constant must be zero, but then the system does not have unit variance. So we can ignore the zero case, as it cannot happen. They are also equal when the temporal variance is $\sqrt{2}$. In this case, the input is not truly DBM, since $\gamma = 0$. The system is just Gaussian noise. A predictive coding system would fail on this as well, since the information between successive samples will be 0 regardless of the transformation used.

In summary, if the input to the system is DBM transformed by an invertible nonlinearity, then the optimal predictive coding algorithm should learn exactly the same function as the optimal temporal invariance algorithm. Both should invert the nonlinearity and return the slower of the two DBM processes, scaled to account for output noise. If the input to the system can only be transformed to something which is “close” in some sense to DBM, then both predictive coding and temporal invariance would try to get close. They may differ on exactly what transform is closer, but they would at least be shooting for the same goal. Thus temporal invariance is equivalent to predictive coding if the input can be transformed to DBM, and it approximates predictive coding if the input can be transformed to something which is approximately DBM.

**Generalization to More Output Dimensions**

So far, we have considered a system with two inputs and one output. The generalization to more inputs is straightforward, but the generalization to more outputs is more interesting. In the case of predictive coding, the optimal use of output bandwidth is to make the outputs statistically independent. If each output recovers a different DBM process, then this will be true, and the coding network will optimally use its output bandwidth. Likewise for temporal invariance, the outputs are constrained to be uncorrelated, which they will be if each output recovers a different DBM process. We prove in Appendix A.5 that the predictive coding objective function will be larger for smaller temporal variance DBMs, and the temporal variance is trivially smaller for smaller temporal variance DBMs. So it seems likely that both predictive coding and temporal invariance will use separate outputs for each DBM, and if they do, they will both pick out the set of DBMs for which the temporal variance is the smallest.

**Conclusions**

We have argued that if the world is characterized by a DBM process and the sensor input is an invertible nonlinear transformation of it, then the optimal code for the input is to learn the inverse nonlinearity and thus recover the original DBM process. This result is independent of whether “optimal” is defined by static predictive coding or by
temporal invariance. This allows temporal invariance to be seen as an approximation of predictive coding, and this approximation becomes exact in the case where the input is an invertible static transformation of damped Brownian motion.

3.3.2 Bayesian Surprise

Bayesian surprise is a principle due to Itti and Baldi [Itti and Baldi, 2004] [Itti and Baldi, 2005c] which is designed to quantify how surprising certain data is. They use this to model attention in primates, arguing that attention is devoted to data which is surprising. For a given piece of data $x$, the amount of surprise due to $x$ is defined as

$$S(x) = D_{p(M|x)\|p(M)} = \int_M p(M|x) \ln \frac{p(M|x)}{p(M)} dM$$

where $M$ is a model of the input, and $D_{\|}$ denotes the Kullback-Liebler divergence. A piece of data is therefore surprising if it changes the agent’s belief about the world.

Bayesian surprise is very similar to predictive curiosity. To see this, note that the expected surprise of a given piece of unknown data is given by

$$\int p(x)S(x)dx = \int p(x) \int p(M|x) \ln \frac{p(M|x)}{p(x)} dM dx$$

$$= \int \int p(M, x) \ln \frac{p(M|x)}{p(x)} dM dx$$

$$= I(x; M)$$

where $I(x; M)$ is the mutual information between the data and the model. Recall that the restricted version of the predictive curiosity goal is to maximize

$$I(S^{\leq t}, A^{\leq T}; M)$$

The difference between Bayesian surprise and the restricted version of predictive curiosity is that $x$ is only specified as data received by the model. While the agent may be able to attend to different things, it cannot affect the world around it. In the domain Itti and Baldi have focussed on, the agent can make saccades to various locations in a video, but it cannot change what is happening in the video. Because of this, they simply use $x$ in the equation, and do not make a distinction between whether $x$ is just information about $S$ or includes information about $A$. For their purposes, $x$ could be viewed as the input from the eye and the proprioceptive input of where the eye is. In contrast, predictive curiosity is specifically designed for active worlds, where the agent is not just observing the world, but is actively seeking out its most interesting aspects. Thus the main difference between Bayesian surprise and the restricted version of predictive curiosity is a question of domain. Of course, the unrestricted definition of pre-
predictive curiosity is significantly different from Bayesian surprise, both mathematically and conceptually.

3.3.3 Intelligent Adaptive Curiosity

Oudeyer et al. proposed a model of curiosity called Intelligent Adaptive Curiosity (IAC) [Oudeyer and Kaplan, 2004] [Oudeyer et al., 2005]. IAC is specified as an algorithm used to learn about the world. This algorithm is embodied in an Aibo robot which is placed in a playground environment and learns to interact with the toys there.

The IAC algorithm assumes that the world is Markov and, for simplicity, that the world is sufficiently memoryless that planning is not necessary. At each moment in time, the Aibo has a percept $S^t$ and takes an action $A^t$, and this results in a new percept $S^{t+1}$. IAC remembers all tuples of \{$S^t, A^t, S^{t+1}, t$\}. These tuples are each put into a bag of past experiences, and whenever there is sufficient reason, a bag is split into two bags. Each bag has not only a set of experiences, but an expert which tries to predict from $S^t$ and $A^t$ what $S^{t+1}$ will be. For each prediction, there is a variance between the prediction of $S^{t+1}$ and the true $S^{t+1}$. If the expert is actually learning, this variance should decrease as the expert gains experience. Each bag also contains a model of how the expert’s variance has been changing with time. When the Aibo has to choose an action, it finds the appropriate expert for each possible action it can take, and chooses the action whose expert’s variance has been decreasing the fastest. In this way, the Aibo chooses the action it thinks it can learn the most from.

The IAC algorithm is an approximate implementation of predictive curiosity for a special case. Suppose that the conditional probability of $S^{t+1}_i$ is given by

$$p(S^{t+1}_i | S^t, A^t) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(s^{t+1}_i - \mu(s^t, A^t))^2}{2\sigma(s^t, A^t)^2}}$$

where $\sigma(s^t, A^t)$ is the amount of noise present in the input, and $\mu(s^t, A^t)$ determines the mean of next input. Suppose further that $\mu$ and $\sigma$ are piecewise constant, and that the IAC algorithm has already split the bags properly. In this case, the entropy of the world model is just the sum of the entropy of $\mu$ and $\sigma$ for each regime.

Since the regime is determined by \{ $S^t, A^t$ \}, the agent’s choice of action partially determines which regime its next data sample comes from on each time step. The predictive curiosity goal states that the agent should choose its action so that for the chosen regime, the information

$$I(\mu, \sigma; x^n | x^{<n})$$

is greater than for any other regime it could choose, where $n$ is the number of samples seen in this regime so far, including the current one, and each $x$ is one sample.
from the regime. This entropy can be rewritten as

\[ I(\mu, \sigma; x^n | x^{<n}) = H(\mu, \sigma | x^{<n}) - H(\mu, \sigma | x^{<n}) \]

The entropy \( H(\mu, \sigma | x^{<n}) \) for a particular data sample \( x^{<n} \) has only an additive dependence on \( n \). To see this, note first that it can be rewritten as

\[ H(\mu, \sigma | x^{<n}) = H(\mu | \sigma, x^{<n}) + H(\sigma | x^{<n}) \]

The first term is independent of the true value of the population mean \( \mu \). If we add some constant \( c \) to each of the samples of \( x^{<n} \), then the distribution of \( \mu \) simply shifts up by \( c \), and the entropy doesn’t change. Also, its dependence on \( \sigma \) is additive. If we multiply \( \sigma \) and each of the points \( x^{<n} \) by \( c \), then the entropy will increase by \( \ln c \), regardless of \( n \). Since the entropy of \( \mu \) does not depend on \( \mu \) and depends only additively on \( \sigma \), it can be written as

\[ H(\mu | \sigma, x^{<n}) = f(n) + \ln \sigma \]

The same argument applies to the second term, so it can be rewritten as

\[ H(\sigma | x^{<n}) = g(n) + \ln \sigma \]

Thus the full entropy is given by

\[ H(\mu, \sigma | x^{<n}) = h(n) + \ln \sigma \]

The information in question is then

\[ I(\mu, \sigma; x^n | x^{<n}) = h(n) - h(n - 1) \]

which is only dependent on \( n \). As Bialek points out in [Bialek et al., 2001a], the information \( I(\mu, \sigma; x^{<t}) \) must be sublinear in \( n \) (actually, logarithmic in our case), which means that the information above must decrease with \( n \). Therefore, the optimal predictive action agent should just choose its action to make the regime it gets data for be the regime for which it has received the least data.

The IAC algorithm is concerned with the variance of \( x^n - \bar{\mu} \), where \( \bar{\mu} \) is the sample mean for the regime, not including the current sample \( x^n \). This variance is equal to the variance of \( x - \mu \) plus the variance of \( \bar{\mu} - \mu \), since \( x^n - \mu \) and \( \mu - \bar{\mu} \) are independent. The variance of \( x^n - \mu \) is \( \sigma^2 \), and the variance of \( \bar{\mu} - \mu \) is \( \sigma^2 / n \), so the total variance is \( \sigma^2 (1 + \frac{1}{n}) \). This decreases monotonically with \( n \), but is proportional to \( \sigma \). Like predictive curiosity, the IAC algorithm thus chooses the regime for which it has seen the least data, but unlike predictive curiosity, it weights the amount of data it has by the
variance of the regime. This unnecessarily biases IAC to explore regimes which have less variance.

The notion of exploring regimes evenly seems at first glance to make the agent act in a very boring fashion. It should spend as much time staring at blank walls as it should spend playing with toys. However, this is not the case, because the regimes are not distributed evenly. In order to model the data well, there only needs to be one blank wall regime, but many, many toy regimes. As a result, it will spend vastly more time playing with toys than it will staring at the wall.

### 3.3.4 Reinforcement Learning

According to the reinforcement paradigm, the agent should choose its actions to maximize

$$\sum_{t=0}^{T} r_t$$

where $r_t$ is the reward it gets at time $t$. The predictive action principle states that the agent should maximize

$$\sum_{t=0}^{T} I(A_t; S^{>t} | A^{<t}) - I(S^t, A^t; S^{>t} | S^{<t}, A^{<t}, M)$$

These can be trivially combined to form

$$\sum_{t=0}^{T} r_t + I(A_t; S^{>t} | A^{<t}) - I(S^t, A^t; S^{>t} | S^{<t}, A^{<t}, M)$$

Intuitively, this implies that the agent is rewarded by gathering information about its environment, and by representing information about its future, but it is also rewarded by the external reward signal $r_t$. This gives a proper unit to reward signals: bits. A given reward $r_t$ of 5 bits is therefore equivalent to learning 5 bits about the environment in the predictive curiosity regime, and equivalent to representing 5 bits about the future input in the predictive coding regime.

A different view of the relationship between reinforcement learning comes from the fact that a curious agent has implicitly a way to be trained: it can be trained through boredom. This is possibility is investigated in Chapter 5. The results of training using boredom are not quite the same as the standard reinforcement learning results, in that an agent will not asymptotically approach the desired behavior. Nevertheless, it demonstrates the implicit reinforcement learning like behavior of predictive curiosity without any modifications at all.
3.4 Conclusions

In this chapter we have presented the predictive action objective function and related it to other models. We have shown that under the proper circumstances, it can act like predictive coding, predictive curiosity, temporal invariance, Bayesian surprise, intelligent adaptive curiosity, and can be trivially extended to include reinforcement learning. In the next two chapters, we will explore specific aspects of it with concrete examples to get a better feel for its behavior.
4 Coding Models

4.1 Introduction

This chapter describes three networks for doing sensory coding. The first model processes only static images, and is presented only as a stepping stone to the second model which actually processes video. This model is an efficient coding model which uses deterministic neurons. The last model is a predictive coding model which uses Poisson neurons.

4.2 Matching Pursuit based models

There is a longstanding suggestion that neurons use firing rates to convey continuous values over time. This requires that the number of spikes a neuron sends within the time window for which its input is relatively stationary must be high enough for the receiver to measure its average firing rate with reasonable accuracy. If a neuron only gets to fire one or two spikes before its input changes, then this method will not work.

Several recent studies (e.g. Thorpe et al [Fabre-Thorpe et al., 2001], Antal et al [Antal et al., 2000], Johnson and Olshausen [Johnson and Olshausen, 2003]) suggest that a measure of recognition is achieved in the visual system on a time scale that allows few neurons time even to fire once. In concert with these studies, Thorpe, Delorme, and VanRullen [Thorpe et al., 2001] argue that only one percent of the neurons need to fire a single time in order to convey an image with reasonable fidelity, assuming that the timing of the spikes is allowed to convey information. Further, Meister et al [Keat et al., 2001] argue that precise spike timing is possible, and can therefore be used to convey information.

The sparse coding principle has been put forward in some ways to be a surrogate for minimizing the number of spikes while still being able to work with continuous valued functions. Of course, real neurons output spikes. To this end, Ballard, Zhang, and Rao [Ballard et al., 2001] suggested a model for sparse coding that assumes spikes
come in volleys, and codes information based on the time offset between a spike and a volley. However, volleys are highly correlated spike trains, and if one is attempting to minimize the number of spikes, then spikes should be independent. This inconsistency is accounted for through the hypothesis that the brain is multitasking and doing many things at once, and this conjunction of processes makes the spikes appear independent. We seek a simpler explanation.

A different class of spike based models is based on an algorithm called matching pursuit. Smith demonstrated very efficient encoding of audio using matching pursuit [Smith and Lewicki, 2005] and Perrinet et al. proposed a model of sparse coding to learn codes for static images and a model to learn codes for movies [Perrinet et al., 2003] [Perrinet and Samuelides, 2001]. Likewise, we propose both a static model and a dynamic model, using essentially the same network architecture as Perrinet but different learning rules. Perrinet’s model uses spikes as input, but requires volleys and does not develop temporal receptive fields. At the cost of using grayscale input, our network is able to avoid using volleys and develop temporal receptive fields.

4.2.1 Matching Pursuit

We now introduce the Matching Pursuit algorithm, which is at the core of this network. Consider the problem of representing samples from a dataset $\Omega$ of $N$ dimensional vectors. Suppose that each sample is to be represented as a weighted sum of $M$ unit basis vectors $b_1, b_2, ... b_m$, so sample $\tilde{s} \in \Omega$ is represented as:

$$\tilde{s} = \sum_{i=1}^{M} a_i \hat{b}_i$$

where $a_i$ is the amount of basis vector $\hat{b}_i$ to add in to the reconstruction of $\tilde{s}$. We use the hat to indicate that a particular vector (e.g. $\hat{b}$) is a unit vector, whereas $\vec{b}$ would not necessarily be one. If $M = N$, this problem can be solved with matrix inversion. If $M > N$, then the problem is underconstrained. Lewicki argues in [Lewicki and Sejnowski, 2000] that this is commonly a better model of natural data than the complete case. However, finding a unique representation of $\tilde{s}$ requires some other constraint. In the context of this paper that constraint is that the $a$ values should be sparse. Here, sparse is only roughly defined, but it means that only a few of the $a_i$ values are large, and most are near zero. Mallat and Zhang [Mallat and Zhang, 1993] designed an algorithm called Matching Pursuit (MP) to code $\tilde{s}$ using a sparse $\vec{a}$.

MP is a greedy iterative algorithm, where one basis vector is added in to the reconstruction at each step. MP starts by defining a residual $\tilde{s}^k$, which is the part of the input not coded for after $k$ steps. At startup, $\tilde{s}^0 = \tilde{s}$, and $\vec{a} = \vec{0}$. Then, at each step, the index
\( n_k \) of the next basis vector chosen is

\[
    n_k = \text{argmax}_i |\hat{s}^{k-1} \cdot \hat{b}_i| \tag{4.2}
\]

where the vertical bars denote absolute value. Equation 4.2 corresponds to choosing the basis vector that can most decrease the residual error. The corresponding coefficient of the chosen basis vector is then set to

\[
    a_{n_k} = \hat{s}^{k-1} \cdot \hat{b}_{n_k} \tag{4.3}
\]

and the new residual becomes

\[
    \hat{s}^k = \hat{s}^{k-1} - a_{n_k} \hat{b}_{n_k} \tag{4.4}
\]

This process is illustrated in Figure 4.1.

---

Figure 4.1: MP takes two iterations to encode the target 'x' using two basis vectors, shown as dashed lines. Left, the lower basis vector passes closest to the 'x', so it is chosen and the optimal amount of it is added to the reconstruction. Right, the optimal amount of the remaining vector is used. The arrow points to the final result, but some error still remains.

For reasons discussed in the next section, our algorithm does not have the luxury of choosing the optimal basis vector at every step. Fortunately, Mallat and Zhang provide a modification to the algorithm to deal with this difficulty. They suggest that if there is a means of choosing a basis vector such that

\[
    |\hat{s}^{k-1} \cdot \hat{b}_{n_k}| \geq \gamma \text{argmax}_i |\hat{s}^{k-1} \cdot \hat{b}_i| \tag{4.5}
\]

then this can be used instead, although the \( a \) values will be less sparse as \( \gamma \) decreases, where \( 0 < \gamma \leq 1 \) is a parameter that controls how close a vector needs to be to optimal before it can win a bid (with 1 meaning it has to be optimal).
4.2.2 The Static Model

Both of the matching pursuit models we discuss here have the same architecture, shown in Figure 4.2A, although the image presentation time does not figure in to the model for video.

In the static model, each unit has a basis vector associated with it, and the coefficient for that unit’s basis vector is the reciprocal of the delay between the time that the image was presented to the system and the time that the unit fired. When a unit fires, a new iteration of MP begins. To implement MP with local units requires that each unit keep track of a single scalar (analogous to a membrane potential), \( v^k_i \), which is the potential of unit \( i \) at the end of iteration \( k \). Using this, the MP equations can be written in the following form:

\[
v^0_i = \frac{\vec{b}_i \cdot \vec{s}}{|\vec{b}_i|^2} \quad (4.6)
\]

\[
n_k = \arg\max_i v^{k-1}_i \quad (4.7)
\]

\[
a_{n_k} = v^{k-1}_i \quad (4.8)
\]

\[
v^k_i = v^{k-1}_i - \frac{a_{n_k} (\vec{b}_i \cdot \vec{b}_{n_k})}{|\vec{b}_i|^2} \quad (4.9)
\]

The argmax is automatic because the units signal \( v^k_i \) by firing at some time, and the unit that fires first wins the argmax. Notice that these equations imply that

\[
v^k_i = \frac{\vec{b}_i \cdot \vec{s}^k}{|\vec{b}_i|^2} \quad (4.10)
\]

Figure 4.2: A) Network architecture. B) Spike generation in temporal model
which means that Equation 4.7 does not perfectly match Equation 4.2 if $\vec{b}_i$ is not a unit vector for all units. However, it can be matched up with Equation 4.5 using $
abla = |\vec{b}_{\text{min}}|/|\vec{b}_{\text{max}}|$ where $\vec{b}_{\text{min}}$ and $\vec{b}_{\text{max}}$ are the basis vectors with the minimum and maximum magnitudes, respectively. In practice, this seems to be around 0.5.

The above equations allow the units to run MP such that each unit only needs to know its own basis vector, the correlation of its basis vector with those of the other units, the input, and its potential at any time. However, the model still does not contain any way to adapt the basis vectors. The simplest answer to this is to just use the delta rule:

$$\vec{b}_i^k = \vec{b}_i^{k-1} + \eta a_i^k (\vec{s} - \vec{b}_i^{k-1})$$  \hspace{1cm} (4.11)

where $\eta$ controls the learning rate of the network, and this equation is only used by the unit that is firing in that iteration. This equation allows other units to keep track of their correlation to the unit that just fired, since

$$\vec{b}_j \cdot \vec{b}_i^k = \vec{b}_j \cdot \vec{b}_i^{k-1} + \eta a_i^k (\vec{s} - \vec{b}_j \cdot \vec{b}_i^{k-1})$$  \hspace{1cm} (4.12)

However, the unit that just fired has no way to know what its update did to its correlations with the basis vectors of the other units. If these correlations are not updated, the network tends to become unstable. The cleanest mathematical solution to this is to note that if unit $i$ fired, then unit $j$ will properly update its correlation to unit $i$. Therefore, after firing, each unit should just read back its correlation from the other units right after firing.

While simple, this operation lacks a clear analog in biology. A more biologically plausible solution is to ensure that no unit can excite another unit by firing. This is done by modifying Equation 4.11 to

$$v_i^k = v_i^{k-1} - Q \left[ a_{nk} \left( \vec{b}_i \cdot \vec{b}_{nk} \right) \right]$$  \hspace{1cm} (4.13)

where $Q$ is a step function. This solution is somewhat intuitive, since basis vectors that negatively correlate should probably not both be strongly used in reconstructing the same input. However, there is no guarantee that the optimal way to reconstruct the input does not use such basis vectors. Fortunately, this does solve the network stability problem, since it ensures that no unit can have a greater potential than it did at the beginning of the process.

### 4.2.3 Results for Static Images

We tested the model on whitened natural images of forests. The whitening code used was from Olshausen’s web site, and was the same whitening code used in [Olshausen, 1996]. Random 12x12 image patches were taken from the images and fed to
4.2.4 The Temporal Model

The temporal model is very similar to the static model and shares the same network architecture. One major difference is that the basis vectors are time dependent, so they can be seen as videos instead of images. In Equation 4.9, when a unit fired, it caused a drop in the potential of the other units. In the temporal model, this drop becomes dependent on the amount of time that has elapsed since the unit fired, as the correlation between two videos depends on the time delay between them.

There are two major difficulties unique to the temporal model. The first is that in the static model, a unit signaled how well it correlated with the input by waiting for a certain amount of time before firing. In the temporal model, not only is there no obvious time marker to base a delay off of (without assuming synchronous firing), but the time of a spike must already signal when the receptive field should be added into the reconstruction of the input. The second major difficulty is that basis vectors become very blurry video if delta rule learning is used with threshold crossing units. This is because the threshold is always crossed before the potential reaches its peak. In the
simple case where the input video perfectly matches the basis video, this means that the basis video will be modified to be more like a delayed version of itself (since it fired early), causing blurring.

These difficulties can be overcome with several adjustments to the model. The first is to make the unit wait for peaks in the input, and fire after they occur. If the delta rule is used on the input at the time of the peak, then blurring is avoided. The delay between the peak and firing can be chosen to be inversely proportional to the height of the peak. This allows the units to compete for which one will fire first. Figure 4.2B shows this process (ignore the spike amplitude temporarily).

In order to compete properly, the peaks from which units base their delays must have some measure of correspondence. To achieve this, let the temporal center of mass of unit $i$’s video be defined as:

$$m_i = \frac{\sum_{t=1}^{T} t\bar{b}_it \cdot \bar{b}_it}{\sum_{t=1}^{T} \bar{b}_it \cdot \bar{b}_it}$$ (4.14)

where $T$ is the number of frames in the unit’s video, and $b_{it}$ is the $t^{th}$ frame of unit $i$’s video. To make the peaks comparable, each unit shifts its video in time so that $m_i = \frac{T}{2}$, filling the new frames with zeros. This ensures that two units with similar videos peak at approximately the same time on a particular input, and the delay from their peaks can become the determining factor in which unit spikes first.

Finally, in order to reconstruct the input it is necessary to know how tall the peak was and where the peak was. Where the peak was can be determined from the timing of the spike and the height of the peak, so only the height of the peak needs to be conveyed. There are two possibilities for this. The first is that the unit fires multiple times, and the number of times it fires conveys the height of the peak. Thus if one spike is assumed to convey $\bar{b}(t)/2$, then the unit can convey the height of the peak quantized to $\bar{b}(t)/2$. The second, simpler strategy assumes that the unit can signal an amplitude when it spikes, which means it only needs to spike once per peak. Dunlap has recently made the suggestion [Park and Dunlap, 1998] that there may indeed be a way that biological neurons could convey an analog value with each spike, although this is far from well accepted. Analogously, the units in the temporal model are allowed to signal an analog value with each spike in the interest of simplicity of the model. The entire model for spiking time and height is shown in Figure 4.2B.

In the static model, if a unit was not allowed to read back another unit’s estimate of their correlation, then the network was unstable so long as a unit’s firing could excite its neighbor. In the temporal case, this limitation is necessary for stability as well, and it becomes even more necessary because the estimate even with read back is inaccurate. This is due to the way units shift their video segments in time to keep the centroid in Equation 4.14 in the right place. Even with read back, this temporal shift is not taken into account, so the network becomes unstable if units are allowed to excite each other.
4.2.5 Results on Temporal Data

Test data was generated from static images using the same technique as Wiskott and Sejnowski [Wiskott and Sejnowski, 2002]. A window is translated, rotated, and scaled through time over a static image, and the input to the network is what is under the window at each time step. The various transformations of the window are all performed simultaneously; on every step the window is translated, rotated, and scaled.

Given this input, the network learned a set of temporal receptive fields that again look like spatially local, oriented Gabor filters. However, these filters are a function of time. Most are a combination of translated and rotated Gabor filters. Some are almost purely one or the other, while very few appear to represent pure scaling.

One frame of several of the resulting video segments is shown in Figure 4.3B. Note that at any given time frame, the filters look like spatially oriented Gabor filters. Figure 4.3C shows a single unit’s video as it evolves through time. This particular unit appears to be looking for a diagonal edge moving from the top right to the bottom left. Figure 4.3D shows the temporal course of the reconstruction error plotted against the average number of spikes per frame. The sequence starts on the right, before the network was significantly trained. Over the course of 30,000 frames, the error remained essentially constant, but the number of spikes per frame decreased by a factor of about six, showing that the network does indeed minimize the number of spikes used to represent the input.

4.3 Predictive Coding Network

While predictive coding is theoretically a very nice idea, it has not enjoyed great success in the neural coding community because of the difficulty in learning good non-linear predictive codes. We are of the opinion that this difficulty has been enhanced by a strong tendency of researchers to equate the goal of conveying information about the future with the goal of creating an estimate of the future inputs with low variance, as the network in the previous section attempts to do. The first goal states that the Shannon information should be maximized, and the second goal states that the Fisher information should be maximized. Shannon information is more general than Fisher information, in that high Fisher information implies high Shannon information, but high Shannon information does not imply high Fisher information. As a simple example of this, consider a sequence of images. There are a very large number of possible images, but suppose that given a set of images up to the current moment, it is possible to deduce that the next image will either be all white on the left half and all black on the right or vice versa, with equal probability. In this case, the Shannon information is very high, since the a priori very large number of possibilities has been cut down to only two. However, the Fisher information is very small, because the expected value of the next image is just gray, and the variance between the expected image and the real image is going to be very high. Since it seems plausible that knowledge such as
this could be useful to an organism, we choose to cast predictive coding in terms of Shannon information.

Systems which maximize Fisher information need to propose a method of reconstructing the input from the output in order to define the Fisher information. A consequence of using Shannon information is that reconstruction of the input from the system’s output may no longer be possible. In the above example, if later outputs of the system do not clarify which of the two images actually was seen, then no accurate reconstruction of the input will be possible. While this may seem intuitively to be a drawback, we argue that this liberates the system of the burden of having to worry about reconstruction. The system we propose does not reconstruct the input, nor can it do so. It can be difficult to propose a feasible means for a network of spiking neurons to reconstruct the input it has received from other spiking neurons, and our network sidesteps this difficulty.

4.3.1 Algorithm Derivation

This algorithm was designed with predictive action in mind, but the definition of predictive action used by this algorithm is obsolete. The definition used was that if the network follows some policy, then there is an information between the output of the network and the future input. Since the network can affect not only its own outputs, but also its inputs in the general case, the network must be careful to properly measure the information it conveys. While this definition of predictive action was deemed unsatisfactory, the network is still a fine predictive coding network. Further, if there is any reason to revisit the above definition, this network may again serve to help explore the properties of the definition.

This algorithm is designed as a two layer network, where layer $S$ represents the sensor input and layer $A$ represents the action taken by the network (the output spike train). Layer $A$ consists of spiking neurons, and layer $S$ can be either spiking neurons or analog inputs. Since $A$ can take input from spiking neurons, more layers can be added to the network, but they do not affect the derivation.

Let us define some notation. First, capital letters for layers indicate the random variables, and lowercase letters indicate particular values. For layer $X \in \{S, A\}$, $X^t$ represents the outputs of all neurons in $X$ at time $t$ and $X^{<t}$ represents the collective outputs of the neurons prior to $t$ ($X^{\leq t}$, $X^{\geq t}$, and $X^{>t}$ are defined analogously). $X^t_i$ is the output of neuron $i$ at time $t$. Time is assumed to pass in infinitesimal but synchronous $^1$ steps of size $dt$, so $X^t_i \in \{0, 1\}$.

$w_k$ represents an unspecified weight value in neuron $k$. There are three functions, $f_i$, $g_i$, and $h_i$, which will be discussed at length later on. Each of these functions

---

$^1$See Appendix B.1 for a rigorous definition of what we mean by “synchronous”
implicitly takes a set of weights, and the subscript for the function denotes the neuron whose weights are being used. These functions are assumed to have finite ranges and to change continuously with their weights. When $f$ and $h$ are used without subscripts or arguments, they denote values in the ranges of $f$ and $h$, and are used as summation variables. $t$ and $\tau$ represent time, with $d\tau = dt$. They are discrete when summed over and continuous when integrated over, and $x_i^t$ denotes the value of neuron $i$ in layer $x$ at time step $t$ when $t$ is discrete and at time $t$ when $t$ is continuous.

Maximizing Information

In order to maximize the information between the input and the output of the network, we need to define information carefully. The difficulty comes from the fact that the network sees one long stream of data and generates one long stream of output. Formally, the information is zero because there is only one possible input stream. Two things are needed to fix this. First, the input stream is broken up into an infinite number of overlapping fixed size chunks (one which starts on each time step except for the last few). Second, the network’s output must be probabilistic, so that any input can generate any output. Given all of this, the information the network conveys for a single chunk of time can be written as

$$I(S^{\geq t}; A^t|A^<t)$$

where $t$ is summed over the chunk length. The information inside this sum can be rewritten as:

$$I(S^{\geq t}; A^t|A^<t) = H(A^t|A^<t) - H(A^t|S^{\geq t}, A^<t)$$

$$+ H[A^t|\tilde{f}(S^{\geq t}, A^<t), \tilde{h}(A^<t)] - H[A^t|\tilde{f}(S^{\geq t}, A^<t), \tilde{h}(A^<t)]$$

$$+ H[A^t|\tilde{h}(A^<t)] - H[A^t|\tilde{h}(A^<t)]$$

$$= H[A^t|\tilde{h}(A^<t)] - H[A^t|\tilde{h}(A^<t)]$$

$$+ H[A^t|\tilde{f}(S^{\geq t}, A^<t), \tilde{h}(A^<t)] - H[A^t|S^{\geq t}, A^<t, \tilde{f}(S^{\geq t}, A^<t), \tilde{h}(A^<t)]$$

$$- \left\{ H[A^t|\tilde{h}(A^<t)] - H[A^t|A^<t, \tilde{h}(A^<t)] \right\}$$

$$= I[A^t; \tilde{f}(S^{\geq t}, A^<t)|\tilde{h}(A^<t)]$$

$$+ I[A^t; S^{\geq t}, A^<t|\tilde{f}(S^{\geq t}, A^<t), \tilde{h}(A^<t)] - I[A^t; A^<t|\tilde{h}(A^<t)]$$

$$\approx I[A^t; \tilde{f}(S^{\geq t}, A^<t)|\tilde{h}(A^<t)]$$

$$+ I[A^t; S^{\geq t}, A^<t|\tilde{f}(S^{\geq t}, A^<t)] - I[A^t; A^<t|\tilde{h}(A^<t)]$$

where $\tilde{f}$ and $\tilde{h}$ can be any arbitrary function, up until the last line, where we assumed that $p(a^t|\tilde{f}, \tilde{h}) = p(a^t|\tilde{f})$. This is to say that $\tilde{f}$ preserves all of the information between
and $a^t$ that $\tilde{h}$ does. Notice that in the middle equation, several equivalencies such as this $H[A^t|A^{<t}] = H[A^t|A^{<t}, \tilde{h}(A^{<t})]$ were used. These are true because $\tilde{h}(A^{<t})$ provides no information about anything that is not already contained in $A^{<t}$.

The first term of the new expression can be viewed as an approximation to the original information, with the latter two being error terms in the approximation. To see how to use this, let $p(a_i^t = 1|s^{<t}, a^{<t}) = g_i(s^{<t}, a^{<t}) dt$. This function is computed by each neuron in the network at each time step to determine the probability of that neuron firing at that time step. Notice that if $g$ is a function only of $s^{<t}$ and not of $a^{<t}$, then the neuron is a standard Poisson neuron.

Since $\tilde{g}$ determines the behavior of the network, it determines $I(S^{\geq t}; A^t|A^{<t})$, and therefore the value of the sum of the three new terms, regardless of what $f$ and $\tilde{h}$ are. As we will show, it is possible to derive a gradient ascent rule which allows us to find the $\tilde{g}$ which maximizes $I[A^t; \tilde{f}(S^{\geq t}, A^{<t})|\tilde{h}(A^{<t})]$. If we can choose $\tilde{f}$ and $\tilde{h}$ to minimize the other two terms, then the first term should be close to the real information, and maximizing it should approximately maximize the real information. It may seem unintuitive to minimize the last term, since it positively contributes to the information. However, we are choosing $f$ and $\tilde{h}$ to do this, and the true information is independent of $f$ and $\tilde{h}$. Therefore, minimizing the last term only shuffles its information into the first term, but does not actually change the information.

In order to shuffle information away from the first error term, notice that if $p(a_i^t = 1|s^{\geq t}, a^{<t}) = f_i(s^{\geq t}, a^{<t}) dt$, then this term would be zero, because the most that can be said about $a^t$ is its probability, and $f$ already says that, so $a^{<t}$ has nothing left to say. Therefore, this term can be minimized by approximating the above relationship. To this end, we make $f$ a linear estimate of probability of the neuron spiking at time $t$ given $s^{\geq t}$ and $a^{<t}$. Although a nonlinear estimate could conceivably provide more information, the linear estimate seems to be satisfactory so far. This same strategy is used for the second term as well.

Before launching into the derivation, it is useful to note that there is only one independence assumption that can be made. One can assume that for a given time step, the probability of an output neuron firing is independent of anything else that happens at that time step given all of the past history. Thus the firing of two output neurons for a given time step are independent, and the firing of an output neuron on a given time step is independent of anything that happens to the input on that time step.

To derive the gradient ascent algorithm, we will take the gradient of the approximation of equation 4.15 with respect to a weight $w_k$ in $g$. Since weights in $f$ and $h$ are adapted separately as described above, we will ignore them in this section of the derivation. Also, for ease of reading, only the broad overview of the derivation is presented in the main text, with a number of details handled in the appendices. Whenever a jump is made from one line to the next where the intervening steps are in the appendices, a footnote mark appears on the equals sign of the second line, and the footnote tells where to find the intervening steps.
\[
\frac{\partial}{\partial w_k} \sum_t I[\tilde{f}(S^{\geq t}, A^{< t}); A^t|\tilde{h}(A^{< t})] \\
= \frac{\partial}{\partial w_k} \sum_t \sum_{\tilde{h}} p(\tilde{h}) \sum_a p(a^t, \tilde{f}, \tilde{h}) \ln \frac{p(a^t|\tilde{f}, \tilde{h})}{p(a^t|\tilde{h})} \\
= \sum_{t, \tilde{h}, \tilde{f}, a^t} \left\{ \ln \frac{p(a^t|\tilde{f}, \tilde{h})}{p(a^t|\tilde{h})} \frac{\partial}{\partial w_k} p(a^t, \tilde{f}, \tilde{h}) \right. \\
+ p(a^t, \tilde{f}, \tilde{h}) \left[ \frac{\partial}{\partial w_k} p(a^t|\tilde{f}, \tilde{h}) - \frac{\partial}{\partial w_k} p(a^t|\tilde{h}) \right] \right\} \\
= \sum_{t, \tilde{h}, \tilde{f}, a^t} \left\{ \ln \frac{p(a^t|\tilde{f}, \tilde{h})}{p(a^t|\tilde{h})} \frac{\partial}{\partial w_k} p(a^t, \tilde{f}, \tilde{h}) \right. \\
+ p(\tilde{f}, \tilde{h}) \frac{\partial p(a^t|\tilde{f}, \tilde{h})}{\partial w_k} - p(\tilde{f}|a^t, \tilde{h}) p(\tilde{h}) \frac{\partial p(a^t|\tilde{h})}{\partial w_k} \right\} \\
= 2 \sum_{t, \tilde{h}, \tilde{f}, a^t} \ln \frac{p(a^t|\tilde{f}, \tilde{h})}{p(a^t|\tilde{h})} \frac{\partial}{\partial w_k} p(\tilde{f}|\tilde{h}, s, a)p(\tilde{h}|s, a)p(s, a) \\
\approx 3 \sum_{t, \tilde{h}, \tilde{f}, s, a} \ln \frac{p(a^t|\tilde{f})}{p(a^t|\tilde{h})} \frac{\partial}{\partial w_k} \delta[\tilde{f}, \tilde{f}(S^{\geq t}, A^{< t})] \delta[\tilde{h}, \tilde{h}(A^{< t})] p(s, a) \\
= \sum_{t, s, a} \left[ \sum_{\tilde{f}, \tilde{h}} \delta[\tilde{f}, \tilde{f}(S^{\geq t}, A^{< t})] \delta[\tilde{h}, \tilde{h}(A^{< t})] \ln \frac{p(a^t|\tilde{f})}{p(a^t|\tilde{h})} \right] \frac{\partial}{\partial w_k} p(s, a) \\
= \sum_{t, s, a} \ln \frac{p(a^t|\tilde{f}(S^{\geq t}, A^{< t})}{p(a^t|\tilde{h}(A^{< t})} \frac{\partial}{\partial w_k} p(s, a) \\
= 4 \sum_{t, s, a, i} \ln \frac{p(a^t|\tilde{f}(S^{\geq t}, A^{< t})}{p(a^t|\tilde{h}(A^{< t})} \frac{\partial}{\partial w_k} p(s, a) \\
= 5 \sum_{t, s, a, i} \ln \frac{p(a^t|\tilde{f}(S^{\geq t}, A^{< t})}{p(a^t|\tilde{h}(A^{< t})} p(s, a) \sum_{\tau} \frac{\partial}{\partial w_k} p(a^\tau_k|s^{< \tau}, A^{< \tau}) p(a^\tau_k|A^{< \tau}, s^{< \tau})
\[ p(s, a) = \sum_{i, s, a} p(s, a) \left[ \sum_{\tau} \frac{\partial}{\partial w_k} p(a_k^\tau = 1 | s^{<\tau}, a^{<\tau}) \right. \]
\[ \left. \left( 1 - a_k^\tau \right) \frac{\partial}{\partial w_k} p(a_k^\tau = 1 | s^{<\tau}, a^{<\tau}) \right] \]
\[ \cdot \sum_i \left\{ a_i^t \ln \frac{p[a_i^t = 1 | \tilde{f}(s^{\geq t}, a^{<t})]}{p[a_i^t = 1 | \tilde{h}(a^{<t})]} \right. \]
\[ \left. + (1 - a_i^t) \left\{ p[a_i^t = 1 | \tilde{h}(a^{<t})] - p[a_i^t = 1 | \tilde{f}(s^{\geq t}, a^{<t})] \right\} \right\} \]

\( p(s, a) \) can be sampled from the empirical distribution, and the three remaining probabilities in this expression are defined by functions that return their firing rate as follows:

\[ p(a_i^t = 1 | a^{<t}, s^{<t}) = g_i(s^{<t}, a^{<t}) dt \]
\[ p[a_i^t = 1 | \tilde{f}(s^{\geq t}, a^{<t})] \approx f_i(s^{\geq t}, a^{<t}) dt \]
\[ p[a_i^t = 1 | \tilde{h}(a^{<t})] \approx h_i(a^{<t}) dt \]

The second relation here is because \( f \) is trained to estimate \( p[a_i^t = 1 | s^{\geq t}, a^{<t}] \). If it is a good estimate, then the relation is true. The third relation follows the same reasoning.

Substituting these probabilities in the previous expression yields:

\[ \sum_{t, s, a} p(s, a) \left[ \sum_{t_b^{<\tau} \leq t \leq t_e} \frac{\partial}{\partial w_k} g_k(s^{<\tau}, a^{<\tau}) - \int_{t_b}^{t_e} \frac{\partial}{\partial w_k} g_k(s^{<\tau}, a^{<\tau}) d\tau \right] \]
\[ \cdot \sum_i \left\{ a_i^t \ln \frac{f_i(s^{\geq t}, a^{<t})}{h_i(a^{<t})} + (1 - a_i^t) \left\{ h_i(a^{<t}) - f_i(s^{\geq t}, a^{<t}) \right\} \right\} dt \]

Here \( t_b \) and \( t_e \) are the beginning and ending time of the current time chunk, and as such, they are functions of \( t \). At this point, it is time to turn our attention to how to actually implement such a system. In biology, each dendrite needs to keep track of the total of the sum and integral over \( f_i \) and \( h_i \) need to be adapted by means which will be discussed in detail later. The sums over \( i \) are a single global sum over all the neurons, and could conceivably be handled as a single chemical concentration,

\[ \text{See Appendix B.2.1} \]
\[ \text{The approximation here is that } p(a^t | \tilde{f}, \tilde{h}) \approx p(a^t | \tilde{f}). \text{ This will be exact if } \tilde{f}(s^{\geq t}, a^{<t}) \text{ contains all the information about } a^{<t} \text{ that } \tilde{h}(a^{<t}) \text{ does.} \]
\[ \text{See Appendix B.2.2} \]
\[ \text{See Appendix B.2.3} \]
where the neurons add or take out chemical as needed. The biggest hurdle for biology comes from the fact that \( f_i \) cannot be calculated in real time, since it requires knowing the future input. Therefore, each neuron has to delay calculating \( f_i \) until it knows what happens next, and this in turn means that the chemical concentration needs to be delayed, and the whole learning process needs to be delayed. While possible, this is fairly messy.

On a computer, these present no difficulty. However, care must be taken in order to make the simulation run efficiently. To make a simulation run efficiently, we will assume that there are periods of time where \( g_i, f_i, \) and \( h_i \) do not change. In particular, they change only when the input changes or when an output neuron spikes. This allows the simulation to handle all the intervening time as a single simulation step.

To handle a single simulation step, let \( t_0 \) be the start time of the step and \( t_f \) be the end time. The step will range from time \( t_0 < t \leq t_f \), and in the interval \( (t_0, t_f) \), nothing happens. At time \( t_f \), the input may change (if the input is continuous) or a spike may occur (in either layer, if the input is not continuous). Further, the time chunk start and end times \( t_b \) and \( t_e \) are assumed to stay the same during the time interval \( (t_0, t_f] \). This gives us the following relations:

\[
\begin{align*}
g_i(s^<, a^<) &= g_i(s^{<t_0}, a^{<t_0}) \\
h_i(a^<) &= h_i(a^{<t_0}) \\
f_i(s^{>}) &= f_i(s^{>t_f})
\end{align*}
\]

At every time \( t \), some amount is contributed to the total derivative of the information with respect to a given weight. In the interval of time \( (t_0, t_f] \), this amount is given by:

\[
\frac{\partial}{\partial w_k} \sum_{t_0 < t \leq t_f} I[\tilde{f}(S^{>t}); A^t|\tilde{h}(A^{<t})]dt
\]

\[
= \sum_{t, s, a} p(s, a) \left[ \sum_{t_0 < \tau \leq t_e} \frac{\partial}{\partial w_k} g_k(s^{<\tau}, a^{<\tau}) - \int_{t_0}^{t_e} \frac{\partial}{\partial w_k} g_k(s^{<\tau}, a^{<\tau}) d\tau \right]
\]

\[
\cdot \sum_i \left\{ a_i^{t_f} \ln \frac{f_i(s^{>t_f}, a^{<t_0})}{h_i(a^{<t_0})} + \int_{t_0}^{t_f} \left\{ h_i(a^{<t_0}) - f_i(s^{>t_f}, a^{<t_0}) \right\} dt \right\}
\]

\[
= \sum_{t, s, a} p(s, a) L_{k}^{t_f}
\]

\[
\cdot \sum_i \left\{ a_i^{t_f} \ln \frac{f_i(s^{>t_f}, a^{<t_0})}{h_i(a^{<t_0})} + (t_f - t_0) \left\{ h_i(a^{<t_0}) - f_i(s^{>t_f}, a^{<t_0}) \right\} dt \right\}
\]
This expression, while messy, is basically ready to go into a program to simulate the network as soon as \( g, f, \) and \( h \) are defined. These will be discussed later. The two sums over \( i \) are global across neurons, and roughly correspond to the hypothesized chemical concentration mentioned before. There are two of them here instead of one chemical concentration because the one has two different types of behavior. It changes smoothly when nothing is going on, and it changes in spurts when spikes occur. Because of the way the simulation is broken up, these two types of changes have to be handled separately, necessitating two global sums. Incidentally, it is possible to do it with only one sum if the system is not doing active coding.

**Training \( f \) and \( h \)**

\( f \) and \( h \) can be trained by making them an optimal estimate of the probability of seeing a spike at time \( t \) given the available information. This requires care, as \( f \) must be trained off of the real probability of spiking given \( s^{\geq t} \) and \( a^{<t} \), as opposed to trying to match \( g \) as closely as possible. This is because the future may actually be more informative about the current probability of spiking than the probability that the spike was generated from. \( h \) could be trained off of \( g \) or from the real probability, but since we already need to work out the learning rule for \( f \), we may as well use it for \( h \) also.

For simplicity, we will make \( f \) and \( h \) linear estimates of the spike probability. For the moment, we will call our estimated probability \( p \).

\[
J = \sum_t [p - a^t_k]^2
\]

The derivative of this is given by

\[
\frac{\partial J}{\partial w_k} = \sum_t 2[p - a^t_k] \frac{\partial p}{\partial w_k}
\]

The gradient descent learning rule is then

\[
\Delta w_k = -\eta (p - a^t_k) \frac{\partial p}{\partial w_k} = -\eta \frac{\partial p}{\partial w_k} + \eta a^t_i \frac{\partial p}{\partial w_k}
\]

where \( \eta \) is indexed by \( k \) because the learning rates for the output spike history and the input spike future could be different. Of course, \( p \) should be infinitesimal. Therefore,

\footnote{See Appendix B.2.4}
let \( p = r dt \), where \( r \) is the actual function we are training. This gives us

\[
\Delta w_k = -\eta \kappa r dt \frac{\partial r}{\partial w_k} dt + \eta \kappa a_k \frac{\partial r}{\partial w_k} dt
\]

\[
\frac{\partial w_k}{\partial t} = -(\eta \kappa dt) r \frac{\partial r}{\partial w_k} + (\eta \kappa dt) a_k \frac{\partial r}{\partial w_k}
\]

\[
= -\epsilon \kappa r \frac{\partial r}{\partial w_k} + \epsilon \kappa a_k \frac{\partial r}{\partial w_k}
\]

where \( \epsilon \kappa \) is now our learning rate. If we assume the form of \( r \) to be

\[
r = \sum_i w_i y_i
\]

where \( y_i \) are the inputs to the function, then the above equation can be solved in two cases. When there is no spike present, the weights obey the following differential equation, assuming that the inputs do not change when no output spike occurs (like in the previous section):

\[
\frac{\partial w_k(t)}{\partial t} = -\epsilon \kappa r(t) y_k
\]

The solution to this equation for the period of time from \( t_0 \) to \( t_f \) is given by

\[
w_k(t) = w_k(t_0) + \epsilon \kappa r(t_0) y_k \left( e^{-\left( t - t_0 \right)} \sum \epsilon_i y_i^2 - 1 \right)
\]

When a spike occurs, each weight gets \( \epsilon \kappa \frac{\partial w}{\partial w_k} = \epsilon \kappa y_k \) added to it, so the total change in weight from \( t_0 \) to \( t_f \) is given by

\[
w_k(t_f) = w_k(t_0) + \epsilon \kappa r(t_0) y_k \left( e^{-\left( t_f - t_0 \right)} \sum \epsilon_i y_i^2 - 1 \right) + \epsilon \kappa a_k y_k
\]

**The Choice of \( g \)**

The above learning algorithm is general and can be applied to a wide variety of different types of neurons, with the type of neuron specified by the form of the functions \( g, f, \) and \( h \). \( g \) is the most interesting of these, since it specifies how the neuron translates inputs to output spikes, whereas \( f \) and \( h \) are related only to learning.

We have begun small by choosing a simple threshold Poisson neuron. Weights \( w_k^g \) are used in the following manner:

\[
g(s < t, a < t) = \epsilon + T \left[ \sum_i w_i^g s_i^g \right]
\]
where \( T(x) = 0 \) if \( x < 0 \) and \( T(x) = x \) if \( x > 0 \). \( \epsilon \) is a small spontaneous firing rate. It is needed because \( g \) appears in the denominator of the learning rule, and therefore should not go to zero. Notice that \( g \) does not depend on \( a < t \) for this neuron. This is partly for simplicity, as it would open the door for the network to run in an unstable fashion, and partly because there is some reason to think that biological neurons are Poisson without feedback.

For the derivative of \( g \), we use

\[
\frac{\partial}{\partial w_{i,t}} g(s < t, a < t) = s_i^t Q \left[ \sum_{i,t} w_i^t s_i^t \right]
\]

where \( Q(x) = 0 \) if \( x < 0 \) and \( Q(x) = 1 \) if \( x \geq 0 \). The original function is not smooth when the argument of \( T \) is zero, so the derivative is undefined. We don’t have a clean answer for what to do about this, and it may be taken as an argument that we would be better using squaring Poisson neurons instead of threshold linear ones, since the derivative at 0 would then be 0. At present, we are ignoring this difficulty.

**Minimizing Spike Count**

Left on its own, the learning rule above will result in the network using more and more spikes, because higher spike rates can convey more information. However, we wish to maximize information while minimizing the number of spikes. To do this, our total objective function is given by

\[
I[f(S^{\geq t}, A^{< t}); A^{t} | h(A^{< t})] - \gamma \sum_a p(a) \sum_{i,t} a_i^t
\]

The second term is the expected number of spikes in each time chunk, and \( \gamma \) trades off the amount of information the network conveys against how many spikes it uses. Maximizing the objective function implies minimizing the second term, which is done as follows:

\[
\frac{\partial}{\partial w_k} \sum_a p(a) \sum_{i,t} a_i^t = \sum_t \sum_{s,a} p(s,a) L_k^t \sum_i a_i^t
\]
As a learning rule, this gives

\[ \frac{\partial}{\partial w_k} \sum_{t_0 < t \leq t_f} \sum_a p(a) \sum_i a_i^t \]

\[ = L_k^t \sum_i a_i^{t_f} \]

Therefore, subtracting some amount of \( L_k^t \) from each weight whenever any neuron spikes decreases the spike rate.

### 4.3.2 Conclusions

We have described a mechanism for training a predictive coding network that is fairly biologically plausible, using nearly local spiking neurons. In contrast to previous networks, this network does not use volleys, and it explicitly attempts to maximize the predictive coding objective function defined in Equation 3.1. Applying it to static images and video cause it to develop static Gabor filters and temporally whitened Gabor filters. Figure 4.4A shows for one neuron both the function of the past which it used and the function of the future it used. A spike from this neuron can be interpreted as saying that the pattern it has seen indicates that the pattern it predicts is coming. Figure 4.4B shows the functions of all neurons for the time right before a spike, and figure 4.4C shows the predictions of all neurons for right after the neuron spikes. The strong similarity of the input before a spike and the input after a spike is due to the temporal stability of the input. For inputs which changed in predictable ways, this similarity may no longer hold.
Figure 4.4: A) Timecourse for one neuron. The four frames on the left are the filter the neuron uses on the past to determine whether it is going to fire. The four frames on the right are the filter the neuron uses to predict the future. B) The most recent history frame for all neurons. C) The prediction frame for the next input.
5 Predictive Curiosity Models

Predictive curiosity can in theory be applied to any task an agent may face. However, in practice no tractable algorithm has been found to carry it out, and this limits the scope of problems that it can be tried on. The next few sections show some examples of cases where it can be tried, and the results of these cases.

5.1 Recognizing a Picture

One domain which we have tried predictive curiosity on is the task of recognizing which image from a dataset the agent is looking at. This task only uses the Bayesian surprise aspects of predictive curiosity, and is vaguely reminiscent in spirit to the work independently done by Itti and Baldi on modeling human saccades [Itti and Baldi, 2005a] [Itti and Baldi, 2005b] [Itti and Baldi, 2006]. Suppose that an image is selected from a known set of images. An agent looks at the image, and is supposed to determine which of the images in the dataset it is looking at. If the agent’s eyes are perfect, then this task is easy. If the agent’s eyes have a sharp fovea, weak peripheral vision, and noise from all of its photodetectors, then the problem gets harder. The agent must choose the action to take (the saccade target) which will allow it to predict as much as possible about the future state of the world (which picture from the dataset it is looking at).

To make things concrete, suppose that there are \( N \) images in the dataset and \( S \) points in the image which the agent can make a saccade to. Regardless of where the agent’s eye is pointed, it receives data from each of the \( S \) points. However, if the saccade point is far from a given point, the values from that point will be attenuated more before adding noise. Thus the agent gets the most information from the saccade point, and progressively less from points which are farther away from it. For simplicity, there is only one filter at each saccade point.

Let \( f_{is} \) denote the output of the filter at saccade point \( s \) for input image \( i \). This value can be modeled as the true filter value attenuated by some factor, plus some amount of noise. Let \( a_{ps} \) denote the attenuation of saccade point \( s \) when the agent is looking at saccade point \( p \). This is a hardcoded function that specifies the characteristics of the
fovia and peripheral vision. When the agent is looking at image \( i \) at saccade point \( p \) the input it receives is given by

\[
x_s = a_{ps} f_{is} + \sigma \delta_s
\]

where \( \sigma \) is the standard deviation of the noise, and \( \delta_s \) is Gaussian noise for saccade point \( s \), which we will consider to be independent with zero mean and unit variance.

The agent begins its visual search with a flat prior distribution of images from the dataset. We will denote the probability of image \( i \) at time step \( t \) as \( d_t^i \), where

\[
\forall i, \quad d_0^i = \frac{1}{N}
\]

If at time \( t \), saccade point \( s^t \) is chosen and the agent receives input \( x^t_s \), the distribution at the next moment it time is given by

\[
d_{t+1}^i = \frac{p(x^t_i | i, s^t, d^t)}{\sum_{i'} p(x^t_i | i', s^t, d^t) d_{t}^{i'}} = \frac{p(x^t_i | i, s^t, d^t)}{\sum_{i'} p(x^t_i | i', s^t, d^t) d_{t}^{i'}}
\]

Since \( s^t \) must be chosen based only off of the current distribution of images \( d^t \), it cannot change the distribution of the images, so \( p(i | s^t, d^t) = p(i | d^t) = d_t^i \).

The distribution of images gives rise to an entropy at each point in time, and the predictive curiosity goal for this problem is to reduce that entropy as fast as possible. The greedy solution is then to choose at time \( t \) the saccade \( s^t \) such that the entropy of the distribution \( d_{t+1}^i \) is minimized. At time \( t \) when choosing saccade \( s^t \), the agent knows \( d^t \) and which \( s^t \) it is considering, but it does not know the input \( x^t \) it will receive or the resulting distribution \( d_{t+1}^i \). Fortunately, it has a distribution of \( x \) values for each possible saccade, given by

\[
p(x^t_i | s^t, d^t) = \sum_{i'} p(x^t_i | i', s^t, d^t) p(i' | s^t, d^t) = \sum_{i'} d_{t}^{i'} p(x^t_i | i', s^t)
\]

It knows \( d^t \) and \( p(x | i', s) \), so it has a distribution of \( x \) values for each saccade point it may choose.

Although the entropy of the distribution \( d_{t+1}^i \) is analytically intractable, it can be numerically approximated. To do this, first rewrite the entropy as follows:

\[
H(i | s^t, X^t, d^t) = -\sum_{i, x^t} p(i, x^t | s^t, d^t) \log p(i | x^t, s^t, d^t)
\]

\[
= -\sum_{i, x^t} p(i | s^t, d^t) p(x^t | i, s^t, d^t) \log p(i | x^t, s^t, d^t)
\]

\[
= -\sum_{i} d_{t}^{i} \sum_{x^t} p(x^t | i, s^t) \log p(i | x^t, s^t, d^t)
\]
Here, the capitalization is meant to reflect that the agent is calculating the entropy of \( i \) given a particular saccade point \( s^t \) and distribution \( d^t \), but taking the expected value over all possible inputs \( X^t \) from this saccade. \( X^t \) is then a random variable, which is why we capitalize it here. To approximate this entropy, the agent can sample a value of \( i \) from \( d^t \), sample a value of \( x^t \) from \( p(x^t|i, s^t) \), and calculate \( p(i|x^t, s^t, d^t) \) using \( d^t \) and the sampled \( x^t \). If it does this \( K \) times, sums up the negative logs of \( p(i|x^t, s^t, d^t) \), and at the end divides by \( K \), then this estimate should converge on the true entropy as \( K \) approaches infinity.

The agent now knows how to estimate for each place it can saccade to what the expected entropy of the world will be after looking there. The greedy predictive curiosity solution is simply to choose the saccade location where that entropy is expected to be the lowest, and this is exactly the strategy we use.

For our first test of the system, we crafted a test set of images with a known optimal pattern. This set is shown in Figure 5.1. Each test image is broken up into a 4x4 grid. The agent was given 16 saccade points, each of which was centered on one of the grid squares. The agent had no peripheral vision, and the fovia was not large enough to accommodate more than just the grid square the agent was looking at, but the noise was low. This made the agent’s input essentially binary at each time step: the agent knew either that the chosen grid location was white or was black.

Inspection of the dataset shows that in exactly half of the images, the top left corner square was white, and this is true of no other square. The agent should therefore look at that square first, and it gains 1 bit of information by doing so. If the top left square
Figure 5.2: One of the natural images for the saccade task. A) The original image. B) The filtered image. C) The filtered image with a saccade at the + sign (upper left).

was white, then in exactly half of the remaining images, the square to the right of the top left square was white, so the agent should look at that one. If the top left square was black, then the one under it is the most informative. This sort of pattern continues for four saccades. If the agent has looked at the optimal set of points, then there is exactly one square which will give a whole bit of information about the image, and that square is uniquely determined by the squares which the agent has already seen. After four saccades, the agent knows which image it is looking at. Not surprisingly, for $K = 1000$ we found that the agent almost always follows this optimal policy.

Next, we moved to natural images. We used a dataset of 16 natural images, 16 saccade points for each image, and 1000 samples to estimate the entropy of the next step. The filter applied to each saccade point was a Mexican hat. Figure 5.2A shows one of the images before filtering. Figure 5.2B shows the same image after being filtered, and Figure 5.2C shows the filtered image with noise, for a saccade pointed at the + sign. The size of the + sign reflects the size of the Mexican hat filter used. Also, the noise was added to the image before filtering in such a way as to make the filter statistics come out right for the chosen $\alpha_{ns}$ and $\sigma$ values. This is mathematically equivalent, since no two filters ever overlap, but it makes it easier to gauge the quality of the image visually.

<table>
<thead>
<tr>
<th>Chosen Saccade</th>
<th>Random Saccade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.22 ± 0.02 (bits)</td>
<td>2.46 ± 0.02 (bits)</td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of remaining entropy for saccades chosen using predictive curiosity against saccades chosen randomly

We again allowed the system four saccades for each image, and measured the entropy of the image after the saccades. We also compared this against choosing saccades...
randomly, and the results are shown in table 5.1. The remaining entropy when choosing saccades carefully was less than half of the remaining entropy when choosing saccades randomly.

5.2 Playing Mastermind

MasterMind™ is a two person game played on a board, shown in Figure 5.3. The board consists of pegs and holes. There are six colors of pegs, and as many of each color as is desired. The holes are arranged in rows, and each row has four holes. There are eleven such rows, plus one special one. At the beginning of the game, one player (the code setter) secretly chooses four pegs and places them in the special set of holes. These holes have a cover over them which allows only the code setter to see them. The object of the second player (the code breaker) is to figure out which pegs the code setter chose, and what positions he put them in. To do this, he makes guesses. Each guess is made by placing four pegs in one of the rows. The code setter scores this guess by indicating how many of the pegs in the guess are the correct colors in the correct positions, and how many are correct colors in wrong positions. For example, if the correct answer is Red, Red, Blue, Green, and the guess was Blue, White, Black, Green, then the code setter would indicate that one peg was in the correct location (the green one), and one was in the wrong position (the blue one). The code breaker continues to make guesses until either he guesses correctly and wins, or runs out of guesses (there are only 11 rows for guesses), and loses.

Predictive curiosity plays this game without knowing what the object is. Since the goal of predictive curiosity is to learn as much about the world as possible, it chooses its moves optimally in order to determine as quickly as possible what the hidden pattern is. Unfortunately, we do not have an optimal predictive curiosity algorithm to try this on. However, it can be approximated by an optimal greedy algorithm. The greedy algorithm learns as much as possible on each move, without regard to future moves. To do so, it keeps track of all of the possible hidden patterns which are consistent with the observations it has made (there are only 1,296 possible hidden patterns, so this can be done directly). At each step, it maximizes the information between the result of its guess and the hidden pattern. This information is given by:

$$I(R; P) = H(P) - H(P|R)$$

where $H(P)$ is the entropy of the hidden pattern just before making the guess (already conditioned on previous results), and $H(P|R)$ is the entropy of the pattern given the result. By choosing the moves which minimize the second term, predictive curiosity plays mastermind.

The greedy predictive curiosity player is vastly superior to an average human player. It can win in an average of 4.612 moves. Knuth’s greedy solver achieves an average
Figure 5.3: A completed game of MasterMind. Starting on the left, the code breaker makes guesses about the hidden pattern the code setter chose. The code setter places a black marker for each guess in the correct position and a white marker for each guess in the wrong position. The code breaker won in nine moves.

of 4.478 moves [Knuth, 1977], and optimal play results in an average of 4.340 moves [Koyama and Lai, 1993]. All three of these players make a maximum of six moves, whereas inexperienced human players sometimes lose even with eleven chances!

5.3 Near Optimal algorithm for an MDP

We now turn to an application of predictive curiosity where the world is clearly affected by the actions of the agent. Suppose that an agent lives in a world which is characterized by an unknown Markov decision process, with a known number of states $K$. If the agent has an arbitrary amount of time to run, and the MDP is such that all states can be visited infinitely often, then the agent can just maximize the restricted version of predictive curiosity given by:

$$\sum_{t=0}^{T} I(S^t, A^t; M|S^{<t}, A^{<t})$$

Assuming that the prior distribution over MDPs with the given number of states is flat, the entropy of the model given all past history is given by:

$$H(M|S^{<t}, A^{<t}) = \sum_{x,a} H(T^a_x|S^{<t}, A^{<t})$$
where $T_{xy}^a$ is the probability of transitioning from $x$ to $y$ given that the agent took action $a$, and $T_x^a$ is a vector containing the probabilities of transitioning to any state given that the current state is $x$ and the action is $a$. The entropy of the model can be rewritten in this way because $T_{x1}^a$ is independent of $T_{x2}^a$ given $\{S^{<t}, A^{<t}\}$. It is independent because the only information the agent has about $T_x^a$ comes from moments when $S^t = x$ and $A^t = a$. At these moments, $S^{t+1}$ conveys information about $T_x^a$. Since this is all the information the agent has, and it is all contained in $\{S^{<t}, A^{<t}\}$, then $T_{x1}^a$ must be independent of $T_{x2}^a$ given $\{S^{<t}, A^{<t}\}$, and the stated relationship holds.

Assuming that the prior distribution for $T_x^a$ was uniform, the probability of $T_x^a$ given $\{S^{<t}, A^{<t}\}$ is given by the Dirichlet distribution, and equals

$$p(T_x^a | S^{<t}, A^{<t}) = \frac{1}{Z(n_x^a)} \prod_{y=1}^{K} (T_{xy}^a)^{n_{xy}^a + 1}$$

where $n_{xy}^a$ is the number of times in the history when a transition from $x$ to $y$ was seen and action $a$ was taken. $Z(n_x^a)$ is given by

$$Z(n_x^a) = \frac{\prod_{y=1}^{K} \Gamma(n_{xy}^a + 1)}{\Gamma\left(K + \sum_{y=1}^{K} n_{xy}^a\right)}$$

where $\Gamma(x)$ is the Gamma function. The entropy of $T_x^a$ is given by

$$H(T_x^a | S^{<t}, A^{<t}) = \ln Z(n_x^a) - \sum_{y=1}^{K} n_{xy}^a \left[ \Psi(n_{xy}^a + 1) - \Psi \left( K + \sum_{z=1}^{K} n_{xz}^a \right) \right]$$

where $\Psi(x)$ is the digamma function, defined as $\frac{\partial}{\partial x} \ln \Gamma(x)$ [Honkela, 2001]. This expression can be simplified using Stirling’s approximation [Paris and Kaminsky, 2001] which states that

$$\ln \Gamma(x) = \ln(x - 1)! \approx (x - 1) \ln(x - 1) - x + 1$$

Using this, $\Psi(x)$ can be approximated by

$$\Psi(x) = \frac{\partial}{\partial x} \ln \Gamma(x) \approx \frac{\partial}{\partial x} \left[ (x - 1) \ln(x - 1) - x + 1 \right] = \ln(x - 1)$$
and \( \ln Z(n_x^a) \) can be approximated as

\[
\ln Z(n_x^a) = \ln \frac{\prod_{y=1}^{K} \Gamma(n_{xy}^a + 1)}{\Gamma(K + \sum_{y=1}^{K} n_{xy}^a)} = \sum_{y=1}^{K} \ln \Gamma(n_{xy}^a + 1) - \ln \Gamma(K + \sum_{y=1}^{K} n_{xy}^a) \\
\approx \sum_{y=1}^{K} (n_{xy}^a \ln n_{xy}^a - n_{xy}^a) - \left[ (K - 1 + \sum_{y=1}^{K} n_{xy}^a) \ln \left(K - 1 + \sum_{y=1}^{K} n_{xy}^a\right) - K + 1 - \sum_{y=1}^{K} n_{xy}^a \right] \\
= \sum_{y=1}^{K} n_{xy}^a \ln n_{xy}^a - \left( K - 1 + \sum_{y=1}^{K} n_{xy}^a \right) \ln \left( K - 1 + \sum_{y=1}^{K} n_{xy}^a \right) + K - 1
\]

Substituting this back into the entropy of \( T_x^a \) yields

\[
H(T_x^a | S^{<t}, A^{<t}) \approx \sum_{y=1}^{K} n_{xy}^a \ln n_{xy}^a - \left( K - 1 + \sum_{y=1}^{K} n_{xy}^a \right) \ln \left( K - 1 + \sum_{y=1}^{K} n_{xy}^a \right) + K - 1 \\
- \sum_{y=1}^{K} n_{xy}^a \ln n_{xy}^a + \ln \left( K - 1 + \sum_{z=1}^{K} n_{xz}^a \right) \\
= (1 - K) \ln \left( K - 1 + \sum_{y=1}^{K} n_{xy}^a \right) + K - 1
\]

Since Stirling’s approximation of \( \ln \Gamma(x) \) gets better as \( x \) gets larger, this estimate of the entropy will be worst when the agent first starts up, and will improve with time. It is well worth noting that this approximation of the entropy of \( T_x^a \) is only function of the total number of times that action \( a \) has been taken from state \( x \). It doesn’t actually matter which state the agent observed next.
Using this estimate, the information that the agent gets about the world when it takes action $a$ from state $x$ is given by

$$I(x, a; M|S^{<t}, A^{<t}) = H(T_x^a|S^{<t}, A^{<t}) - H(T_x^a|x, a, S^{<t}, A^{<t})$$

$$\approx (1 - K) \left[ \ln \left( K - 1 + \sum_{y=1}^{K} n_{xy}^a \right) - \ln \left( K + \sum_{y=1}^{K} n_{xy}^a \right) \right]$$

$$\approx \frac{K - 1}{K - 1 + \sum_{y=1}^{K} n_{xy}^a}$$

where this last equation is because

$$\ln(x + 1) - \ln x \approx \frac{d}{dx} \ln x = \frac{1}{x}$$

for reasonable sized $x$, which is the domain we are concerned with.

Returning again to the problem of optimally exploring an MDP world, suppose that the agent has been wandering in the world for long enough to make our approximations good, and the agent knows that it has only $N$ steps left before it dies, where $N$ is very small. The agent could in principle build a tree of all possible futures, and take the action for which the expected information is the greatest. It could figure out the optimal action at each of the leaves of the tree, and then move back a time step and figure out the optimal actions for each of those nodes, and so forth until it reaches the present moment. Unfortunately, the amount of work this requires is exponential in $N$.

To make the problem tractable, consider what happens if the last thing the agent does is take action $a$ from state $x$, see state $y$, and die. It gains a certain amount of information from this. At the present moment, if it were somehow magically transported to state $x$, took action $a$, and saw state $y$, it would also gain some information. If the agent has been around for a while and $N$ is small, then these two informations are going to be very close to each other, because it can’t take action $a$ from state $x$ enough times in the remaining $N$ time steps to make much difference. So when the agent considers the information it learns right before dying, it could approximate that information by the information it would get by taking the same action from the same state at the present moment. This approximation allows the agent to use a tractable algorithm.

To figure out what action to take, the agent needs to keep track of the information that it can gain from each state at each distance from its time horizon. We will call this information $I^{\tau}_{xy}$ for state $x$, where $\tau$ is the distance from the time horizon. If the agent is at state $x$ on its deathbed, then it should take the action it can gain the most information from, and it should set $I^{0}_{xy}$ to the amount of information it would gain by taking action $a$ from state $x$ given only what it knows now. If the agent is not on its deathbed, then it
should set $I_x^r$ as follows:

$$I_x^r = \max_a \left[ I(x, a; M | S^{<t}, A^{<t}) + \sum_{y=1}^{K} p(y|x, a) I_y^{r-1} \right]$$

where $p(y|x, a)$ the expected value of $T_{xy}^a$, given by [Gelman et al., 1995]

$$\langle T_{xy}^a \rangle = \frac{1 + n_{xy}^i}{K + \sum_{z=1}^{K} n_{xz}^a}$$

If the time horizon is $N$ steps away, then the agent should calculate for each state $I_x^0$, and then should calculate for each state $I_x^1$, and so forth all the way to $I_x^{N-1}$. It can then choose its action at the current moment in time as

$$a = \arg \max_a \left[ I(x, a; M | S^{<t}, A^{<t}) + \sum_{y=1}^{K} p(y|x, a) I_y^{r-1} \right]$$

This algorithm allows an agent which exists in an MDP world with a finite time horizon to pick a nearly optimal action, in time proportional to how far out the time horizon is. In the infinite horizon case, there are at least two things that can be done. The first is to simply pick a large time horizon, and at any point in time pick the best action for that time horizon. This allows the agent to choose actions which are best for a reasonable time frame, although perhaps not for an infinite time frame. Another approach is to use discounted rewards. The agent could choose the action which maximizes

$$\sum_t \gamma^t I(S^t, A^t; M | S^{<t}, A^{<t})$$

at each point in time. While the concept of discounted information seems very strange, it is exactly analogous to the use of discounted rewards in reinforcement learning. For our purposes, the agent could calculate $I_x^r$ as

$$I_x^r = \max_a \left[ I(x, a; M | S^{<t}, A^{<t}) + \gamma \sum_{y=1}^{K} p(y|x, a) I_y^{r-1} \right]$$

and choose its best action $a$ as

$$a = \arg \max_a \left[ I(x, a; M | S^{<t}, A^{<t}) + \gamma \sum_{y=1}^{K} p(y|x, a) I_y^{r-1} \right]$$

Now, to solve the time horizon problem, the agent needs only to carry the calculation of $I_x^r$ out far enough that $\gamma^r$ is negligible.
We tested this algorithm first on a very simple MDP world. The world has 10 states and the agent has two actions. The states are organized into a line, and the actions are left or right. Whenever the agent chooses to go right, it succeeds unless it is in the rightmost state, in which case it remains in place. However, when the agent chooses to go left, it goes left with probability $\frac{1}{4}$ and stays in place otherwise. If the agent is in the leftmost state, then it stays in place.

We compared a predictive curiosity agent to a random agent. Since Q learning explores by making random choices, this compares how fast a Q learning agent explores to how fast a predictive curiosity agent explores. The predictive curiosity agent did not use discounted rewards, but instead used a fixed time horizon of 100 time steps. We simulated both agents for 100,000 time steps. The results are shown in table 5.2.

<table>
<thead>
<tr>
<th></th>
<th>Predictive Curiosity</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6893</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>9903</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>9930</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>9925</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>9997</td>
<td>91</td>
</tr>
<tr>
<td>6</td>
<td>9933</td>
<td>374</td>
</tr>
<tr>
<td>7</td>
<td>10120</td>
<td>1212</td>
</tr>
<tr>
<td>8</td>
<td>10004</td>
<td>4631</td>
</tr>
<tr>
<td>9</td>
<td>10137</td>
<td>18878</td>
</tr>
<tr>
<td>10</td>
<td>13158</td>
<td>74805</td>
</tr>
</tbody>
</table>

Table 5.2: Comparison of the behavior of the predictive curiosity agent with the random agent. A) The number of times each state was visited. State 1 is the leftmost state. B) The number of times each action was used.

The random agent clearly did not explore the world well; it never once reached the leftmost three states. In contrast, the predictive curiosity agent explored the world states almost equally, with only the two boundary states explored unevenly. The random agent chose its actions equally, whereas the predictive curiosity agent chose its actions in a very biased fashion so that it explored the states more evenly.

To test this on a more interesting system, we moved on to a two dimensional maze and tested the predictive curiosity agent against the random agent, both in the presence of wind and in its absence. Figure 5.4 shows the results for the case where there was no wind. Figure 5.4A shows how well the maze was explored by the random agent. The area of the dot in each room of the maze is proportional to the number of times the room was visited. Figure 5.4B shows the same diagram for the predictive curiosity agent. It visited all of the states fairly equally. Figure 5.4C shows a comparison of how
Figure 5.4: Exploration of the deterministic world. A) How well the maze was explored in the first 1000 time steps by the random agent. B) How well the maze was explored in the first 1000 time steps by the predictive curiosity agent. C) A comparison of the distribution of states visited. The dashed line is the random agent, and the solid line is the predictive curiosity agent.

Figure 5.5: Exploration of the windy world. A) How well the maze was explored in the first 2000 time steps by the random agent. B) How well the maze was explored in the first 2000 time steps by the predictive curiosity agent. C) A comparison of the distribution of states visited. The dashed line is the random agent, and the solid line is the predictive curiosity agent.

evenly the states were visited by the two agents. This graph displays the same data as the previous two in a different form. It shows a histogram of the number of times each state was visited by an agent, with the states sorted by their visitation count. The random agent visited some states a great many times and completely missed nearly half of them, whereas the predictive curiosity agent visited them nearly equally.

The same experiment was run again in the presence of wind blowing up and to the right. In this case, when an agent went down or left, the action succeeded with
probability 0.8. When it failed, the agent remained in place. The results of this are shown in Figure 5.5. In this case, the agents were allowed to explore for 2000 time steps, since neither agent explored the maze as well as without wind. It is interesting that the predictive curiosity agent is still affected by wind. One could imagine that it would be impervious to anything of the sort. However, this appears to be just a temporary phenomenon. Figure 5.6 shows the results of running the predictive curiosity agent and the random agent on a 5x5 maze with wind for 50,000 iterations. It seems that in the early iterations, the predictive curiosity agent is affected by wind, but as the number of iterations grows, the agent learns to overcome the wind. By 50,000 iterations, the agent is almost totally unaffected by wind, whereas the random agent will always be affected by it.

In conclusion, the predictive curiosity agent is able to explore MDP worlds far more effectively than the random agent. When the world is biased against particular areas being explored well, the agent takes longer to get around to them, but once it learns how to find them, it explores them just as well as it explores the rest of the world.

5.3.1 Reinforcement Learning

Curiosity contains an intrinsic method of reward: an agent which is curious can be trained through the judicious application of boredom. This can be illustrated using a maze with a penalty box. In the normal maze, the agent can take four action from each state. In this new maze, the agent can choose from eight actions at each state. Four of them correspond to speaking English, and the other four correspond to speaking French. In the top half of the maze, the agent must speak English, and in the bottom
half, the agent must speak French. If the agent speaks the wrong language, he ends up in the penalty box. The penalty box is a specific square where the agents' actions are completely ignored. After some (stochastic) amount of time, the agent is returned to the maze again. With this setup, the agent must not speak the wrong language very often, or it will not get much chance to explore the maze. This setup is shown in Figure 5.7A.

Figure 5.7: A) An example maze with a penalty box. The agent must speak English in the top half of the maze and French in the bottom half, or else it ends up in the penalty box. B) The number of mistakes made as a function of time for three different settings of the penalty box. C) The number of mistakes made as a function of time for three different maze sizes.

Since the agent is curious, it learns not to speak the wrong language very often. Figure 5.7B shows how often the agent spoke the wrong language as a function of
how long on average it had to spend in the penalty box for each mistake. There are several interesting things about this graph. First, the agent does indeed learn over time to speak the wrong language less often. Therefore, the basic idea of using boredom seems to work. Second, the longer the agent is stuck, the fewer mistakes it makes. This is in keeping with the intuition that it becomes more careful as the punishment increases. Third, the agent never stops making mistakes. This is true because the agent is never absolutely certain what will happen if it speaks French in English territory. After exploring the rest of the maze very well, it becomes worth its while to try yet again to see if something different happens when it speaks the wrong language. While this basic curiosity may be soured somewhat by lengthening the time in the penalty box, a curious agent can never be made uncurious.

Another aspect of the agent’s behavior can be explored by changing the size of the maze. If there is only one room in the maze, and it is the penalty box, then the agent will not care whether it is speaking the wrong language or not. If the maze has thousands of rooms, then the agent will have a great deal of exploration to do without speaking the wrong language. Therefore, we expect that as the maze size increases, the agent will make fewer mistakes for the same penalty time. This is indeed the case, as is shown in Figure 5.7C. This curve shows two things of interest. First, it shows that the agent does make fewer mistakes on larger mazes. Second, it shows that the larger the maze, the longer it takes to converge on the number of mistakes it makes. This makes sense, because whenever it encounters a new room, it does not know which actions mean what. Therefore, as a simple lower bound, the agent must make four mistakes for every room before it has any idea how to avoid the penalty box. Larger mazes therefore imply longer learning times.

We thus conclude that curious agents can indeed be trained using boredom, but only to a point. Once the agent has thoroughly explored its environment, it becomes bored everywhere, and the penalty box is just a different place to be bored. Therefore, it will take the actions it has been trained not to take, simply because it hasn’t taken them very much. This problem can be somewhat alleviated by making the world more interesting, but it can never be altogether cured.

5.4 Conclusions

In this chapter we have demonstrated predictive curiosity in three different domains: choosing saccades, playing a game, and exploring mazes. In all three cases, there was a well defined optimal solution which was intractable, and there was a reasonable approximation to it. All three of these approximations had good results. Further, we have shown an application of predictive curiosity to a reinforcement learning problem, with interesting conclusions. Much future work needs to be done in order to develop practical algorithms of predictive curiosity to run in more challenging environments, but the cases we have shown here at least begin to demonstrate its merits.
6 Conclusion

Many principles have been proposed to explain the behavior of the brain. The coding models include efficient coding, predictive coding, and temporal invariance. Bayesian surprise has been used to model attention, and models of action include reinforcement learning and intelligent adaptive curiosity.

This dissertation proposed a new principle, called predictive action, which unifies the above principles and clarifies the relationships between them. Predictive action is an information theoretic principle which defines curiosity in terms of gaining Shannon information about the world. Like predictive coding, it is a purely intrinsic principle, in the sense that an agent can evaluate its behavior according to this principle purely based on its inputs and outputs, while remaining agnostic about the real nature of the world.

The remainder of this chapter summarizes the contributions of the dissertation and presents future research directions.

6.1 Dissertation Contributions

The contributions of this dissertation can be divided into three areas: work on theoretical principles, practical algorithms to implement principles, and results of practical algorithms. The key theoretical contributions include:

- A novel formulation of predictive coding which solves difficulties with previous definitions
- A novel definition of curiosity
- A novel principle (predictive action) to explain the behavior of the brain
- Unification of many existing principles with predictive action

The key algorithms/results include:
- A novel efficient coding algorithm
- A novel predictive coding algorithm
- A novel algorithm to perform near optimal exploration of an MDP
- Several demonstrations of predictive action on interesting systems

6.2 Future Research Directions

There is much future work still to be done. In section 3.2.2, a different, perhaps better version of curiosity was noted. This definition should be studied, and if it turns out to be better, it should be merged with predictive coding to form a superior version of predictive action. Next would be to work towards an optimal, tractable algorithm for performing predictive action for any given world. Ideally, such an algorithm would be in the form of a spiking neural network which implements a form of incremental Levin search (e.g. [Schmidhuber, 2004]) tailored to doing predictive action. This seems to be a long way off, however, and research on practical algorithms for doing predictive action on limited world models (such as MDPs or POMDPs) may give more insight in the short term.

There are a very wide variety of circumstances where predictive action can be used, and this dissertation has covered only a very narrow slice of them. Predictive action should be tried in as many circumstances as possible in order to gain certainty that the principle is adequate for explaining intelligent behavior.

Finally, this dissertation has drawn some parallels between the brain and a predictive action system. If predictive action proves to be the organizing principle for the brain, then many more such parallels must exist. Much work could be done along the lines of matching up the behavior of predictive action with that of the brain.

6.3 Concluding Remarks

The concept of intelligence has been studied for over a century. According to “Mainstream Science on Intelligence” [Gottfredson, 1997a], which was signed by 52 intelligence researchers in 1994, intelligence is

a very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience. It is not merely book learning, a narrow academic skill, or test-taking smarts. Rather, it reflects a broader and deeper capability for comprehending our surroundings – “catching on”, “making sense” of things, or “figuring out” what to do.
This definition was proposed by psychologists and it is measured using IQ tests [Gottfredson, 1997b]. Although this definition and the means of measuring it are completely useless to artificial intelligence research, the AI community has proposed nothing better. We have instead turned to working particular problems on agreed upon data sets.

Intelligence must be defined rigorously. Just as information theory put a rigorous foundation to such fields as data compression and communication over noisy channels, so a proper definition of intelligence will put a rigorous foundation to nearly any pursuit in artificial intelligence.

This dissertation has proposed a rigorous definition of how intelligent an agent’s behavior is for any given circumstance. We have demonstrated this definition for a few simple examples, but a vast amount of research will have to go into testing any definition of intelligence. The definition given here may well be flawed, but even if this proves to be the case, the work we have done may provide some footing on which to make a better one. We hope we have at least provided a good starting point.
Bibliography


A Temporal Invariance and Predictive Coding

A.1 The Damped Brownian Motion Process

The DBM process is a Markov chain defined by

\[ x_{t+1} = \gamma x_t + \epsilon n_t \]

where \( x_t \) is the value at time \( t \), \( n_t \) is zero mean unit variance Gaussian noise, and \( 0 < \gamma < 1 \) and \( \epsilon > 0 \) control the properties of the process. The probability \( p(x_{t+1}|x_t) \) is given by:

\[ p(x_{t+1}|x_t) = \frac{1}{\epsilon \sqrt{2\pi}} e^{-\frac{(x_{t+1}-\gamma x_t)^2}{2\epsilon^2}} \]

If \( x_t \) is sampled from the stationary distribution of \( x \), then \( x_{t+1} \) should be also. Therefore, if we assume that \( x_t \) is sampled from a given distribution, and we can prove that this implies that \( x_{t+1} \) is sampled from the same distribution, then we have properly guessed the stationary distribution. We will guess that \( x_t \) is sampled from a zero mean unit variance Gaussian distribution, such that the probability of \( x_t \) is given by

\[ p(x_t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x_t^2}{2}} \]

The marginal distribution for \( x_{t+1} \) is then given by

\[
\int_{-\infty}^{\infty} p(x_t)p(x_{t+1}|x_t)dx_t = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x_t^2}{2}} \frac{1}{\epsilon \sqrt{2\pi}} e^{-\frac{(x_{t+1}-\gamma x_t)^2}{2\epsilon^2}} dx_t
\]

\[
= \int_{-\infty}^{\infty} \frac{1}{\epsilon 2\pi} e^{-\frac{x_t^2}{2} - \frac{(x_{t+1}-\gamma x_t)^2}{2\epsilon^2}} dx_t
\]
This exponent can be rewritten as follows

\[-\frac{x_t^2}{2} - \frac{(x_{t+1} - \gamma x_t)^2}{2\epsilon^2}\]

\[= -\frac{x_t^2}{2} - \frac{x_{t+1}^2}{2\epsilon^2} + \frac{\gamma x_t x_{t+1}}{\epsilon^2} - \frac{\gamma^2 x_t^2}{2\epsilon^2}\]

\[= -x_t^2 \left(\frac{1}{2} + \frac{\gamma^2}{2\epsilon^2}\right) + x_t \frac{\gamma x_{t+1}}{\epsilon^2} - \frac{x_{t+1}^2}{2\epsilon^2}\]

\[= -\left(\frac{1}{2} + \frac{\gamma^2}{2\epsilon^2}\right) \left[x_t^2 - x_t \frac{\gamma x_{t+1}}{\epsilon^2 \left(\frac{1}{2} + \frac{\gamma^2}{2\epsilon^2}\right)}\right] - \frac{x_{t+1}^2}{2\epsilon^2}\]

\[= -\frac{x_{t+1}^2}{2\epsilon^2} + \frac{1}{2} \left(1 + \frac{\gamma^2}{\epsilon^2}\right) \left(\frac{\gamma x_{t+1}}{\epsilon^2 + \gamma^2}\right)^2\]

\[= -\frac{1}{2} \left(1 + \frac{\gamma^2}{\epsilon^2}\right) \left(x_t - \frac{\gamma x_{t+1}}{\epsilon^2 + \gamma^2}\right)^2 - \frac{x_{t+1}^2}{2\epsilon^2} + \frac{\gamma^2 x_{t+1}^2}{2\epsilon^2 \left(\epsilon^2 + \gamma^2\right)}\]

\[= -\frac{x_{t+1}^2}{2} \left(\frac{\epsilon}{\sqrt{\epsilon^2 + \gamma^2}}\right)^2 - \frac{x_{t+1}^2}{2} \left(\frac{\gamma}{\sqrt{\epsilon^2 + \gamma^2}}\right)^2\]

The original integral then becomes

\[\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x_t^2}{2}} \frac{(x_{t+1} - \gamma x_t)^2}{2\epsilon^2} dx_t\]

\[= \frac{1}{\sqrt{\epsilon^2 + \gamma^2}} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{x_{t+1}^2}{2(\sqrt{\epsilon^2 + \gamma^2})^2}} dx_t\]

\[= \frac{1}{\sqrt{\epsilon^2 + \gamma^2}} \frac{1}{\sqrt{2\pi}} e^{-\frac{x_{t+1}^2}{2(\sqrt{\epsilon^2 + \gamma^2})^2}}\]

Therefore the stationary distribution of \(x_{t+1}\) is the same as the stationary distribution of \(x_t\), and both are zero mean unit covariance Gaussian, provided that the following condition is met:

\[\epsilon^2 + \gamma^2 = 1\]

DBM is specified by two degrees of freedom, and one of them is used in making the variance of the stationary distribution unity. The other is used to control the temporal variance of the process. Assuming that \(\epsilon\) and \(\gamma\) are chosen to satisfy the above criteria,
the temporal variance can be calculated as follows:

\[
\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(x_t)p(x_{t+1}|x_t)(x_{t+1}-x_t)^2 \, dx_{t+1} \, dx_t
\]

\[
= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x_t^2}{2}} \frac{1}{\epsilon \sqrt{2\pi}} e^{-\frac{(x_{t+1}-\epsilon x_t)^2}{2\epsilon^2}} (x_{t+1}^2 + x_t^2 - 2 x_t x_{t+1}) \, dx_{t+1} \, dx_t
\]

\[
= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x_t^2}{2}} (x_t^2 + \gamma^2 x_t^2 + \epsilon^2 - 2\gamma x_t^2) \, dx_t
\]

\[
= 1 + \gamma^2 + \epsilon^2 - 2\gamma
\]

\[
= 2(1 - \gamma)
\]

Thus if the temporal variance is denoted \(\sigma^2\), then \(\gamma\) and \(\epsilon\) are given by

\[
\gamma = 1 - \frac{\sigma^2}{2}
\]

\[
\epsilon = \sigma \sqrt{1 - \frac{\sigma^2}{4}}
\]

### A.2 Linear Combinations of DBM are not DBM

Consider the following three processes:

\[
x_t^A = \gamma^A x_{t-1}^A + \epsilon^A n_t^A
\]

\[
x_t^B = \gamma^B x_{t-1}^B + \epsilon^B n_t^B
\]

\[
y_t = \alpha x_t^A + \beta x_t^B
\]

\(x^A\) and \(x^B\) are DBM, and the question is whether or not \(y\) is DBM. We prove below that \(y\) is DBM exactly when the temporal variances of \(x^A\) and \(x^B\) are equal. To do this, let us rewrite the definition of the distribution \(y^t\) as follows:

\[
y_t = \alpha x_t^A + \beta x_t^B
\]

\[
= \alpha \left( \gamma^A x_{t-1}^A + \epsilon^A n_t^A \right) + \beta \left( \gamma^B x_{t-1}^B + \epsilon^B n_t^B \right)
\]

\[
= \left( \alpha \gamma^A x_{t-1}^A + \beta \gamma^B x_{t-1}^B \right) + \left( \alpha \epsilon^A n_t^A + \beta \epsilon^B n_t^B \right)
\]

\[
= \gamma^y y_{t-1} + \epsilon^y n_t^y
\]
In order for the last two lines to be equal, \( \alpha \gamma^A x_{t-1}^A + \beta \gamma^B x_{t-1}^B \) must equal \( \gamma^Y y_{t-1} \). \( \gamma^Y y_{t-1} \) can be written as

\[
\gamma^T y_{t-1} = \gamma^Y \left( \alpha x_{t-1}^A + \beta x_{t-1}^B \right)
\]

\[
= \alpha \gamma^Y x_{t-1}^A + \beta \gamma^Y x_{t-1}^B
\]

\[
\neq \alpha \gamma^A x_{t-1}^A + \beta \gamma^B x_{t-1}^B
\]

These last two lines cannot be equal for all values of \( x_{t-1}^A \) and \( x_{t-1}^B \) unless \( \gamma^A = \gamma^B \). Since \( \sigma^2 = 2(1 - \gamma) \), we have

\[
\gamma^A = \gamma^B \iff \sigma^A = \sigma^B
\]

Therefore, the sum of two DBM processes is not DBM if the temporal variances of the processes are not equal. If they are equal, then we can let \( \gamma^Y = \gamma^A = \gamma^B \), and the decay term of \( y \) behaves properly.

The noise terms of interest are thus given by

\[
\alpha \epsilon A n^A_t + \beta \epsilon B n^B_t \overset{?}{=} \epsilon Y n^Y_t
\]

Since \( n^A_t \) and \( n^B_t \) are Gaussian, the values of \( n^Y_t \) which satisfy the above equation are also distributed by a Gaussian, so we can let these be equal. Thus the sum of two DBM processes with the same temporal variance is a new DBM process.

### A.3 DBM is an Information Theoretic Stationary Point

Suppose that \( x \) and \( y \) are independent DBM processes with possibly different temporal variances, and they are coded at time \( t \) by \( z_t \), which is some function \( f(x_t, y_t) \) plus Gaussian noise.

\[
x_{t+1} = \gamma_x x_t + \epsilon_x n^x_t
\]

\[
y_{t+1} = \gamma_y y_t + \epsilon_y n^y_t
\]

\[
z_t = f(x_t, y_t) + \sigma n^z_t
\]

where \( n_t \) denotes zero mean unit variance Gaussian noise, and \( \sigma \) is the variance of the noise added to the system’s output. If the goal of the system is to choose \( f \) to maximize

\[
I(x_{t+1}, y_{t+1}; z_t)
\]

then \( f(x_t, y_t) = ax_t \) is an stationary point of this objective function under a fixed variance constraint. The variance of \( z_t \) is the variance of \( ax_t \), which is \( a^2 \), plus the
variance of the noise added to \( f(x_t, y_t) \), which is \( \sigma^2 \). Under the constraint that the variance of \( z_t \) must be unity, \( a^2 + \sigma^2 = 1 \), so \( a = \sqrt{1 - \sigma^2} \). That this is a stationary point can be proven as follows

\[
\delta \left[ I(x_{t+1}, y_{t+1}; z_t) + \lambda \left( \int_{z_t} p(z_t) z_t^2 dz_t - 1 \right) \right] \\
= \delta H(x_{t+1}, y_{t+1}) + \delta H(z_t) - \delta H(x_{t+1}, y_{t+1}, z_t) + \lambda \int_{z_t} \delta p(z_t) z_t^2 dz_t \\
= \delta H(z_t) - \delta H(x_{t+1}, y_{t+1}, z_t) + \lambda \int_{z_t} \delta p(z_t) z_t^2 dz_t
\]

where \( \delta \) denotes the variation with respect to \( f \), and \( \lambda \) is a Lagrange multiplier. The first term is given by

\[
\delta H(z_t) \\
= -\delta \int_{z_t} p(z_t) \ln p(z_t) dz_t \\
= -\int_{z_t} \delta p(z_t) \left[ 1 + \ln p(z_t) \right] dz_t \\
= -\delta \int_{z_t} p(z_t) dz_t - \int_{z_t} \delta p(z_t) \ln p(z_t) dz_t \\
= -\delta (1) - \int_{z_t} \delta \left[ \int_{x_t, y_t} p(x_t, y_t)p(z_t|x_t, y_t) dx_t dy_t \right] \ln p(z_t) dz_t \\
= -\int_{x_t, y_t} p(x_t, y_t) \int_{z_t} \delta p(z_t|x_t, y_t) \ln p(z_t) dz_t dx_t dy_t \\
= -\int_{x_t, y_t} p(x_t, y_t) \int_{z_t} \delta \left[ \frac{1}{\sqrt{2\pi} \sigma} e^{-\frac{(z_t-f(x_t, y_t))^2}{2\sigma^2}} \right] \ln p(z_t) dz_t dx_t dy_t \\
= -\int_{x_t, y_t} p(x_t, y_t) \int_{z_t} \frac{1}{\sqrt{2\pi} \sigma} e^{-\frac{(z_t-f(x_t, y_t))^2}{2\sigma^2}} \\
\cdot \frac{[z_t - f(x_t, y_t)] \delta f(x_t, y_t)}{\sigma^2} \ln p(z_t) dz_t dx_t dy_t \\
= -\int_{x_t, y_t} p(x_t, y_t) \delta f(x_t, y_t) \int_{z_t} p(z_t|x_t, y_t) \frac{z_t - f(x_t, y_t)}{\sigma^2} \ln p(z_t) dz_t dx_t dy_t
\]
The second term is given by

\[
\delta H(x_{t+1}, y_{t+1}, z_t)
\]

\[
= -\delta \int_{x_{t+1}, y_{t+1}, z_t} p(x_{t+1}, y_{t+1}, z_t) \ln p(x_{t+1}, y_{t+1}, z_t) dx_{t+1} dy_{t+1} dz_t
\]

\[
= -\delta \int_{x_{t+1}, y_{t+1}, z_t} \delta p(x_{t+1}, y_{t+1}, z_t) [1 + \ln p(x_{t+1}, y_{t+1}, z_t)] dx_{t+1} dy_{t+1} dz_t
\]

\[
= -\delta(1) - \int_{x_{t+1}, y_{t+1}, z_t} \delta p(x_{t+1}, y_{t+1}, z_{t+1}) \ln p(x_{t+1}, y_{t+1}, z_t) dx_{t+1} dy_{t+1} dz_t
\]

\[
= -\int_{x_{t+1}, y_{t+1}, z_t} \delta \left[ \int_{x_{t+1}, y_{t+1}} p(x_t, y_t) p(x_{t+1}, y_{t+1}, z_t | x_t, y_t) dx_t dy_t \right]
\]

\[
\times \ln p(x_{t+1}, y_{t+1}, z_t) dx_{t+1} dy_{t+1} dz_t
\]

\[
= -\int_{x_t, y_t} \int_{x_{t+1}, y_{t+1}} p(x_{t+1}, y_{t+1} | x_t, y_t) \delta p(z_t | x_t, y_t) dx_t dy_t
\]

\[
\times \ln p(x_{t+1}, y_{t+1}, z_t) dx_{t+1} dy_{t+1} dz_t dx_t dy_t
\]

\[
= -\int_{x_t, y_t} \int_{x_{t+1}, y_{t+1}} p(x_{t+1}, y_{t+1} | x_t, y_t) \delta f(x_t, y_t) \int_{x_{t+1}, y_{t+1}, z_t} p(x_{t+1}, y_{t+1} | x_t, y_t) p(z_t | x_t, y_t) dx_{t+1} dy_{t+1} dz_t
\]

\[
\times \frac{z_t - f(x_t, y_t)}{\sigma^2} \ln p(x_{t+1}, y_{t+1}, z_t) dx_{t+1} dy_{t+1} dz_t dx_t dy_t
\]

The third term is given by

\[
\lambda \int_{z_t} \delta p(z_t) z_t^2 dz_t
\]

\[
= \lambda \int_{z_t} \delta \left[ \int_{x_t, y_t} p(x_t, y_t) p(z_t | x_t, y_t) dx_t dy_t \right] z_t^2 dz_t
\]

\[
= \int_{x_t, y_t} p(x_t, y_t) \delta f(x_t, y_t) \int_{z_t} \lambda p(z_t | x_t, y_t) \frac{z_t - f(x_t, y_t)}{\sigma^2} z_t^2 dz_t dx_t dy_t
\]

Notice that all three of these terms begin with the same integral over \( x_t \) and \( y_t \), and that this term includes \( \delta f(x_t, y_t) \). We can drop this outside integral and rewrite our
condition for a stationary function as

\[ \forall x_t, y_t, \quad \int_{z_t} p(z_t|x_t, y_t) \frac{z_t - f(x_t, y_t)}{\sigma^2} \left[ \lambda z_t^2 - \ln p(z_t) \right] \\
+ \int_{x_{t+1}, y_{t+1}} p(x_{t+1}, y_{t+1}|x_t, y_t) \ln p(x_{t+1}, y_{t+1}, z_t) dx_{t+1} dy_{t+1} \right] dz_t = 0 \]

To see why this has to be true, suppose that for some values of \( x_t \) and \( y_t \) this was violated. In that case, \( \delta f(x_t, y_t) \) could be chosen to be nonzero for just these values, and the resulting integral would be nonzero.

Since we are proving that \( f(x_t, y_t) = x_t \) is a stationary point, we will evaluate the above expression using that function. In an ideal world, it would be good to find every function \( f \) which is a stationary point, and prove that our function is the only maximum. Unfortunately, even proving that our function is a maximum is beyond the scope of this paper. To prove that it is a stationary point, we will evaluate each term in turn. The first term is:

\[ \frac{\lambda}{\sigma^2} \int_{z_t} p(z_t|x_t, y_t) \left[ z_t^3 - z_t^2 f(x_t, y_t) \right] dz_t \]

Since \( p(z_t|x_t, y_t) \) is a Gaussian with mean \( f(x_t, y_t) \) and standard deviation \( \sigma \), this is equal to

\[ \frac{\lambda}{\sigma^2} \left\{ f(x_t, y_t) \left[ f(x_t, y_t)^2 + 3\sigma^2 \right] - \left[ f(x_t, y_t)^2 + \sigma^2 \right] f(x_t, y_t) \right\} = 2\lambda f(x_t, y_t) = 2\lambda a x_t \]

where the last equality is for the case we are considering, where \( f(x_t, y_t) = ax_t \). The second term is given by

\[ \int_{z_t} p(z_t|x_t, y_t) \frac{z_t - f(x_t, y_t)}{\sigma^2} \ln p(z_t) dz_t \]

To evaluate this, let us first evaluate \( p(z_t) \) for the case of \( f(x_t, y_t) = ax_t \). In this case, \( z_t \) is the sum of two Gaussians. The first, \( ax_t \), is zero mean with variance \( a^2 \). The second, the noise, is zero mean, with variance \( \sigma^2 \). The sum of these is a Gaussian with zero mean and variance \( a^2 + \sigma^2 \). Since \( a \) was chosen to make this variance unity, we have

\[ p(z_t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z_t^2}{2}} \]
Returning to our expression, we have

\[
\int_{z_t} p(z_t | x_t, y_t) \frac{z_t - f(x_t, y_t)}{\sigma^2} \ln p(z_t) dz_t
\]

\[
= \int_{z_t} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(z_t - a x_t)^2}{2\sigma^2}} \frac{z_t - a x_t}{\sigma^2} \ln \left( \frac{1}{\sqrt{2\pi}} e^{-\frac{z_t^2}{2}} \right) dz_t
\]

\[
= -\ln \sqrt{2\pi} \int_{z_t} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(z_t - a x_t)^2}{2\sigma^2}} \frac{z_t - a x_t}{\sigma^2} dz_t
\]

\[
- \frac{1}{2\sigma^2} \int_{z_t} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(z_t - a x_t)^2}{2\sigma^2}} \left( z_t^3 - a x_t z_t^2 \right) dz_t
\]

\[
= -a x_t \left( a^2 x_t^2 + 3\sigma^2 \right) - a x_t \left( a^2 x_t^2 + \sigma^2 \right)
\]

\[
= -a x_t
\]

The last of our three terms is given by

\[
\int_{z_t} p(z_t | x_t, y_t) \frac{z_t - f(x_t, y_t)}{\sigma^2} p(x_{t+1}, y_{t+1} | x_t, y_t) \ln p(x_{t+1}, y_{t+1}, z_t) dx_{t+1} dy_{t+1} dz_t
\]

We will begin simplifying this by evaluating \( p(x_{t+1}, y_{t+1}, z_t) \) as follows:

\[
p(x_{t+1}, y_{t+1}, z_t)
\]

\[
= \int_{x_t, y_t} p(x_{t+1}, y_{t+1}, z_t, x_t, y_t) dx_t dy_t
\]

\[
= \int_{x_t, y_t} p(x_{t+1} | x_t) p(y_{t+1} | y_t) p(z_t | x_t, y_t) p(x_t) p(y_t) dx_t dy_t
\]

\[
= \int_{x_t, y_t} \frac{1}{e^{x_t} e^{y_t} \sigma (2\pi)^{\frac{3}{2}}} e^{-\frac{(x_{t+1}-a x_t)^2}{2\sigma^2} - \frac{(y_{t+1}-a y_t)^2}{2\sigma^2} - \frac{(z_t-a x_t)^2}{2\sigma^2} - \frac{z^2}{2}} dx_t dy_t
\]

Conveniently, DBM is reversible, in the sense that if the DBM process is used to generate a sequence of samples, the reverse of the samples is also distributed according to the same DBM process. That is, \( p(a \rightarrow b) = p(b \rightarrow a) \), where the \( a \) and \( b \) are samples and the arrow indicates temporal sequence. This can be seen by studying the exponent.
in $p(x_t, x_{t+1})$, which is

$$-\frac{(x_{t+1} - \gamma_x x_t)^2}{2\epsilon_x^2} - \frac{x_t^2}{2}$$

$$= -\frac{x_{t+1}^2}{2\epsilon_x^2} + \frac{\gamma_x x_t x_{t+1}}{\epsilon_x^2} - \frac{\gamma_x^2 x_t^2}{2\epsilon_x^2} - \frac{x_t^2}{2}$$

$$= -\frac{x_{t+1}^2}{2\epsilon_x^2} + \frac{\gamma_x x_t x_{t+1}}{\epsilon_x^2} - \frac{(\gamma_x^2 + \epsilon_x^2) x_t^2}{2\epsilon_x^2}$$

$$= -\frac{x_{t+1}^2}{2\epsilon_x^2} + \frac{\gamma_x x_t x_{t+1}}{\epsilon_x^2} - \frac{x_t^2}{2\epsilon_x^2}$$

where the last line comes from the definition of DBM, which states that $\gamma_x^2 + \epsilon_x^2 = 1$. Notice that this last line is symmetric in $x_t$ and $x_{t+1}$, so DBM is reversible.

This relation allows us to rewrite our expression as

$$\int_{x_t, y_t} \frac{1}{\epsilon_x \epsilon_y \sigma(2\pi)^2} e^{-\frac{(x_t - \gamma_y x_{t+1})^2}{2\epsilon_y^2} - \frac{y_t^2}{2\sigma^2} - \frac{(y_t - \gamma_y y_{t+1})^2}{2\sigma^2}} \, dx_t \, dy_t$$

$$= \frac{1}{2\pi} e^{-\frac{x_t^2}{\epsilon_x^2} - \frac{z_t^2}{\sigma^2}} \int_{x_t} \frac{1}{\epsilon_x \sigma(2\pi)^2} e^{-\frac{(x_t - \gamma_x x_{t+1})^2}{2\epsilon_x^2} - \frac{z_{t+1}^2}{2\sigma^2} - \frac{(z_{t+1} - x_{t+1})^2}{2\sigma^2}} \, dx_t$$

The exponent inside this integral can be rewritten as

$$-\frac{(x_t - \gamma_x x_{t+1})^2}{2\epsilon_x^2} - \frac{(z_t - a x_t)^2}{2\sigma^2}$$

$$= -x_t^2 \left( \frac{1}{\epsilon_x^2} + \frac{a^2}{2\sigma^2} \right) + 2x_t \left( \frac{\gamma_x x_{t+1}}{2\epsilon_x^2} + \frac{a z_t}{2\sigma^2} \right) - \frac{\gamma_x^2 x_{t+1}^2}{2\epsilon_x^2} - \frac{z_t^2}{2\sigma^2}$$

$$= -\frac{1}{2} \left( \frac{1}{\epsilon_x^2} + \frac{a^2}{2\sigma^2} \right) \left[ x_t^2 - 2x_t \frac{\gamma_x x_{t+1}}{2\epsilon_x^2} + \frac{a z_t}{2\sigma^2} + \left( \frac{\gamma_x x_{t+1}}{2\epsilon_x^2} + \frac{a z_t}{2\sigma^2} \right)^2 \right]$$

$$+ \frac{\gamma_x x_{t+1}^2}{2\epsilon_x^2} - \frac{a z_t^2}{2\sigma^2} - \frac{x_{t+1}^2}{2\epsilon_x^2}$$

$$= -\frac{1}{2} \left( \frac{1}{\epsilon_x^2} + \frac{a^2}{2\sigma^2} \right) \left( x_t - \frac{\gamma_x x_{t+1} \sigma^2 + a z_t \epsilon_x^2}{a^2 \epsilon_x^2 + \sigma^2} \right)^2$$

$$+ \frac{\gamma_x x_{t+1} \sigma^2 + 2a \gamma_x x_{t+1} z_t \epsilon_x^2 \sigma^2 + a^2 \epsilon_x^4 \sigma^2}{2\epsilon_x^2 \sigma^2 (a^2 \epsilon_x^2 + \sigma^2)}$$

$$- \frac{\gamma_x^2 x_{t+1} \sigma^4 + 2a \gamma_x x_{t+1} z_t \epsilon_x^2 \sigma^2 + a^2 \epsilon_x^4 \sigma^2}{2\epsilon_x^2 \sigma^2 (a^2 \epsilon_x^2 + \sigma^2)}$$

$$- \frac{z_t^2 \epsilon_x^2 (a^2 \epsilon_x^2 + \sigma^2)}{2\epsilon_x^2 \sigma^2 (a^2 \epsilon_x^2 + \sigma^2)}$$
\[ p(x_{t+1}, y_{t+1}, z_t) = \frac{1}{2\pi} e^{-\frac{x_{t+1}^2 + y_{t+1}^2}{2}} \] 
\[ = \frac{1}{\sqrt{a^2 e_x^2 + \sigma^2}} e^{-\frac{x_{t+1}^2}{2a^2 e_x^2 + \sigma^2}} \] 
\[ = \frac{1}{\sqrt{a^2 e_x^2 + \sigma^2}^\frac{3}{2}} e^{-\frac{x_{t+1}^2}{2}} \] 
\[ \cdot \frac{1}{\sqrt{\frac{1}{\sigma^2} + \frac{2}{\sigma^2}}} e^{-\frac{y_{t+1}^2}{2}} \ln p(x_{t+1}, y_{t+1} | x_t, y_t) dx_{t+1} dy_{t+1} \] 
\[ = c - \frac{\gamma_x x_{t+1}^2}{2(a^2 e_x^2 + \sigma^2)} dx_{t+1} \] 
\[ = c' - \frac{z_t^2}{2} - 2az_x^2 dx_{t+1} \] 
\[ = c' - \frac{z_t^2 - 2az_x^2}{2(a^2 e_x^2 + \sigma^2)} \]
where \( c \) and \( c' \) denote everything that is not a function of \( z_t \). The last of the three terms is thus given by

\[
\int_{z_t} p(z_t|x_t, y_t) \frac{z_t - ax_t}{\sigma^2} \left[ c' - \frac{z_t^2 - 2az_t\gamma^2 x_t}{2(a^2\epsilon^2_x + \sigma^2)} \right] dz_t
\]

The integral over \( c' \) is zero, since the mean of \( z_t \) is \( ax_t \), so we have

\[
- \frac{1}{2\sigma^2(a^2\epsilon^2_x + \sigma^2)} \int_{z_t} p(z_t|x_t, y_t)(z_t - ax_t)(z_t^2 - 2az_t\gamma^2 x_t) dz_t
\]

\[
= - \frac{1}{2\sigma^2(a^2\epsilon^2_x + \sigma^2)} \int_{z_t} p(z_t|x_t, y_t) \left[ z_t^3 - az_t^2(2\gamma^2 x_t + x_t) + 2z_t a^2\gamma^2 x^2_t \right] dz_t
\]

\[
= - \frac{ax_t(a^2 x^2_t + 3\sigma^2) - a(a^2 x^2_t + \sigma^2)(2\gamma^2 x_t + x_t) + 2a^3\gamma^2 x^3_t}{2\sigma^2(a^2\epsilon^2_x + \sigma^2)}
\]

\[
= - \frac{3a\sigma^2 x_t - 2a\sigma^2\gamma^2 x_t - a\sigma^2 x_t}{2\sigma^2(a^2\epsilon^2_x + \sigma^2)}
\]

\[
= - \frac{ax_t(1 - \gamma^2_x)}{a^2\epsilon^2_x + \sigma^2}
\]

\[
= - \frac{ax_t}{a^2 + \sigma^2/\epsilon^2_x}
\]

where this last term is because, by the definition of DBM, \( 1 - \gamma^2_x = \epsilon^2_x \).

Our constraint can now be written as

\[
\forall x_t, y_t, \quad 2\lambda ax_t - ax_t - \frac{ax_t}{a^2 + \sigma^2/\epsilon^2_x} = 0
\]

This constraint is satisfied when

\[
\lambda = \frac{1}{2a^2 + 2\sigma^2/\epsilon^2_x} + \frac{1}{2}
\]

Therefore, \( f(x_t, y_t) = ax_t \) is a stationary point of the objective function under the unity variance constraint.

**A.4 DBM is a Temporal Invariance Stationary Point**

Borrowing the definitions of \( x_t, y_t, \) and \( z_t \) from the previous appendix, suppose now that we wish to find the \( f \) which maximizes

\[
\langle (z_{t+1} - z_t)^2 \rangle
\]
under the constraints of zero mean and unit variance, where \( \langle \cdot \rangle \) denotes expected value over time. We will prove that \( f(x_t, y_t) = ax_t \) is again a stationary point when \( a = \sqrt{1 - \sigma^2} \). Since \( x_t \) is zero mean, this choice clearly satisfies the zero mean constraint, and it is proved in the previous appendix that it satisfies the unit variance constraint as well. To prove that it maximizes the objective function under these constraints, we will evaluate the variation of the objective function with respect to \( f \), as shown below

\[
\delta \int_{z_t, z_{t+1}} p(z_t, z_{t+1})(z_{t+1} - z_t)^2 dz_t \, dz_{t+1} + \lambda_1 \delta \int_{z_t} p(z_t) z_t \, dz_t + \lambda_2 \delta \left( \int_{z_t} p(z_t) z_t^2 \, dz_t - 1 \right) = \int_{z_t, z_{t+1}} \delta p(z_t, z_{t+1})(z_{t+1} - z_t)^2 dz_t \, dz_{t+1} + \int_{z_t} \delta p(z_t) (\lambda_1 z_t + \lambda_2 z_t^2) \, dz_t
\]

The first term can be evaluated as follows:

\[
\int_{z_t, z_{t+1}} \delta p(z_t, z_{t+1})(z_{t+1} - z_t)^2 dz_t \, dz_{t+1} = \delta \int_{z_t, z_{t+1}} p(z_t, z_{t+1})(z_{t+1}^2 - 2z_{t+1}z_t + z_t^2) \, dz_t \, dz_{t+1} = \delta \left[ \int_{z_t} p(z_{t+1}) z_{t+1}^2 \, dz_{t+1} \right] \]

\[
+ \left[ \int_{z_t, z_{t+1}} p(z_t, z_{t+1}) 2z_{t+1}z_t \, dz_t \, dz_{t+1} + \int_{z_t} p(z_t) z_t^2 \, dz_t \right]
\]

\[
= 2\delta \left[ \int_{z_t} p(z_t) z_t^2 \, dz_t + \int_{z_t, z_{t+1}} p(z_t, z_{t+1}) z_t z_{t+1} \, dz_t \, dz_{t+1} \right]
\]

\[
= 2\delta \left\{ \int_{x_t, y_t} p(x_t, y_t) z_t^2 dx_t \, dy_t \, dz_t + \int_{x_t, y_t, z_t, x_{t+1}, y_{t+1}, z_{t+1}} p(x_t, y_t, z_t, x_{t+1}, y_{t+1}, z_{t+1}) \, dz_t \, dz_{t+1} \right\}
\]

\[
= 2\delta \left\{ \int_{x_t, y_t} p(x_t, y_t) [f(x_t, y_t)^2 + \sigma^2] \, dx_t \, dy_t \right\}
\]

\[
+ \int_{x_t, y_t, x_{t+1}, y_{t+1}} p(x_t, y_t, x_{t+1}, y_{t+1}) f(x_t, y_t) f(x_{t+1}, y_{t+1}) dx_t \, dy_t \, dx_{t+1} \, dy_{t+1} \right\}
\]
Putting these two terms back together yields

\[
= 2 \int_{x_t,y_t} \delta p(x_t, y_t) 2 f(x_t, y_t) \delta f(x_t, y_t) dx_t \, dy_t \\
+ 2 \int_{x_t,y_t,x_{t+1},y_{t+1}} p(x_t, y_t, x_{t+1}, y_{t+1}) f(x_t, y_t) \delta f(x_{t+1}, y_{t+1}) dx_t \, dy_t \, dx_{t+1} \, dy_{t+1} \\
+ 2 \int_{x_t,y_t,x_{t+1},y_{t+1}} p(x_t, y_t, x_{t+1}, y_{t+1}) \delta f(x_t, y_t) f(x_{t+1}, y_{t+1}) dx_t \, dy_t \, dx_{t+1} \, dy_{t+1}
\]

\[
= 4 \int_{x_t,y_t} p(x_t, y_t) \delta f(x_t, y_t) \left[ f(x_t, y_t) \\
+ \int_{x_{t+1},y_{t+1}} p(x_{t+1}, y_{t+1}|x_t, y_t) f(x_{t+1}, y_{t+1}) dx_{t+1} \, dy_{t+1} \right] dx_t \, dy_t
\]

This last line used the fact that DBM is symmetric \((p(a \to b) = p(b \to a))\), which was proven in the previous appendix.

The second term can be evaluated as follows:

\[
\int_{x_t,y_t,z_t} \delta p(z_t) (\lambda_1 z_t + \lambda_2 z_t^2) dz_t \\
= \delta \int_{x_t,y_t,z_t} p(x_t, y_t, z_t) (\lambda_1 z_t + \lambda_2 z_t^2) dz_t \\
= \delta \int_{x_t,y_t} p(x_t, y_t) \{ \lambda_1 f(x_t, y_t) + \lambda_2 [f(x_t, y_t)^2 + \sigma^2] \} dx_t, dy_t \\
= \int_{x_t,y_t} p(x_t, y_t) [\lambda_1 \delta f(x_t, y_t) + 2 \lambda_2 f(x_t, y_t) \delta f(x_t, y_t)] dx_t \, dy_t \\
= \int_{x_t,y_t} p(x_t, y_t) \delta f(x_t, y_t) [\lambda_1 + 2 \lambda_2 f(x_t, y_t)] dx_t \, dy_t
\]

Putting these two terms back together yields

\[
\int_{x_t,y_t} p(x_t, y_t) \delta f(x_t, y_t) \left[ \lambda_1 + (2 \lambda_2 + 1) f(x_t, y_t) \\
+ \int_{x_{t+1},y_{t+1}} p(x_{t+1}, y_{t+1}|x_t, y_t) f(x_{t+1}, y_{t+1}) dx_{t+1} \, dy_{t+1} \right] dx_t \, dy_t
\]

In order for this to be zero for all \(\delta f(x_t, y_t)\), the following must be true:

\[
\forall x_t, y_t, \quad \lambda_1 + (2 \lambda_2 + 1) f(x_t, y_t) \\
+ \int_{x_{t+1},y_{t+1}} p(x_{t+1}, y_{t+1}|x_t, y_t) f(x_{t+1}, y_{t+1}) dx_{t+1} \, dy_{t+1} = 0
\]
To verify that $f(x_t, y_t) = ax_t$ is a stationary point, the integral can be evaluated as follows:

$$
\int_{x_{t+1}, y_{t+1}} p(x_{t+1}, y_{t+1}|x_t, y_t) f(x_{t+1}; y_{t+1}) dx_{t+1} dy_{t+1}
\begin{align*}
= & \int_{x_{t+1}} p(x_{t+1}|x_t) a\gamma x_t dx_{t+1} \\
= & a\gamma x_t
\end{align*}
$$

The constraint above now becomes

$$
\forall x_t, y_t. \quad \lambda_1 + (2\lambda_2 + 1) ax_t + a\gamma x_t = 0
$$

which is satisfied when $\lambda_1 = 0$ and $\lambda_2 = \frac{2x_t+1}{2}$. Therefore, $f(x_t, y_t) = ax_t$ is a stationary point for our objective function under the zero mean and unit variance constraints.

It is not surprising that $\lambda_2$ came out nonzero. Any function can be scaled, and since this would scale the temporal variance, $\lambda_2$ has to be nonzero to prevent this. However, it is interesting to note that $\lambda_1$ came out zero. The natural interpretation of this is that the zero mean constraint is not needed. In truth, it is not needed to make our transformation a stationary point, but we believe it is needed in order to make it a minimum. That our transform is a minimum is somewhat intuitive through symmetry arguments. Consider some class of transformations $f^{\mu}$, where each preserves the unity variance of the signal but sets the mean to $\mu$. Clearly, when $\mu = \pm 1$, the temporal variance of the signal must be zero, since the signal must be constant at $\mu$ in order to preserve unity variance from zero. The temporal variance, when plotted against $\mu$, must rise as $\mu$ leaves $-1$, reach a peak, and fall again to be zero when $\mu$ equals $1$. It would not seem surprising that for any such class, the peak happens at $\mu = 0$. If this were the case, then the change in temporal variance with respect to an infinitesimal change in $\mu$ from zero would be zero, which would explain why $\lambda_1 = 0$.

### A.5 Value of Predictive Coding Objective Function for a DBM Process

The information that $z_t$ conveys about $x_{t+1}$ is given by

$$
I(z_t, x_{t+1}) = -\int_{z_t, x_{t+1}} p(z_t, x_{t+1}) \ln \frac{p(z_t)}{p(z_t | x_{t+1})} dz_t dx_{t+1}
$$
From Appendix A.3 we have
\[
p(z_t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z_t^2}{2}}
\]
\[
p(z_t | x_{t+1}) = \frac{1}{\sqrt{a^2 \epsilon_x^2 + \sigma^2 \sqrt{2\pi}}} e^{-\frac{(z_t - a\gamma_x x_{t+1})^2}{2(a^2 \epsilon_x^2 + \sigma^2)}}
\]

The information is therefore given by
\[
- \int_{z_t, x_{t+1}} p(z_t, x_{t+1}) \left[ -\frac{z_t^2}{2} + \ln \sqrt{a^2 \epsilon_x^2 + \sigma^2} + \frac{(z_t - a\gamma_x x_{t+1})^2}{2(a^2 \epsilon_x^2 + \sigma^2)} \right] dz_t \; dx_{t+1}
\]
\[
= \frac{1}{2} - \ln \sqrt{a^2 \epsilon_x^2 + \sigma^2} - \frac{1}{2(a^2 \epsilon_x^2 + \sigma^2)} \int_{z_t, x_{t+1}} p(z_t, x_{t+1}) (z_t^2 - 2a\gamma_x x_{t+1} z_t + a^2 \gamma_x^2 x_{t+1}^2) \; dz_t \; dx_{t+1}
\]
\[
= \frac{1}{2} - \ln \sqrt{a^2 \epsilon_x^2 + \sigma^2} - \frac{1 + a^2 \gamma_x^2}{2(a^2 \epsilon_x^2 + \sigma^2)} - \frac{a\gamma_x}{a^2 \epsilon_x^2 + \sigma^2} \int_{z_t, x_{t+1}} p(z_t, x_{t+1}) x_{t+1} z_t \; dx_{t+1} \; dz_t
\]

The integral is given by
\[
\int_{z_t, x_{t+1}} p(z_t | x_{t+1}) p(x_{t+1}) x_{t+1} z_t \; dx_{t+1} \; dz_t
\]
\[
= \int_{x_{t+1}} p(x_{t+1}) a\gamma_x x_{t+1}^2 \; dx_{t+1}
\]
\[
= a\gamma_x
\]

So the full information is:
\[
\frac{1}{2} - \ln \sqrt{a^2 \epsilon_x^2 + \sigma^2} - \frac{1 + a^2 \gamma_x^2}{2(a^2 \epsilon_x^2 + \sigma^2)} - \frac{a^2 \gamma_x^2}{a^2 \epsilon_x^2 + \sigma^2}
\]
\[
= \frac{1}{2} - \frac{1}{2} \ln \left( a^2 \epsilon_x^2 + \sigma^2 \right) - \frac{1 - a^2 \gamma_x^2}{2(a^2 \epsilon_x^2 + \sigma^2)}
\]

In the noiseless case, \( a = 1 \) and \( \sigma = 0 \), so the expression reduces to \(- \ln \epsilon_x\). Higher values of \( \epsilon_x \) mean that the temporal variance is higher, and they also mean that the information is lower, so lower temporal variance DBM processes have higher information content.
B Predictive Coding Network

B.1 Definition of “synchronous”

By “synchronous”, we mean the following:

\[
\int_0^T f(t) dt + \int_0^T g(t) dt = \lim_{dt \to 0} \left[ \sum_{i=0}^{T/dt} f(i dt) dt + \sum_{i=0}^{T/dt} g(i dt) dt \right]
\]

as opposed to

\[
\int_0^T f(t) dt + \int_0^T g(t) dt = \left[ \lim_{dt \to 0} \sum_{i=0}^{T/dt} f(i dt) dt \right] + \left[ \lim_{dt \to 0} \sum_{i=0}^{T/dt} g(i dt) dt \right]
\]

which is to say that the whole system runs on a synchronous clock that takes infinitesimal time steps, and the limit of \( dt \to 0 \) extends over all equations. This may seem an unnecessary complication, as well as rather trivial. However, it gives rigorous meaning to such convenient expressions as \( \sum_{t} X_t^t, \sum_{t} X_t^t f(t), \) and \( \sum_{X \leq t} f(X \leq t) \). This last one is probably the most important, since a more standard representation of it would properly be written as

\[
\sum_{n_1=0}^{\infty} \frac{1}{n_1!} \int_0^t dt_{1,0} \int_0^{t_{1,1}} dt_{1,1} \cdots \int_0^{t_{1,n_1-1}} dt_{1,n_1-1} \\
\sum_{n_2=0}^{\infty} \frac{1}{n_2!} \int_0^t dt_{2,0} \cdots \int_0^{t_{2,n_2-1}} dt_{2,n_2-1} \\
\vdots \\
\sum_{n_N=0}^{\infty} \frac{1}{n_N!} \int_0^t dt_{N,0} \cdots \int_0^{t_{N,n_N-1}} dt_{N,n_N-1} f(n, \bar{t})
\]
which is perhaps a bit much. In this expression, $N$ is the number of neurons in the layer, the first subscript on times is neuron number, the second is the spike index, and $\bar{t}$ represents the array of spike times. The main reason this is so complicated is because the number of infinitesimals changes with the value being summed over (ten spikes need ten infinitesimals). In our notation, the number of $X$ values is countably infinite, and the sum is proper in its simple form. The onus is on $f$ to be infinitesimal of the proper order. To this end, whenever $p$ is used, it represents a probability distribution, not a density, and will be infinitesimal often. Since $dt$ is global, one could write out the value of these distributions with the $dt$’s in place, but we don’t because it is rather messy.

### B.2 Analytical details of the predictive coding network derivation

#### B.2.1 Terms equalling zero

\[
\sum_{t, h, f, a^t} p(f, \tilde{h}) \frac{\partial p(a^t | f, \tilde{h})}{\partial w_k} = \sum_{t, h, f} p(f, \tilde{h}) \left( \sum_{a^t} p(a^t | f, \tilde{h}) \right) \frac{\partial}{\partial w_k} 1 \\
= \sum_{t, h, f} p(f, \tilde{h}) \left( \sum_{a^t} p(a^t | f, \tilde{h}) \right) 1 \\
= 0
\]

\[
\sum_{t, h, f, a^t} p(f | a^t, \tilde{h}) p(\tilde{h}) \frac{\partial p(a^t | h)}{\partial w_k} = \sum_{t, h, a^t} p(\tilde{h}) \frac{\partial p(a^t | h)}{\partial w_k} \\
= \sum_{t, h} p(\tilde{h}) \left( \sum_{a^t} p(a^t | \tilde{h}) \right) \frac{\partial}{\partial w_k} 1 \\
= \sum_{t, h} p(\tilde{h}) \frac{\partial}{\partial w_k} 1 \\
= 0
\]
B.2.2 Dealing with neurons separately

\[
\sum_{t,s,a} \ln \frac{p[a^t|\bar{f}(s^{\geq t}, a^{<t})]}{p[a^t|h(a^{<t})]} \frac{\partial}{\partial w_k} p(s, a)
\]

\[
= \sum_{t,s,a} \left[ \sum_i \ln \frac{p[a_i^t|\bar{f}(s^{\geq t}, a^{<t}), a_i^{<t}]}{p[a_i^t|h(a^{<t}), a_i^{<t}]} \right] \frac{\partial}{\partial w_k} p(s, a)
\]

\[
= \sum_{t,s,a} \left[ \sum_i \ln \frac{p[a_i^t|\bar{f}(s^{\geq t}, a^{<t}), a_i^{<t} = 0]}{p[a_i^t|h(a^{<t}), a_i^{<t} = 0]} \right] \frac{\partial}{\partial w_k} p(s, a)
\]

That the condition that \(a_i^{<t} = 0\) does not change the value of the expression can be verified by looking separately at the two cases, where \(a_i^t = 1\) and where \(a_i^t = 0\). In the \(a_i^t = 1\) case, note that the probability of coincident spikes ever occurring during a finite length run of the system is zero. Therefore, the probability of any other spike occurring while \(i\) is firing is zero, and the condition is valid. In the \(a_i^t = 0\) case, the condition is violated a finite number of times within a finite time interval. However, the contribution of the logarithm is infinitesimal at each moment in time. Over a finite period of time, these contributions integrate to a finite value, but a finite number of contributions of the wrong infinitesimal amount results only in infinitesimal error. The difference between the second line and the third is infinitesimal, and thus ignorable.

\[
\sum_{t,s,a} \left[ \sum_i \ln \frac{p[a_i^t|\bar{f}(s^{\geq t}, a^{<t}), a_i^{<t} = 0]}{p[a_i^t|h(a^{<t}), a_i^{<t} = 0]} \right] \frac{\partial}{\partial w_k} p(s, a)
\]

\[
= \sum_{t,s,a} \left[ \sum_i \ln \frac{p[a_i^t|\bar{f}(s^{\geq t}, a^{<t})]}{p[a_i^t|h(a^{<t})]} \right] \frac{\partial}{\partial w_k} p(s, a)
\]

This is true because

\[
p \left[ a_i^t|\bar{f}(s^{\geq t}), h(a^{<t}), a_i^{<t} = 0 \right] = p \left[ a_i^t|\bar{f}(s^{\geq t}), h(a^{<t}) \right]
\]

and

\[
p \left[ a_i^t|h(a^{<t}), a_i^{<t} = 0 \right] = p \left[ a_i^t|h(a^{<t}) \right]
\]
To see this, note that for an arbitrary set of conditions $X$,

$$p[a_k^t = 0 | X] = \sum_{a_i^t} p(a_k^t = 0 | X, a_i^t) p(a_i^t | X)$$

$$= \{1 - r_k[X, a_i^t = 0] dt\} \{1 - r_i[X] dt\}$$

$$+ \{1 - r_k[X, a_i^t = 1] dt\} r_i[X] dt$$

$$= 1 - r_k[X, a_i^t = 0] dt - r_i[X] dt + r_k[X, a_i^t = 0] r_i[X] dt^2$$

$$+ r_i[X] dt - r_k[X, a_i^t = 1] r_i[X] dt^2$$

$$= 1 - r_k[X, a_i^t = 0] dt$$

$$= p[a_k^t = 0 | X, a_i^t = 0]$$

where we have defined $p(a_k^t = 1 | \ldots) = r_k(\ldots) dt$. Since $p[a_k^t = 1 | X] = 1 - p[a_k^t = 0 | X]$, this result implies that $p[a_k^t | X] = p[a_k^t | X, a_i^t = 0]$ for $i \neq k$. This extends by induction to $p(a_k^t | X) = p(a_k^t | X, a_{<k}^t = 0)$. The base case is that $p(a_k^t | Y) = p(a_k^t | Y)$, which is clearly true. The inductive step is that $p(a_k^t | Y, a_{<n}^t = 0) = p(a_k^t | Y, a_{<n+1}^t = 0)$, which is an immediate consequence the above theorem for $X = \{Y, a_{<n}^t = 0\}$ so long as $n + 1 < k$, which is the case of interest.
B.2.3 Derivative of history

\[
\frac{\partial}{\partial w_k} p(s, a) = \frac{\partial}{\partial w_k} \prod_{\tau} p(s^\tau, a^\tau | s^{<\tau}, a^{<\tau})
\]

\[
= p(s, a) \sum_{\tau} \frac{\partial}{\partial w_k} p(s^\tau, a^\tau | s^{<\tau}, a^{<\tau})
\]

\[
= p(s, a) \sum_{\tau} \frac{\partial}{\partial w_k} \left[ p(a^\tau | s^{<\tau}, a^{<\tau}) p(s^\tau | s^{<\tau}, a^{<\tau}) \right] \quad \frac{\partial}{\partial w_k} p(s^\tau, a^\tau | s^{<\tau}, a^{<\tau})
\]

\[
= p(s, a) \sum_{\tau} \frac{\partial}{\partial w_k} p(a^\tau | s^{<\tau}, a^{<\tau}) \quad p(s^\tau | s^{<\tau}, a^{<\tau})
\]

\[
= p(s, a) \sum_{\tau} \frac{\partial}{\partial w_k} p(a^\tau | s^{<\tau}, a^{<\tau})
\]

\[
= p(s, a) \sum_{\tau} \frac{\partial}{\partial w_k} \prod_{i=1}^{N_s} p(a_i^\tau | a_{<i}^\tau, s^{<\tau}, s^{<\tau})
\]

\[
= p(s, a) \sum_{\tau} \frac{\partial}{\partial w_k} \prod_{i=1}^{N_s} p(a_i^\tau | a_{<i}^\tau, a^{<\tau}, s^{<\tau})
\]

\[
= p(s, a) \sum_{\tau} \frac{\partial}{\partial w_k} \prod_{i=1}^{N_s} p(a_i^\tau | a_{<i}^\tau, s^{<\tau})
\]

\[
= p(s, a) \sum_{\tau} \frac{\partial}{\partial w_k} p(a_k^\tau | s^{<\tau}, a^{<\tau}) \prod_{i=1}^{N_s} p(a_i^\tau | a_{<i}^\tau, s^{<\tau})
\]

\[
= p(s, a) \sum_{\tau} \frac{\partial}{\partial w_k} p(a_k^\tau | s^{<\tau}, a^{<\tau}) \prod_{i=1}^{N_s} p(a_i^\tau | a_{<i}^\tau, s^{<\tau})
\]

B.2.4 Definition of L

Let \( L^{tf}_k \) be defined as

\[
L^{tf}_k = \sum_{t,s,a} p(s, a) \left[ \sum_{t_b < \tau \leq t} \frac{\partial}{\partial w_k} g_k(s^{<\tau}, a^{<\tau}) \right] - \int_{t_b}^{t_e} \frac{\partial}{\partial w_k} g_k(s^{<\tau}, a^{<\tau}) d\tau
\]
Since \( t_b \) and \( t_e \) do not change on the interval \((t_0, t_f]\), this definition is valid. It is easy to keep track of \( L^t_k \) because any valid interval \((t_0, t_f]\) within the interval \((t_b, t_e]\) contributes the following to \( L^t_k \):

\[
\frac{a^t_f}{a_k} \frac{\partial}{\partial w_k} g_k(s^{<t_f}, a^{<t_f}) - (t_f - t_0) \frac{\partial}{\partial w_k} g_k(s^{<t_f}, a^{<t_f})
\]