MODELING EARLY VISION: PROBABILISTIC COMPUTATION USING SPIKING NEURONS, POPULATION CODES, AND CUDA

by

DANIEL ROBERT COATES

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER SCIENCE

Portland State University
2009

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis describes the design, implementation, and use of a spiking neural network model of visual cortex. Several aspects are studied, concerning biologically-inspired image processing, neural data representation, and parallelization of the network simulation architecture.

The study of networks of spiking neurons is a vibrant research area in computational neuroscience, as these models of neural activity provide a more realistic description of biological data than some of the abstractions that have previously been employed. It has been shown experimentally and theoretically that these networks can exhibit a great richness of functional behavior.

Spiking neural networks are also interesting from the standpoint of theoretical computer science, since they present a computational paradigm that leverages massive parallelism of simple computational nodes with a unique binary communication channel. Using a network of these nodes with a statistical interpretation scheme balances high resolution computation with grace-

ful degradation in the face of uncertain input data and communication noise.

There are several barriers, however, to the use of spiking neuron models in real-world applications. First and foremost is the lack of general principles for problem-solving. Spiking neural networks do not yet have an equivalent to the "backpropagation of error" method that has made artificial neural networks practical. Additionally, the noise inherent in spiking networks can present difficulties when using traditional information processing techniques. Therefore, my primary goal in this work was to identify reliable computational methods using a spiking neural network.

This is the genesis of the software architecture, so engineering, rather than new science, is at the forefront. To this end, I reproduced a family of existing models from the literature in computational neuroscience. The main motivation for this initial approach was to ensure the correctness of each component of the simulation. This step was critical since biologically-inspired models have many parameters, and it is often unclear which characteristics are functionally significant.

Image processing is an ideal application for alternative computational techniques due to the input data ambiguity, implicit parallelism, and the need for more effective algorithms. Neuroscience offers a "working model" in the form of the visual cortex of the brain, which has exquisite visual processing capability. Great strides in computer vision have been due to inspiration from neuroscience, and the present model makes use of many of these notions from biologically-inspired image processing.

Since very little is certain about the specifics of processing in the visual cortex, my ambitions in image processing are modest in this work. As in the model I reproduced, the benchmark task consists of the determination of the orientation of a single rotated line placed in the center of a small grayscale image. The network response is analyzed to quantify the resolution of the code signaling the result of the computation.

The final aspect of the thesis, high-performance computation, was motivated by the need for processing power to handle many trials and large scale models. Neural systems are inherently massively parallel, so they lend themselves to techniques from concurrent programming. The repetitive, "embarrassingly parallel" operations involved in simulating networks of neurons map well to SIMD (single instruction, multiple data) architectures. I leveraged this parallelism using the OpenMP and CUDA architectures.

OpenMP and CUDA are methods to parallelize small kernels of highly parallel code. OpenMP is a popular C-based software abstraction for use on shared-memory CPUs. CUDA is a new architecture for parallel programming created by NVIDIA for use on their graphics processing cards (GPUs). There is growing interest in the use of GPUs for general-purpose computation (known as GPGPU), motived by the availability and low cost of many-core graphics hardware. I implemented components of the spiking simulator in CUDA and OpenMP and profiled the performance on a variety of architectures.

Chapter 2 reviews background material, primarily from neuroscience, that

motivates the model I reproduced. Chapter 3 is an in-depth description of the mechanics of the simulation platform. Chapter 4 presents the results of my experiments with the model, while Chapter 5 offers an analysis of the results and general discussion about the research outcomes and possible future work.

Chapter 2

Background and Related Work

My primary ambition in pursuing this research was to better understand possible computational principles in the brain. Therefore I strove to remain as true to accepted neuroscience as possible. Rather than building "biologically-inspired" algorithms, I endeavored to only utilize principles with a solid basis in experimentation.

Spiking models form the centerpiece of this thesis, and their biological basis is discussed in Section 2.1. While the simulation of spiking behavior facilitates sophisticated temporal codes, for this work I instead opted to examine statistically based approaches to neural coding, which involve computation using the aggregate activity of neuronal populations. This approach is radically different from other strategies. Most computational methods, including artificial neural networks and even many spiking models, have a "digital" flavor that is at odds with biological systems. Often too much emphasis is placed on the response of individual neurons. John Von Neumann wrote in 1958:

> It should also be noted that the message-system used in the nervous system, is of an essentially *statistical* character. In other words, what matters are not the precise positions of definite markers, digits, but the statistical characteristics of their occurrence.[1]

Whether this feature of neural representation is crucial for understanding biological intelligence remains to be seen, but it is possible that the analysis of such codes could provide insight into aspects of cognition such as adaptation, learning, and generalization that still lack convincing artificial realizations. At minimum, statistical codes, introduced in Section 2.3, have a robustness that cannot be matched by comparatively brittle digital representations.

Besides spiking neuron models and statistical codes, the other aspect of neuroscience I draw from is the architecture of the mammalian visual cortex, described in Section 2.2. This incredibly effective system is one of the most heavily studied structures in the brain, and is the inspiration for many models in biologically-inspired computer vision[2][3]. The moderately detailed implementation I reproduced parallels low-level visual processing of simple image features, such as the detection of edges, which occurs in the primary visual cortex.

2.1 Computational Modeling of Neurons

In neuroscience modeling, the smallest computational nodes typically represent single neurons. Some researchers divide this abstraction even further

into cellular components, but that level of detail is not relevant to the present work. At minimum, neurons have a cell body where computation is performed, and connections by "wires" (called *axons* and *dendrites*) to many other neurons. As noted by John Von Neumann, neurons, like digital gates, have a single output on one axon. The inputs, however, typically number in the thousands, unlike typical logical circuits[1].

It is almost universally agreed that electro-chemical spikes, or *action potentials*, are the means by which neurons communicate. The overwhelming belief is that information is transferred between neurons solely by these pulses, which can be interpreted as binary streams. The full interpretation of spike trains is much more complicated and controversial, however, and is covered further in Section 2.3.

The connections between the cell body and its axons and dendrites are known as *synapses*. Through chemical mechanisms, synapses can depress or facilitate spike transmission. Theoretically, this characteristic is modeled by weighting the different inputs. This feature is of prime importance in pattern processing using artificial neural networks, and can be interpreted as a dot product operation between a set of inputs and an input mask.

Neural activity is directional, and spikes affect downstream neurons in either a positive or negative fashion. *Excitatory* neurons send spikes that have an additive effect on the neurons which they are connected to, while *inhibitory* neurons have a subtractive or possibly divisive[4] effect.

2.1.1 Artificial Neural Networks

The first important distinction to make is between spiking neurons and the "analog" units employed by artificial neural networks, first proposed by McCulloch and Pitts[5]. Although inspired by the brain, this neural model transmits continuous numerical values over its axons, rather than binary pulses. This abstraction is justified by the argument that these numbers correspond to an average firing rate.

The McCulloch and Pitts model also supports transmission of either negative or positive values by each node, in contrast to spiking models, in which each unit can be strictly excitatory or inhibitory. The benefit of the McCulloch and Pitts model is that since messages between nodes can be any continuous value, neurons can be interpreted as performing vector operations, and the analytical derivation enables computation such as logistic regression.

2.1.2 Biophysical Models

One of the earliest, and still the most detailed, mathematical descriptions of neural activity is the Hodgkin-Huxley model of a squid neuron, first presented by AL Hodgkin and AF Huxley in 1952[6]. In this model, the neuron is decomposed like an electrical circuit, and a system of fourth-order differential equations tracks the dynamical relationships between the various chemical channels in the neuron. One satisfying result is that the complete activity of a single neuron can be modeled very accurately without any additional contrivances besides the equations, in contrast to the more abstract

alternatives discussed below.

2.1.3 Integrate-and-Fire Models

A simpler mathematical model, first proposed in 1907 by Lapicque[7], attempts to capture the functional behavior of spiking neurons. Specifically, a neuron can be viewed as an integrator of input spikes, with output spikes occurring when a certain threshold is reached. Lapicque's model, often called the *leaky integrate and fire* (LIF) model, is now the preferred abstraction for large networks of neurons, primarily for its computational tractability.

There are several variants of the model. The simplest version requires only a single variable to store the electrical membrane potential of each neuron. To more closely match neuroscience, the range of this variable is usually constrained to lie between around -75.0 and -35.0, which represents millivolts. As inputs arrive, this variable changes according to an update rule. A logical check occurs at each timestep to see if a fixed voltage threshold has been exceeded. If so, a spike is noted in a binary output stream and the membrane potential is reset to a minimal reset value. Then a short refractory period occurs whereby the neuron is unable to fire for a small time.

The name "leaky" comes from the fact that, in the absence of input activity, the membrane potential naturally decays to the resting value under the influence of a static time constant. Mathematically, this *conductance-based* integrate-and-fire model can be described by Equation (2.1), from [8]. The constant C is derived in terms of a time constant τ and the leakage

conductance g_{leak} to determine how quickly the neuron returns to the resting potential V_{rest}. The I term represents synaptic input activity, which will be discussed later in a more detailed model. V_{th} represents the spike threshold, while V_{reset} represents the value to which the neuron is reset after a spike, often the same as V_{rest}.

$$C\frac{\partial V}{\partial t} = g_{leak}(V_{rest} - V) + I \qquad if(V > V_{th})V = V_{reset} \tag{2.1}$$

All of the models discussed so far are known as "single compartment," since the only active component is the cell body, or soma. The other parts of the model (specifically axons and dendrites) are viewed as passive, a common simplification. To simulate richer dynamical behavior, higher order terms can be added. One possibility is to separately capture inhibitory and excitatory input conductances, as done in [9]. The additional input variables are themselves governed by differential equations which make input spikes into smooth *alpha functions* separately for each input synapse. These variables are then used in conjunction with a membrane update equation, (2.2), a more detailed version of (2.1). Slightly different notation, to match the use of the time variable t, follows [9]. The two summation terms represent the combination of all of the input synapses, effectively replacing I in (2.1). The super-threshold spiking behavior is the same as in (2.1).

$$C\frac{dV(t)}{dt} = \sum g_{ex}(V(t) - V_{exc}) + \sum g_{inh}(V(t) - V_{inh}) + g_{leak}(V(t) - V_{leak}) \tag{2.2}$$

Again, C and g_{leak} are numerical constants including the leakage time constant, and the V_{exc}, V_{inh}, and V_{leak} constants represent the reversal potentials of each type of input and the leakage term. The dynamic variables g_{exc} and g_{inh} follow the spike input, convolved with an alpha function, and including the synaptic weights. The constant values that I used, from [10], are given in the Methods chapter in Table 3.2.

2.1.4 Poisson Point Process

Finally, there is a simple but heavily used method for generation of neural spike output based solely on a desired average rate. The Poisson process, which describes the probability of occurrence of random independent events, has been employed successfully to model neural spike trains. A Poisson process has a single free parameter λ that describes the expected number of events in a given unit of time. In neuroscience, this constant corresponds to the average number of spikes in a fixed time interval, and can be calculated from the expected spike rate and the sampling interval.

2.2 Modeling the Early Visual Pathway

The early visual system, spanning from the retina to the primary visual cortex, has been the focus of intense scrutiny since the 1950s. Many mysteries remain (see especially [11] for an essay on what we still *don't* know about the visual system), but an increasing amount of detail continues to be added to models of this system.

Most of the research follows a stereotypical architecture that was characterized fairly well by Hubel and Weisel in the 1960s[12]. Many computational models utilize some aspects of this architecture, including [9, 13, 3]. I reproduced the results of [10], which is a direct descendant of the detailed simulation given in [13].

2.2.1 Retina

The retina is the first stage of processing in the eye. Some researchers model the actual processing circuitry in the retina, but for the present work computational abstractions are used. The "difference of Gaussians" mathematical model was first proposed in 1966 by Enroth-Cugell and Robson[14], and was popularized by Marr[2].

In this model, two classes of retinal output cells are identified: ON-center cells that respond to bright dots on a black background, and OFF-center cells that respond to dark dots on a bright background. Each of these classes has two characteristic input regions: a center and a surround. The output

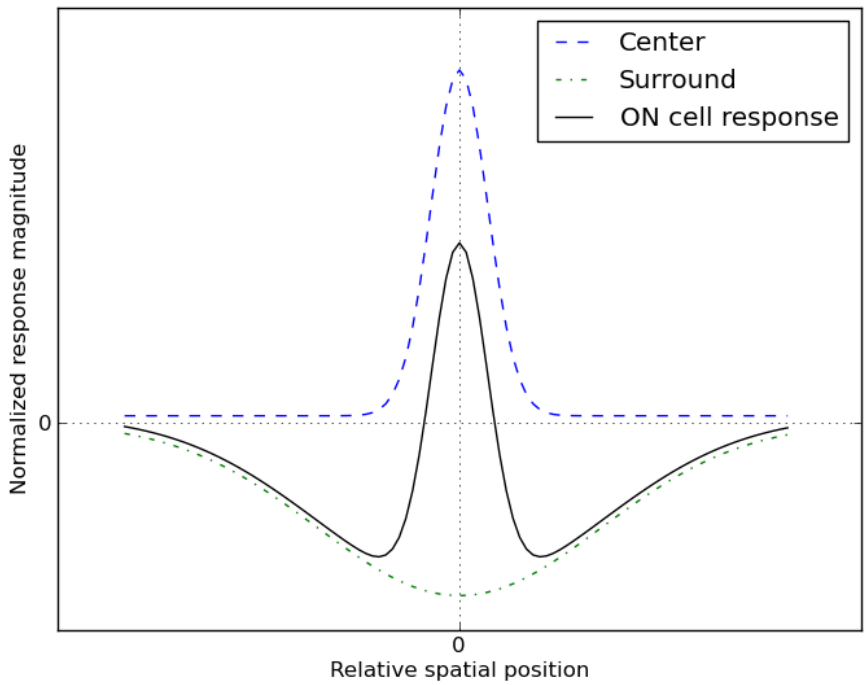

Figure 2.1: Cross-section of circular difference of Gaussians response of retinal ON cell. The ON cell response is modeled by the subtraction of a wide surround Gaussian from a narrow central Gaussian. OFF cells have the inverse response to the ON cell.

behavior of each cell can be described by the difference of two Gaussian filters corresponding to these two regions, as shown in Figure 2.1. An ON cell results from the subtraction of the surround response from the center response, while an OFF cell results from the subtraction of the center response from the center response.

The result of application of these filters is to *whiten* the image. Specifi-

Figure 2.2: Result of filtering image with difference of Gaussian filters. (a) is the original image, (b) is the center response, (c) is the surround response, and (d) is the center response minus the surround response, mimicking the ON cell response.

cally, adjacent pixels are decorrelated, resulting in the accentuation of image discontinuities, which highlights edges and diminishes the response of regions with constant intensity. An example of the result of this filtering on a real image is given in Figure 2.2.

Processing inside the retina has an analog-like flavor, meaning that connected neurons communicate with continuous electrical values rather than the action potentials seen elsewhere in the brain. As such, retinal filtering is often simulated as in traditional image processing, with image intensities being the operant numerical values. The final outputs of the retina are its ON/OFF cells, which fire spikes that travel over the 1.5 million fibers in the optic nerve.

2.2.2 LGN

The retinal volley arrives in the lateral geniculate nucleus of the thalamus, known as the LGN. The cells in the thalamus are often called relay cells, since they seem to act functionally as a "buffer" to the input before it is sent to the primary visual cortex. However, there is processing in the LGN itself, and it accepts input not only from the retina, but also receives a massive backward projection from visual cortex, which is not well understood. Quantitatively, there are more neurons in the LGN than there are retinal fibers, and many computational models either duplicate retinal input at multiple LGN cells[9], or assume a one-to-one correspondence between retinal cells and LGN cells[13]. The present model uses the latter straightforward realization,

without any feedback loops.

2.2.3 Visual Cortex

Processing in the primary visual cortex, called V1, is the primary focus of this work. There are similar cortices for each sensory modality, and many think that understanding cortical processing could be crucial in grasping higher cognitive functions. The current model includes a small subset of V1.

The most widely studied group of cells in V1 constitute the first stage in all visual processing including color, shape, and motion discrimination. These neurons, named "simple-cells" by Hubel and Weisel[12], act like edge-detectors, exhibiting an increase in their firing rate when an edge of the appropriate orientation appears in the small visual region that they are responsive to. There are both excitatory and inhibitory simple cells, with interconnections between the two populations.

The response of simple cells

It was shown in [15] that the response of simple cells can be well described by a Gabor function like that plotted in Figure 2.3. This construction includes a central facilitory ridge surrounded by inhibitory flanks, much like the center-surround cells of the retina but with an angular component. An angled line located exactly on top of the center region provides the maximal response, while a line oriented orthogonal to the central region yields a small response. In general, response is a graded function of the input orientation, with the

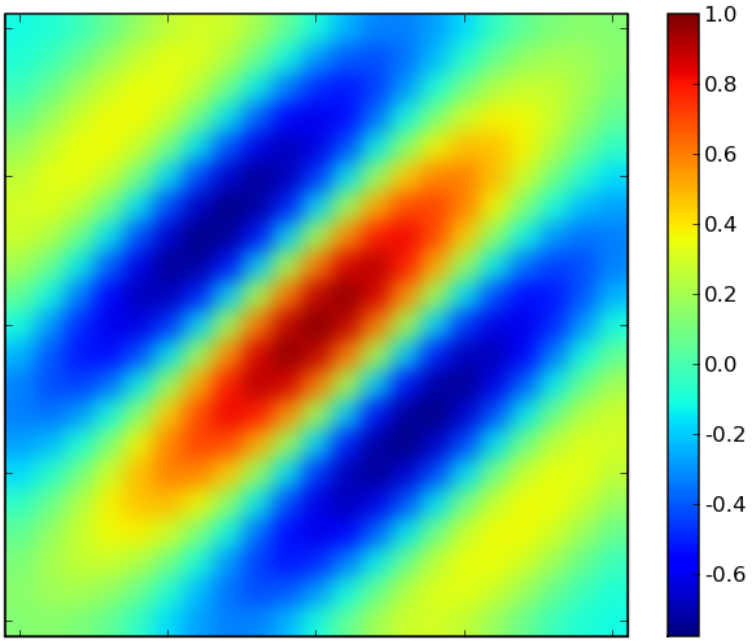

Figure 2.3: Example Gabor function with an orientation ϕ of 45 degrees. Edge detection properties result from the positive region in the center and negative flanking regions.

maximal spike response at the neuron's preferred angle. Some believe that the connectivity between neurons in LGN and V1 is governed by Gabor-like rules[16], and many models, including the one I reproduced, implement this behavior.

Contrast invariance: the iceberg effect

One additional important characteristic of cortical response is the observed phenomena known as the "iceberg effect" of contrast invariance[17], illustrated in Figure 2.4. Due to this property, the output of V1 neurons is relatively insensitive to the intensity of the input, unlike retinal and LGN cells. As shown in Figure 2.4, without this characteristic, the response tuning width is highly dependent on the amplitude of the input, widening and narrowing due to the intensity of the input stimulus. The contrast invariant curves, however, have response tuning widths less sensitive to the input contrast. An intuitive demonstration of this quality is the ability of our visual perception to operate well even in low light situations.

2.3 Neural Coding

This section, which concludes the neuroscience background, describes some of the various coding methods that have been considered in neural computation and specifically visual processing, including notions from signal detection theory and information theory. Several simplifications are used to limit the scope of this review. First, the interest here is in coding single values, and

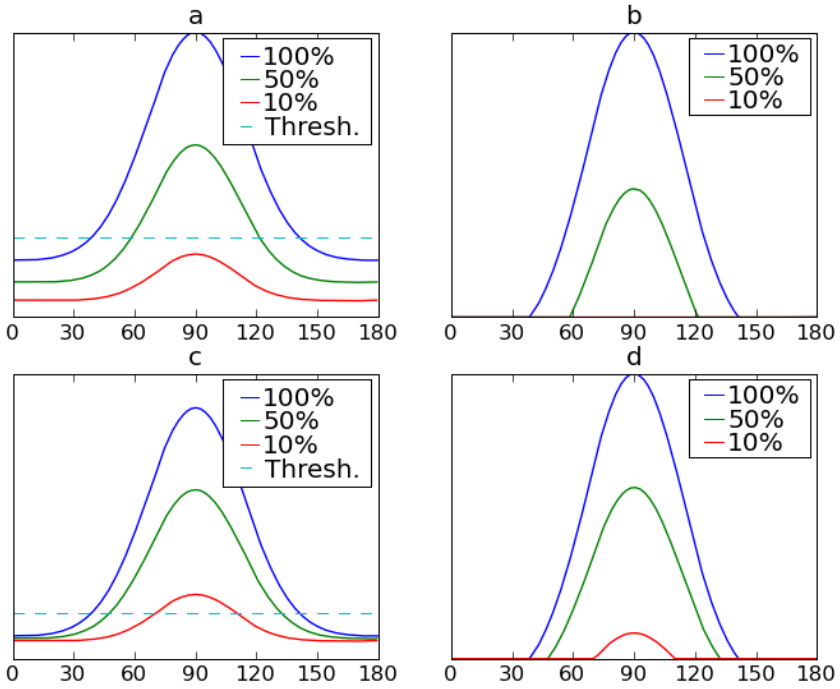

Figure 2.4: Illustration of iceberg effect of contrast invariance. The height of each curve represents the normalized response magnitude of the V1 population. (a) and (c) denote relative population response curves to a variety of input intensities. (b) and (d) show super-threshold response of (a) and (c), respectively. (a) and (b) are not contrast invariant. The thresholded response shown in (b) demonstrates that (a) is highly dependent on input intensity, with zero response at lowest intensity. Conversely, (c) and (d) exhibit contrast invariance, with a nonzero thresholded response at all contrasts.

I invoke the prevalent assumption of independence between neurons. I also eschew sophisticated coding schemes. The technique I focus on uses linear combinations of spike counts, rather than the more sophisticated notions of temporal codes[18] or nonlinear methods[19].

2.3.1 Population Codes

Figure 2.5 illustrates three strategies for encoding a value using multiple units. *Population codes* offer a compromise between the extremes of *intensity coding* (also called rate coding) and *interval coding* (also known as labeled-line coding [20]). The former method uses a single sensor to encode multiple values, and requires individual nodes with few errors and high resolution. Interval coding, on the other hand, uses binary nodes, but needs a large number of reliable units to represent a wide range of values.

The population-based representation, also known as *coarse-coding*[20][21], was first proposed to model the activity of motor neurons in monkeys[22], and has been observed in many neural systems, including sensors in the cricket[23], bat echolocation neurons, and multiple other modalities[20]. In population coding, the combined behavior of an array of graded nodes is used to represent a value, as shown in panel (c) of Figure 2.5. There are advantages of such a scheme, including its resolution and robustness to noise. Since exploration of population coding is a focal point of the thesis, empirical demonstrations of the population coding scheme are given in the following chapters.

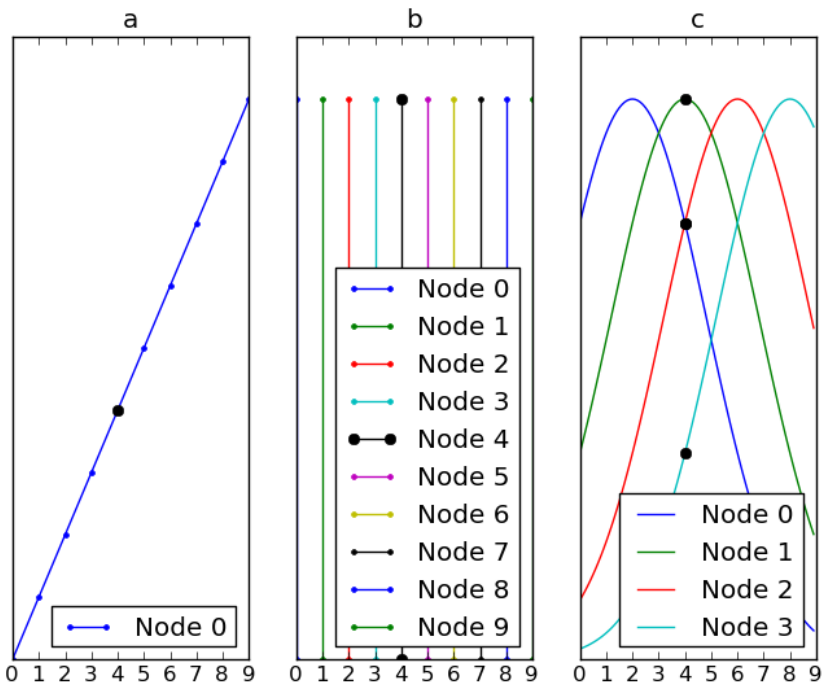

Figure 2.5: Three different encoding strategies. An array of nodes, enumerated in the legends, encodes the value 4. Each node's response curve is shown, with node activity indicated by a filled circle. (a) *intensity coding*: multiple values are encoded by the response of single node. (b) *interval coding*: value is indicated by response of one of many nodes, in "one up" fashion. (c) *population coding*: value is coded by aggregate activity of overlapping responses in several nodes.

An additional step in population decoding is the determination of the coded value given the population response. There are several proposals, ranging from the original notion of *vector coding* or *center of gravity*[24], which uses a linear weighted sum of the sensor outputs, to arbitrary weight vectors learned using perceptrons[25] or maximum likelihood (ML) estimation using kernel fitting[26][27], all of which are explored in the following chapters.

Chapter 3

Methods

In this chapter I describe the mechanics of the spiking neuron simulation. A detailed description of the software is provided, documenting its development, verification, and optimization. Confirmation of the various components was guided by scientific results, which are summarized. Before the detailed material, I first present some general themes of the development approach.

3.1 General Software Topics

3.1.1 Development Cycle

Most theoretical neuroscientists employ the MATLAB environment to write simulations. For applications consisting of operations on vectors and matrices of numbers, its ease-of-use is unparalleled, facilitated by a mature GUI, visualizations tools, and extensive libraries. High performance is not its strong suit, although acceleration is possible through the use of C extensions and some new concurrency facilities. Constructs from traditional computer science are lacking, and some find its licensing model problematic.

To balance my concerns, I used a hybrid development approach. First, I prototyped the system exclusively in Python, using the `pylab` environment. This suite combines several Python numerical packages and an advanced interpreter to achieve MATLAB-like functionality and plotting, albeit without a friendly GUI. When the slow runtime performance of Python became prohibitive, I began to port components to C, and eventually to OpenMP and CUDA. Since I had already constructed the visualization tools in Python (described below), I was able to verify each component of the system individually.

3.1.2 Time-based vs. Event-driven Simulation

When building spiking neuron simulators, there are two main paradigms: *time-based*, or *event-driven*[28]. With the time-based approach, there is a fixed timestep increment; at each iteration processing occurs. Conversely, an event-based model uses dynamic queues to schedule processing on demand. There are several reasons to favor the event-driven approach: the sparsity of spiking may translate to gains in computational efficiency, and greater temporal accuracy is possible, since increasing timing resolution does not impact runtime as in the time-based approach. However, population coding does not rely on fine temporal detail, and the complicated logic to handle event-based simulation is difficult to develop and debug. Furthermore, the processing stream in event-based programming is not as homogeneous as in the case of time-based techniques, minimizing the benefit of SIMD (single instruction

multiple data) parallel architectures. Hence I strictly used the simplistic time-based approach, performing computations at each time iteration.

3.2 Legacy Model Implementation

As stated previously, I reproduced an established model of V1 that owes much to experimental and theoretical neuroscience. A schematic of the network is given in Figure 3.1. The scientific motivation for each component was given in Chapter 2. This section offers a detailed description of the implementation.

3.2.1 Retina

As in biology, the retina is the input entry-point for the model. In the published work I emulated, a 21-by-21 input grid is used. The input takes the form of a standard 256-level grayscale bitmap image file containing a rotated line. To generate the requisite small number of input patterns, I used the GIMP image manipulation tool[29], which provides the means to draw lines and rotate arbitrary angles. See Figure 3.2 for an example stimulus. The GIMP's anti-aliasing filter was crucial to provide the grayscale gradient visible on the edges of the rotated line. With only a binary rendering, it would be impossible to discriminate nearby angles, since their aliased pixels are equivalent.

The first network stage involves applying the difference of Gaussian filters that simulate retinal processing. This was done completely in Python using `scipy`'s built-in two-dimensional convolution operations. For exam-

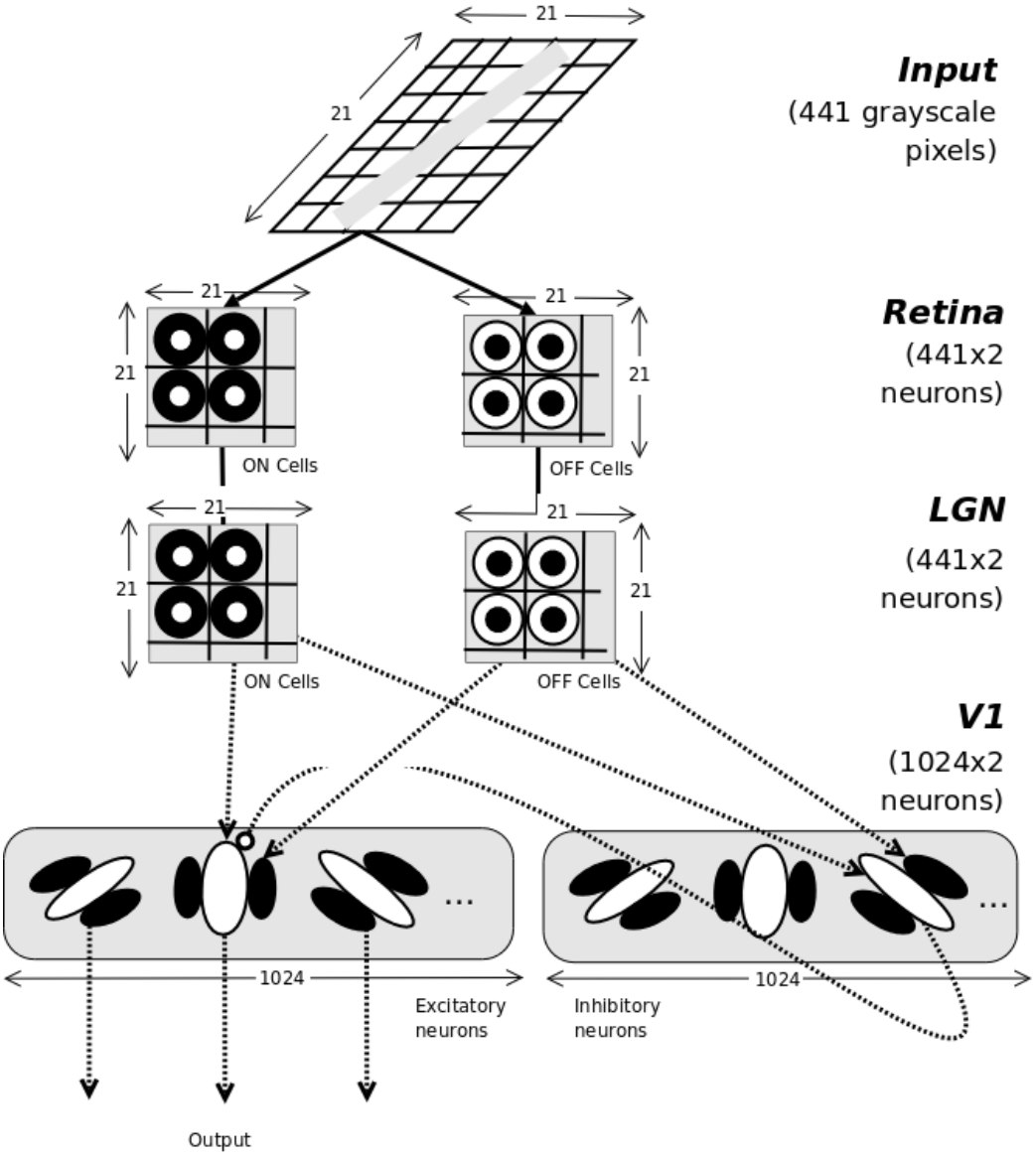

Figure 3.1: Schematic of the network model. Each layer represents a homogeneous population of neurons. Solid lines denote continuous value channels, dotted lines represent spike channels. The circular-headed connection is a lateral inhibitory connection. The input image appears at the retina, and the network response propagates top to bottom, with the final output characterized by spikes from the excitatory neurons at the lower left.

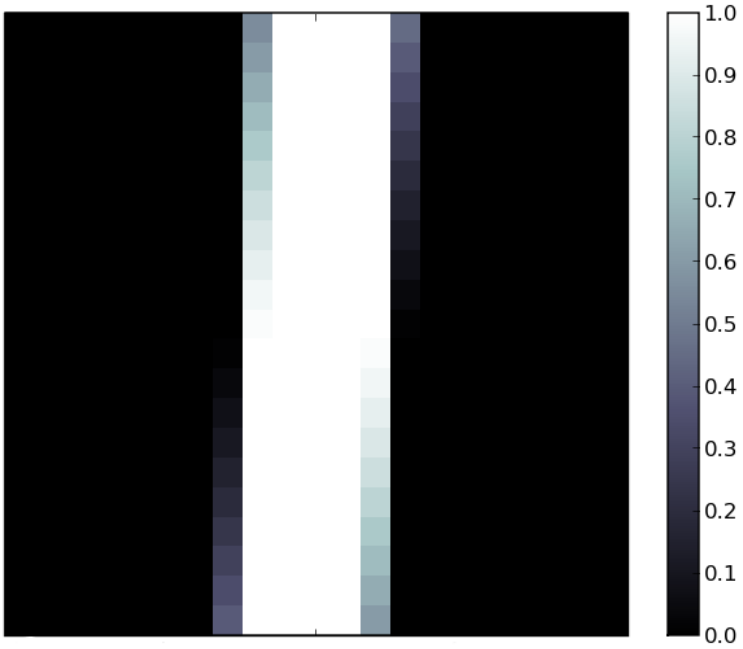

Figure 3.2: Example input stimulus, a 21x21 grayscale image of a vertical bar angled slightly (1 degree clockwise). Normalized pixel intensities, with key shown at right, are used to calculate response of ON and OFF cells, emulating the grid of photon receptors in the retina.

ple, `scipy.signal.convolve2d(image, center_filter)` performs the entire necessary operation for the center spatial filter. Importantly, each filter also has a distinct temporal response, with separate time constants: τ_{center} is 10 (ms), while $\tau_{surround}$ is 20 (ms). The spatial filters are multiplied by the decaying temporal response at each timestep before the two dimensional convolution, resulting in a dynamic overall temporal profile for each ON and OFF cell. To verify this step, I compared the spike rate of the maximally active neurons against the ideal range from the scientific literature. As illustrated in Figure 3.3, the fit is excellent.

The retinal processing was kept in Python throughout development, with the time-dependent rate information written to an intermediate file for use by later stages. This was deemed reasonable since it contains a relatively small number of values (2*1000*441), representing the ON and OFF values at each timestep for all of the 441 (21*21) pixels.

3.2.2 LGN

There is a one-to-one correspondence between neurons in the retina and neurons in the LGN, with the LGN also consisting of both ON and OFF cells. While the retinal layer uses continuous values for each pixel, neurons in the LGN layer generate spikes based on the corresponding retinal value. In model I implemented[10][13], each synapse from LGN to V1 is modeled as an independent Poisson process based on this value. Algorithm 1 describes a Poisson generation algorithm due to Knuth[30]. Per standard practice, I

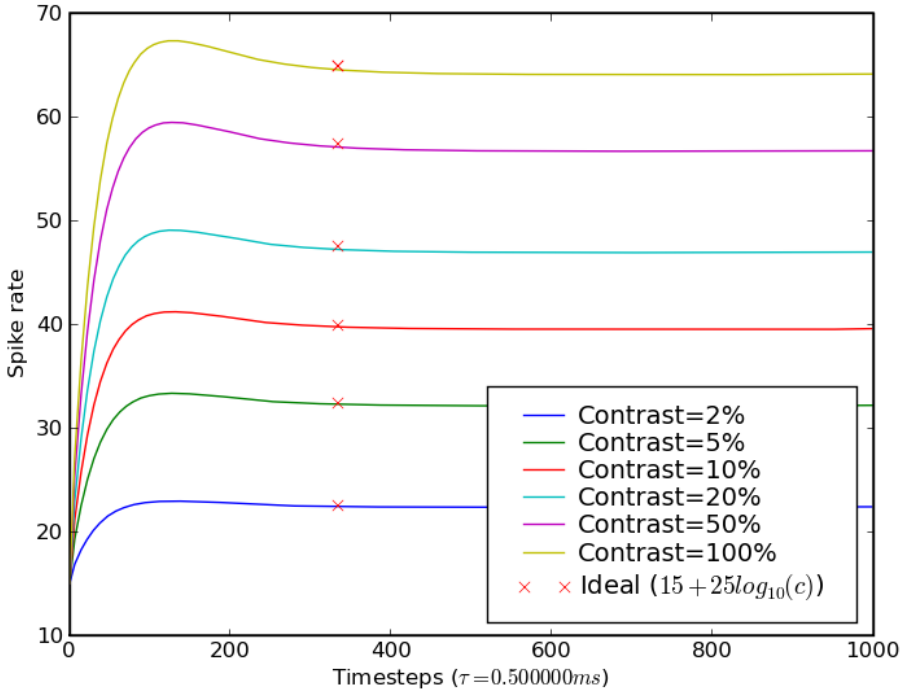

Figure 3.3: Temporal profile of simulated spike rates of a retinal neuron, in response to a range of input image contrasts. Each curve is the response to a different contrast. This cell is spatially located in the center of the oriented line, and has the greatest spike response. Height determines the spike rate, which changes over time due to the temporal activity of the center and surround filter responses. The "x" markers depict the average rate of experimentally observed data for the same contrasts[13], showing good fit to the simulated behavior.

simplified the algorithm to generate a binary event indicator at each timestep. This simplifies the algorithm considerably. Instead of a iterating over possible event counts k in each timestep, all that is required is a single random number and a check against the parameter $e^{-\lambda}$, yielding a true (spike) or false (no spike) result. This does introduce a small deviation from a pure Poisson distribution, which can be mitigated by using a smaller time interval. As an optimization, I pre-computed the exponential $e^{-\lambda}$ for each input rate. λ is given numerically by `neuron firing rate(in Hz)/ 1000.0 * timestep increment (in ms)`.

Interspike intervals (ISIs) are often used to quantify neural spiking. I used this statistic to ensure that the Poisson spike generation mechanism was functioning correctly. The time between successive spikes is measured, and given enough intervals binned in a histogram, a distribution approaching an exponential appears, as shown in Figure 3.4. For validation purposes, the analytically equivalent exponential distribution with mean $1/\lambda$ is also plotted. In more detailed spiking simulations smaller interval sizes are suppressed by the refractory effect, giving the exponential a gentle rise for small values[31].

3.2.3 V1

To emulate the V1 edge detection behavior using the available components of the simulation, the published models select a particular subset of available LGN input neurons for each V1 detector. Each excitatory V1 neuron receives

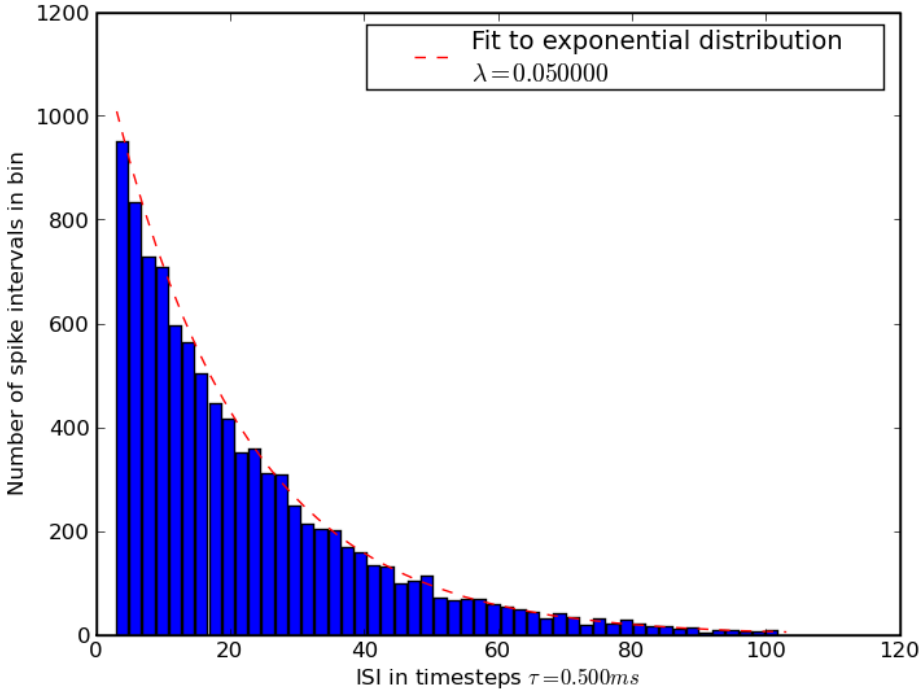

Figure 3.4: Example interspike intervals from Poisson spike generator. A single neuron with a 100 Hz spike rate is simulated for 200,000 half millisecond timesteps. In this time, 9734 spikes are emitted. Interspike intervals (ISIs) measure the time between successive spikes, and are plotted as a histogram. Mathematically, the distribution of times approaches an exponential distribution defined by $e^{-\frac{1}{\lambda}}$, as shown.

Algorithm 1 Poisson generation algorithm, from Knuth[30]

$L \leftarrow e^{-\lambda}$ for the desired rate λ.
$k \leftarrow 0$
$p \leftarrow 1$
repeat
  $k \leftarrow k + 1$
  Generate a uniform random number u in [0,1].
  $p \leftarrow p * u$
until $p \leq L$
return $k - 1$

24 ON and 24 OFF input from LGN, while each inhibitory V1 neuron receives 16 ON and 16 OFF inputs from LGN. The probability of selection, as well as the relative weight of each chosen connection, is dictated by the Gabor function introduced earlier.

The Gabor function that I used, from [10], is described mathematically in Equation (3.1). The values for x and y iterate a grid centered at (0,0), and ϕ is the angle of orientation, which varies between 0 and π over the population of V1 neurons. Equation (3.1) defines a sinusoid, the cos() term, windowed by a two-dimensional Gaussian, the exp() term. Each constant, and its intuitive meaning, is given in Table 3.1.

$$G(x, y, \phi) = e^{-(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2})} cos[2\pi f(xcos\phi - ysin\phi)] \tag{3.1}$$

An example of the outcome of the selection and weighting process is shown in Figure 3.5. These plots show an example connectivity pattern

Table 3.1: Gabor constants

Constant	Value	Meaning
σ_x	1.4	Horizontal extent of exponential window
σ_y	1.4	Vertical extent of exponential window
f	0.5	Spatial frequency: determines number and size of subregions.

between LGN ON and OFF cells and a V1 excitatory neuron preferring 45 degrees. Light pixels denote the location and strength of the connections from the LGN neurons. Dark pixels show the values of the underlying Gabor function distribution used in the probabilistic connection choice. Gray pixels indicate neurons with very low likelihood of connection.

Due to the Gabor connectivity, the spike input to each V1 neuron is implicitly sensitive to the orientation of the image stimulus. This is the essence of the *feedforward* model of orientation tuning. To confirm the proper behavior, I examined the distribution of the spikes from LGN to V1, which has been documented in prior work. Specifically, in [32], the authors plot the total number of LGN spikes influencing each V1 cell in their simulation. The plot shown in Figure 3.6 qualitatively matches these published results.

Each excitatory neuron also has connections for lateral inhibition. There are connections to each V1 excitatory neuron from 30 randomly chosen V1 inhibitory neurons. It is important that the 30 inhibitory neurons are equally distributed throughout the population, rather than having any orientation bias. This mechanism imbues the model with contrast invariance. Greater input intensity causes increased firing in the inhibitory neurons, which then inhibit the excitatory neurons, causing the desired normalization of the ex-

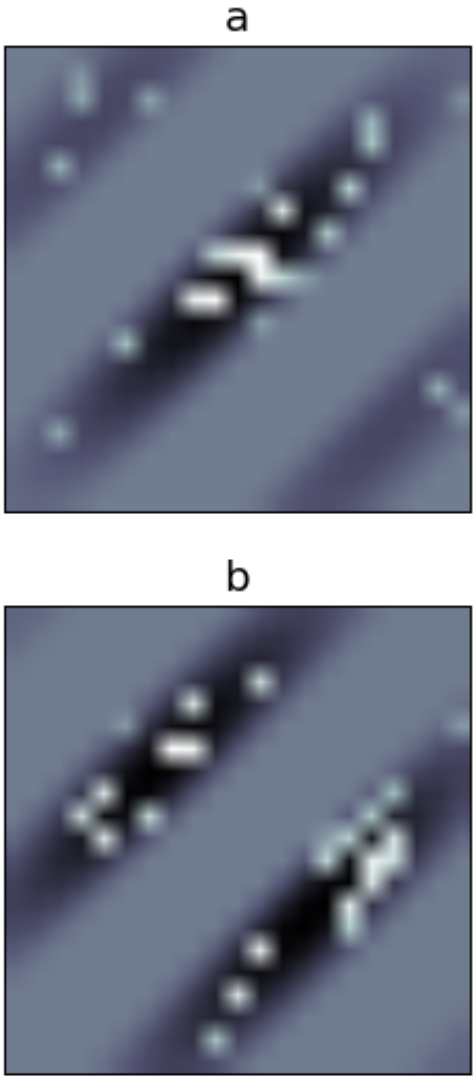

Figure 3.5: Example LGN→V1 connectivity. These figures show LGN neurons randomly chosen to project to a V1 neuron preferring 45 degrees. Each pixel represents an LGN neuron. Light-colored pixels indicate the 24 LGN neurons chosen to project to the V1 neuron, with brighter pixels denoting stronger connection strength. The dark background illustrates the underlying Gabor distribution. Darker pixels denote higher probability of selection, and gray pixels have very low probability of selection. (a) Connections from ON LGN neurons. (b) Connections from OFF LGN neurons.

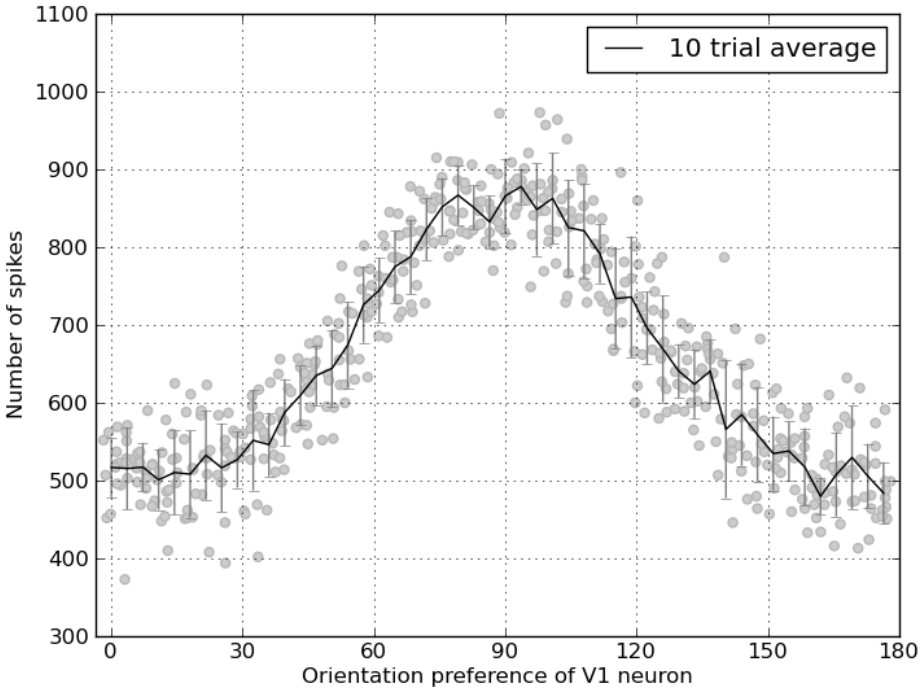

Figure 3.6: Sum of LGN ON and OFF inputs to array of V1 neurons in response to 90° input stimulus. The height of each dot represents the total input spikes to a V1 neuron with angular preference indicated by horizontal position. The 10 trial mean response and standard deviation of each V1 neuron is overlaid on the scatter plot. This figure demonstrates that for the given input stimulus, V1 neurons tuned near 90° receive more LGN input spikes than neurons tuned orthogonally, with a Gaussian-like distribution centered at the orientation of the input stimulus.

citatory response.

As described previously, every synapse has an internal state variable in order to implement an alpha function for smoothly decaying value changes. Figure 3.7 shows the conductance changes of a synapse with particularly active inputs. Ideal alpha functions with a smooth rise and fall require either additional state variables or a memory store of spiking input. Instead, I simplified the realization to have an abrupt rise and natural decay using Equation (3.2). The arrival of input spikes causes an immediate jump in value equal to the weight of the LGN input multiplied by the conductance constant $\bar{g}$ for that synapse. The time constant τ is 1 ms for excitatory synapses and 2 ms for inhibitory synapses. I_t represents synaptic input, taking a value of zero or one at each timestep t.

$$\frac{\partial g}{\partial t} = -exp(-\tau) + \bar{g}\frac{I_t}{\tau} \tag{3.2}$$

All of the synapses for a given V1 neuron are summed as specified by the membrane voltage update equation (2.2). This equation is directly solved using an Euler first-order approximation. The C floating-point expressions in (3.3) show the solution for an excitatory neuron. The constants `V_EXC, V_INH, and V_LEAK` are the voltage reversal potentials, `TAU` is the global timestep interval, `G_LEAK_EXC` is a leakage constant, and `C_EXC` is a scaling term. `gsumE` and `gsumI` variables accumulate all of the synaptic inputs

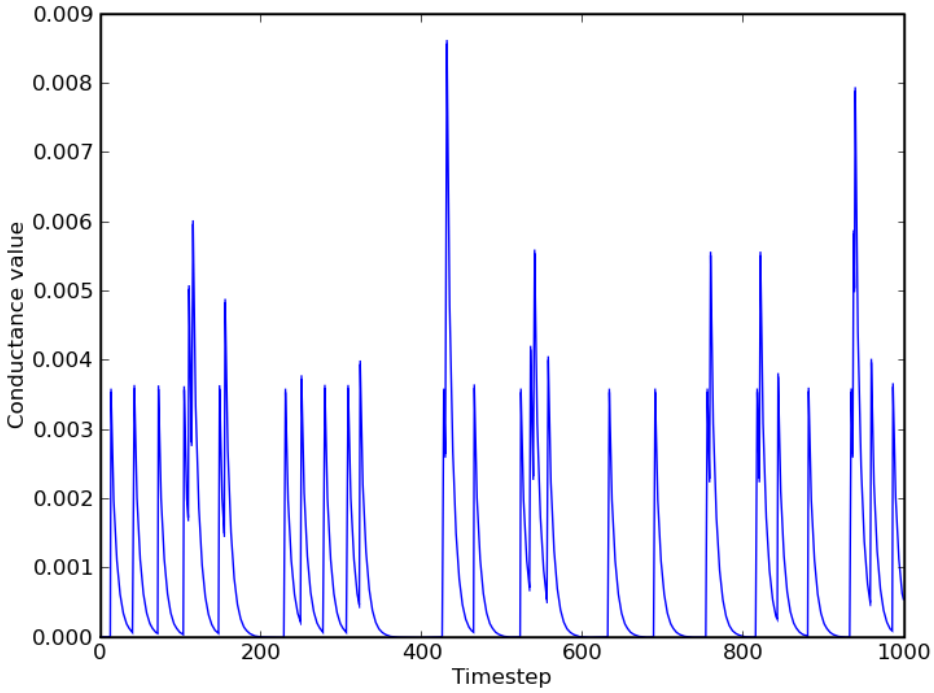

Figure 3.7: Conductance value of a single V1 input synapse over the full simulation lifetime. Each LGN input spike causes an abrupt rise in the unitless conductance value. The value naturally decays to zero by its time constant rate τ, as defined by (3.2).

Table 3.2: Constant parameters in the model

Constant	Value	Meaning
TAU	0.5 ms	Simulation timestep interval
V_EXC	0 mV	Excitatory input reversal potential
V_INH	-70 mV	Inhibitory input reversal potential
V_LEAK	-65 mV	Leakage reversal potential
C_EXC	0.5 nF	Conductance constant (excitatory neurons)
C_INH	0.2 nF	Conductance constant (for inhibitory neurons)
G_LEAK_EXC	25 nS	Leakage constant (excitatory neurons)
G_LEAK_INH	20 nS	Leakage constant (inhibitory neurons)
G_LGN_EXC	4.6 nS	Constant for LGN inputs to excitatory V1 neurons
G_LGN_INH	3.5 nS	Constant for LGN inputs to inhibitory V1 neurons
G_INH_EXC	4.5 nS	Constant for lateral inhibitory inputs to excitatory

governed by (3.2), while V is the actual dynamic membrane variable. See Table 3.2 for the exact constants I used, which are from [10], and are meant to have physiological relevance.

```
dV=gsumI*(V-V_INH)+gsumE*(V-V_EXC)+G_LEAK_EXC*(V-V_LEAK)    (3.3)
V-=TAU*dV/C
```

After the summation of all inputs to a given neuron, the spike condition is tested. This simply involves a comparison of the membrane voltage to a threshold constant. If the threshold is exceeded, the voltage is manually reset to its inhibitory potential, and a binary "1" is noted in the spike output array.

Two example membrane voltages can be seen in Figure 3.8. One V1

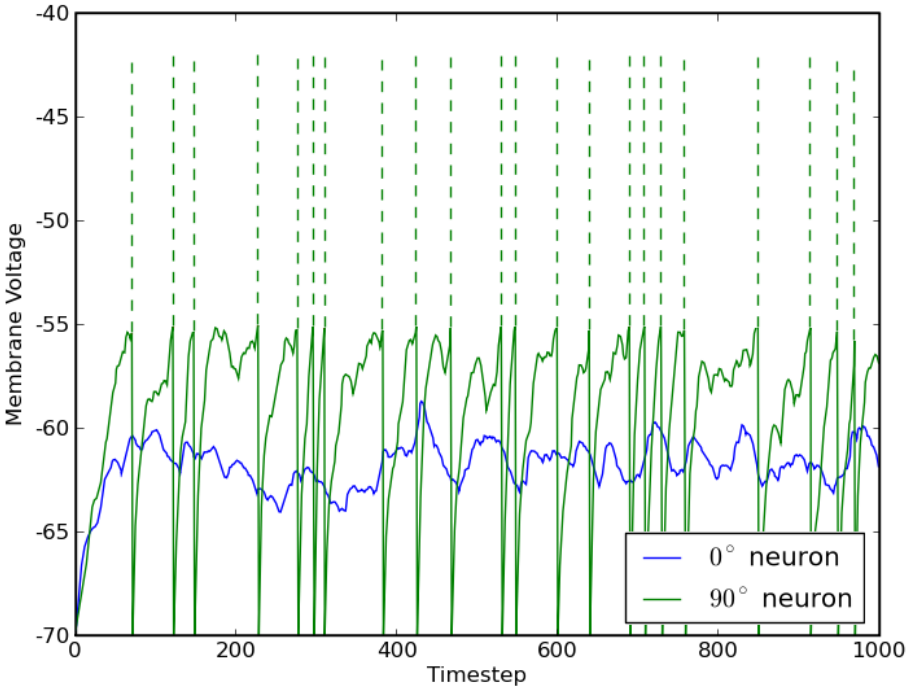

Figure 3.8: Membrane voltage of two V1 neurons, one with orientation preference aligned to the vertical input edge (labeled 90°), and the other one orthogonal to it (labeled 0°). Stimulus input is oriented at 90°. The voltage of each neuron changes over time in response to LGN inputs, lateral inhibitory inputs, and the leakage effect. The vertically-projecting spikes from the 90° neuron are artificially overlaid on the membrane plot as a post-processing step. The 0° neuron does not fire.

neuron, aligned with the vertical bar, is very active, while the other, at the orthogonal horizontal orientation of zero degrees, is less active due to its decreased LGN input. Note that the spikes projecting upward from the trace of the active neuron are simulated, rather than being an emergent property of the differential equations[33]. The apparent voltage jump has been artificially added to the visualization.

3.3 Output Classification

The final step of the model is the extraction of a functional result from the spike train output. Since inhibitory neurons in the brain project locally rather than between disparate regions[34], only the spikes from the excitatory neurons are used in output classification. In the model I implemented, the spikes for each V1 excitatory neuron are counted over a simulation run, and the integer vector of neuron spike counts is used as the final response of the network. Following existing work, I examined several methods of interpreting this noisy response. In this section I describe general techniques for classification and estimation based on an arbitrary vector of values.

Perceptron classifiers are one method for discriminating between two classes of input data vectors. The perceptron method dates to early work in neural networks, but here is used simply to provide a simple binary classifier. With a linearly separable problem, it can provide a weight vector for class discrimination. Specifically, a vector is created with dimensionality of the input plus one for an additional bias term. This weight vector is optimized

using labeled input patterns with Equation (3.4), the perceptron update rule. **W** is the weight vector and lr is a learning rate. The classification is represented as either -1 or +1, based on the sign of the dot product of **W** and **X**. In Equation (3.4), **O** and **T** are the predicted and actual classes, respectively, and **X** are the input patterns with a constant bias term.

$$\mathbf{W} = \mathbf{W} + lr * (\mathbf{T} - \mathbf{O}) * \mathbf{X} \tag{3.4}$$

To further explore orientation estimation from the single-trial spike rate vectors, I implemented two further methods, center of gravity estimation [22][20][24][35] and maximum likelihood (ML) estimation, which for this problem reduces to template matching/curve fitting[35]. Center of gravity is simply a weighted sum of the inputs, as:

$$\hat{\theta} = \frac{\sum_{i=1}^{N} \theta_i (r_i - \gamma)}{\sum_{i=1}^{N} (r_i - \gamma)} \tag{3.5}$$

The observations r_i, are normalized by a constant value γ to prevent bias. Each θ_i takes a value between 0 and π, corresponding to the preferred orientation of the respective V1 neuron.

For ML estimation, I used curve fitting based on the least squares method

of optimization. I used the built-in `scipy.optimize.leastsq` routine provided by the Python scientific library, which implements Levenberg-Marquardt optimization. The optimization kernel was a Gaussian with an arbitrary offset, given in (3.6).

$$\alpha * e^{-\frac{(x-\theta)^2}{(2.0*\sigma)^2}} + \gamma \tag{3.6}$$

By definition, θ, the mean of the Gaussian and the orientation to be estimated, is the center of the kernel, and σ is the usual standard deviation. Two additional parameters are needed to fit the response curve: α defines a magnitude which scales the Gaussian, while γ defines an offset added to each data point, corresponding to spontaneous network activity. I used hard-coded values for σ, α, and γ, while θ was determined using the center of gravity method for the data described earlier.

3.4 Parallel Realization

As I began to scale up the model to thousands of V1 cells, with trials also numbering in the thousands, it became necessary to run on faster hardware. Table 3.3 presents a breakdown of the algorithm, quantifying the loop counts for a problem size of 1024 excitatory and 1024 inhibitory neurons. Step 1 is dictated by the total number of LGN→V1 synapses. Step 2 requires the same number of iterations at Step 1, with the addition of the V1→V1 lateral

Table 3.3: Loop counts with 1024+1024 V1 neurons

Step	Description	Number of elements
1	Poisson spikes	(24+24)*1024+(16+16)*1024
2	Synaptic activity (3.2)	⟨*Step 1 count*⟩+30*1024
3	Neuron synapse summation	1024+1024
4	Neuron update	1024+1024
5	Lateral propagation	30*1024

inhibitory synapses. The two neuron update steps, Step 3 and Step 4, iterate over each neuron. The last step copies inhibitory spikes back into the input stream in preparation for Step 2.

The software was written with clear separation of data between each of these steps. Specifically, the output of Step 1 is a binary array that is consumed by Step 2 to update the dynamic synapse state variables, each of which is independent. Then, the synapses for each neuron are summed. To more easily mitigate shared data contention, the neuron updates, including the summation, are parallelized by V1 neuron instead of by synapse. Finally, spikes emitted from the inhibitory population are fed back into the spike input array needed by Step 2.

With the code structured in this way, parallelization on a shared memory CPU platform is clear-cut, especially with the help of the high-level abstractions provided by OpenMP. For example, the following code snippet, from the neuron update in Step 5, utilizes multiple cores on architectures which support OpenMP, including the recent version of Ubuntu Linux I used.

. . .

```
    #pragma omp parallel default(shared)
    #pragma omp for schedule(static)
    for(neuron=0; neuron<NUM_NEURONS; neuron++) {
        dV[neuron] = G_LEAK_EXC*(V[neuron] - V_LEAK)
        V[neuron] -= TAU*dV[neuron]/C_EXC;
        if (V[neuron] > threshold)
    ...
```

Parallelization of the `for` loop over neurons is automatic, and an implicit *join* operation ensures synchronization after loop completion, which is critical for the given implementation. The number of threads to use for processing is specified on the command-line using the environment variable OMP_NUM_THREADS. The ease of this parallelization method is aided by the independence of each loop iteration/neuron.

The transition from an algorithm in this form to a version for the CUDA architecture is straightforward. Since CUDA has been designed for operations on vectors of data, modifying the code above requires expressing it as a CUDA "kernel" with the appropriate parameters. The following code snippet demonstrates what such a kernel looks like:

```
__global__ void neuron_kernelU(
    float *device_dV, float *device_V,
    ...)
{
```

```
const int      tid = blockDim.x * blockIdx.x + threadIdx.x;
const int THREAD_N = blockDim.x * gridDim.x;

for(int iLine = tid; iLine < THREAD_COUNT; iLine += THREAD_N){
    for(int iRow = 0; iRow < PER_THREAD; iRow++){
        int neuron=iRng + iOut * THREAD_COUNT;
        device_dV[neuron] = G_LEAK_EXC *(device_V[neuron]-V_LEAK);
        device_V[neuron] -= TAU*cuda_neuron_dV[neuron]/C_EXC;

        if (device_V[neuron] > threshold) {
....
```

With CUDA, this kernel is executed simultaneously on multiple threads distributed throughout the many cores of the GPU. CUDA has a very elaborate multi-tiered architecture for problem decomposition[36] that will not be discussed in this thesis. Instead, this exposition will focus strictly on how the neuronal network algorithm was mapped to the CUDA platform using the given code fragment as an example.

The first line of the function converts CUDA internal instance variables, `blockIdx` and `threadIdx`, into a single value suitable for use as a unique identifier. The second line determines the number of neurons each thread will process, which is on the order of 1-10 for the scales I looked at. The outer loop permits iteration over a two-dimensional index, and is not relevant for the neuron loop, since for the problem sizes examined, it is sufficient to

simply have a large set of threads (512), each processing several neurons using the inner loop. Larger sizes require decomposing the data into finer pieces.

Kernels are downloaded and executed by a function call in the C code running on the host. Generally execution is asynchronous to allow overlapping computation between the CPU and the GPU, but for my implementation, which ran almost entirely on the GPU, there was no performance advantage to utilizing asynchronous operation. The breakdown of the algorithm into the structure of Table 3.3 was crucial, since CUDA does not yet provide finely grained synchronization constructs. Instead, the separate kernels of each function as implicit join points.

Moving data onto and off of the card is a time-consuming operation. Therefore, my realization depended on the data remaining resident on the GPU throughout the simulation. The only data on the CPU are the LGN spike rates. At each timestep, this small amount of data (2*441) is transmitted to the GPU. This explains why the variables in the code snippet above have the prefix `device_`: in my implementation this prefix indicates that they reference memory on the GPU itself. As usual when programming in C, memory management is a manual operation.

I architected the system to facilitate execution with either CUDA or OpenMP based on compiler directives. I did this both to allow the incremental debugging of components, and to support comparative performance profiling. Cross-platform profiling used the standard Linux `gettimeofday()`

routine from `sys/time.h`, which, on the systems I tested, gave a resolution of microseconds. CUDA also provides a hardware timer, which I used as a validation of `gettimeofday()` and to provide additional profiling of the CUDA-only portions.

Chapter 4

Results

In this chapter I present the results of my experiments with the simulation. First, I summarize the functional behavior of the model, including reproduction of recent neuroscience research. Then I discuss the runtime performance of my implementation.

4.1 Reproduction of V1 Model

First, to ensure the accuracy of the simulation, I chose to precisely duplicate the results of existing work. I began with the model of [10], which itself owes much to previous work, summarized in Chapter 2. Note that my goals are somewhat different than in traditional neuroscience research. Often, the objective is to include copious biological detail and ensure that behavior fits experimental results. I duplicated these models just to prove the functional operation, and although striving to not violate scientific findings, I was less concerned with many of the biological parameters.

4.2 Output of V1 Excitatory Neuron Population

The fundamental output component of the model is given by the spike counts of the array of excitatory V1 neurons. The total number of spikes emitted by each V1 excitatory neuron over the timesteps of one trial are summed, yielding a vector of integral spike counts. A representative single-trial sample, with 1024 V1 neurons and 1000 half millisecond timesteps, is shown in Figure 4.1. This is the "population response" of the V1 neurons when presented with a 90 degree bar. Each point along the horizontal axis represents a single neuron, with its spike count given by the height of the sample. To more clearly see the shape of the distribution, more samples are needed. Figure 4.2 shows ten trials plotted together, overlaid with the average value and standard deviation at each detector. The mean of the whole population is shown by the horizontal line. The results shown are a good fit to the plots shown in Figure 1B of [10] and Figure 1B of [32].

Since contrast invariance is typically a crucial component of these models, I confirmed that my simulation had this desired quality by running trials with identical stimuli at a range of contrasts, with the results shown in the left panel of Figure 4.3. The contrast invariance is due to the lateral connections from the inhibitory neurons. Repeating the trials, but with the lateral connections disabled, yields the more linear, and thus contrast dependent, shape of the right panel of Figure 4.3. The plots were generated with the same method as the previous plots, by averaging ten trials at each

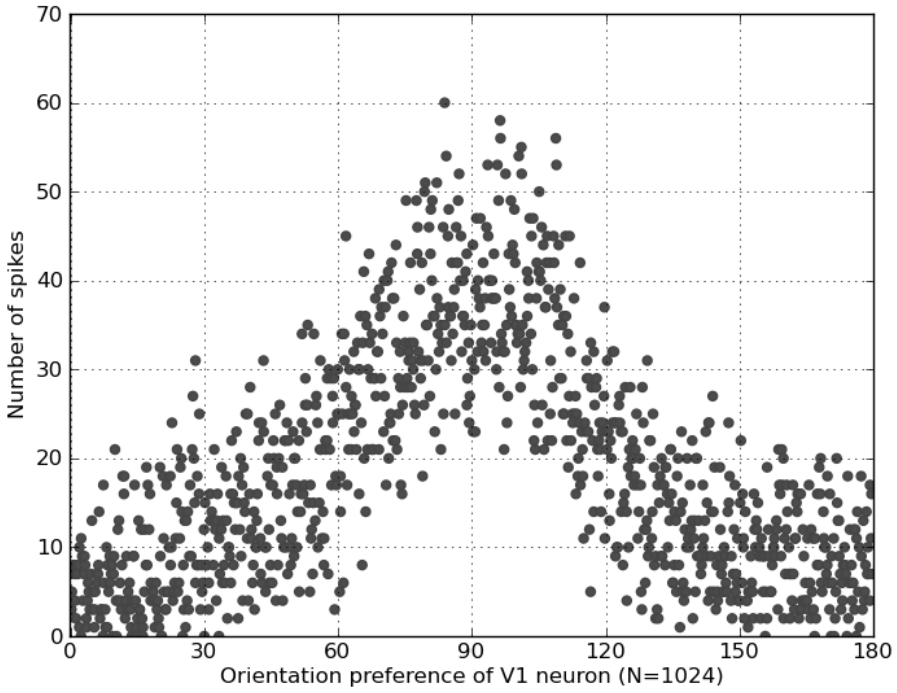

Figure 4.1: Output of population of V1 excitatory neurons in response to a vertical bar. Spike count is totaled over duration of run, 1000 half millisecond timesteps. Each dot represents the total number of spikes emitted by one of the 1024 neurons over the timesteps of a single trial, with preferred orientation indicated by location on horizontal axis and output spike count given by vertical height. The neurons are spaced approximately 0.176 degrees apart. Neurons with orientations near the stimulus orientation are the most highly active, on average.

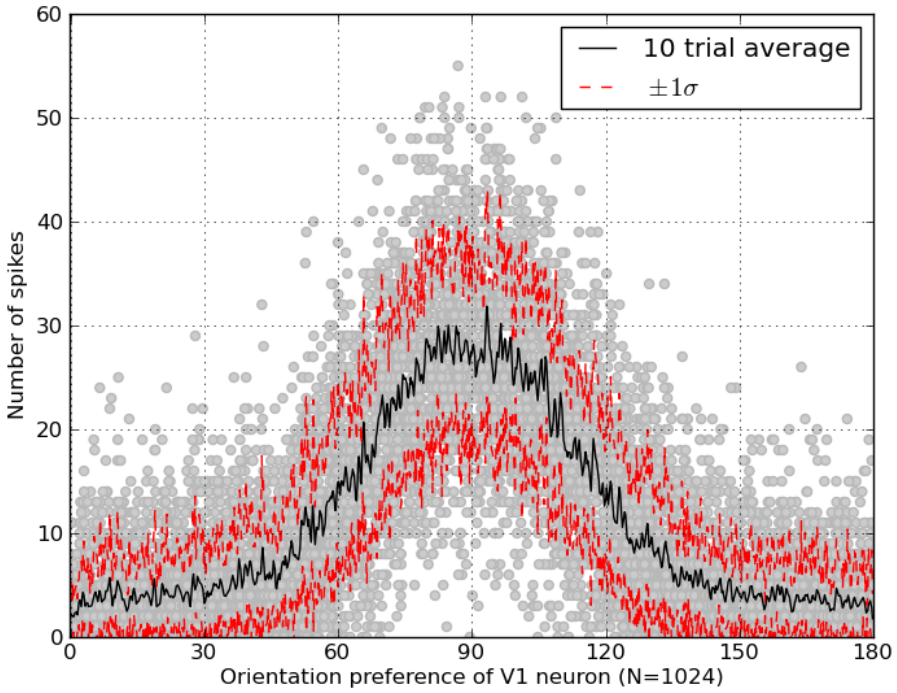

Figure 4.2: Output of the 1024 V1 excitatory neurons over ten trials. Scatterplot of points indicates individual samples, as in Figure 4.1. The 10 trial average response of each V1 neuron is overlaid. Dashed lines indicate standard deviation over the 10 trials.

orientation, with additional smoothing by low-pass filtering.

4.3 Tuning Curves

The tuning curves of four V1 neurons are shown in Figure 4.4. These plots are in in essence the dual of the population response. Whereas population curves show the response of all elements of the population to a a single stimuli, tuning curves show the response of a single (or small number) of V1 neurons to a range of stimuli.

To create these plots I generated stimuli at eighteen orientations. These were the same bar image rotated at orientations between 0 and 180 degrees, spaced ten degrees apart. I presented each image to the network for 100 trials, and calculated the average spike output response of each detector. As expected, the empirical tuning curves do not have exactly the regular Gaussian shape of the theoretical models. It is important to remember that these tuning curves are an emergent property of the underlying Gabor connectivity, and are subject to pixel aliasing and noise due to the Poisson input.

4.4 Orientation Discrimination

In order to interpret the behavior of the spiking network, the spike count output must be post-processed. Several proposed methods were described in Section 2.3 and Section 3.3. I studied the empirical performance of three methods: a perceptron for binary discrimination of two nearby orientations,

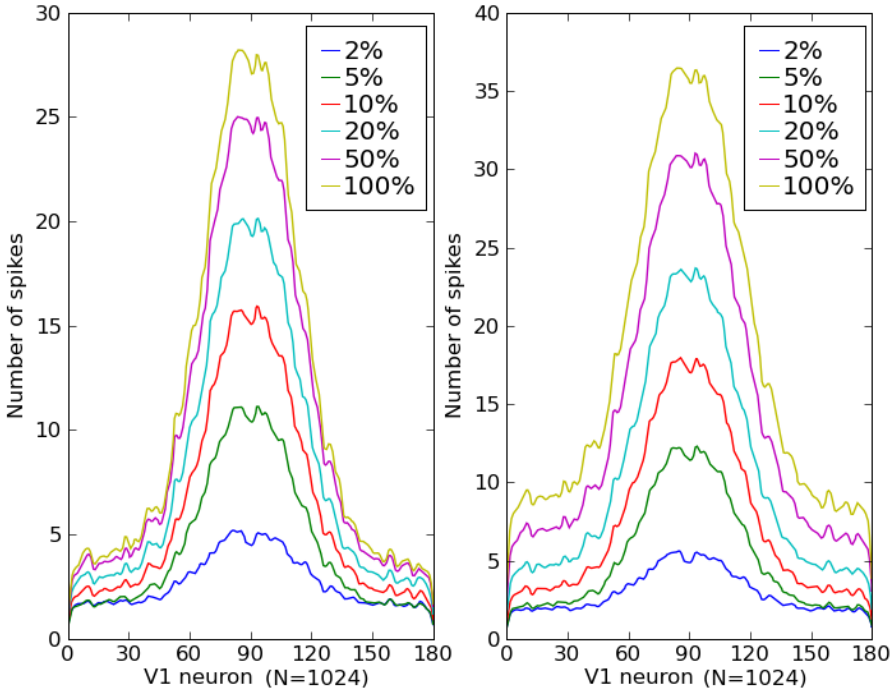

Figure 4.3: Network response to a range of stimuli with differing pixel intensities, demonstrating contrast invariance. Left panel shows contrast invariant network response. Right panel show network response with inhibitory connections disabled, with contrast sensitive response. Note especially the marked difference between the tails of the curves between the left and right plot. Compare to Figure 2.4

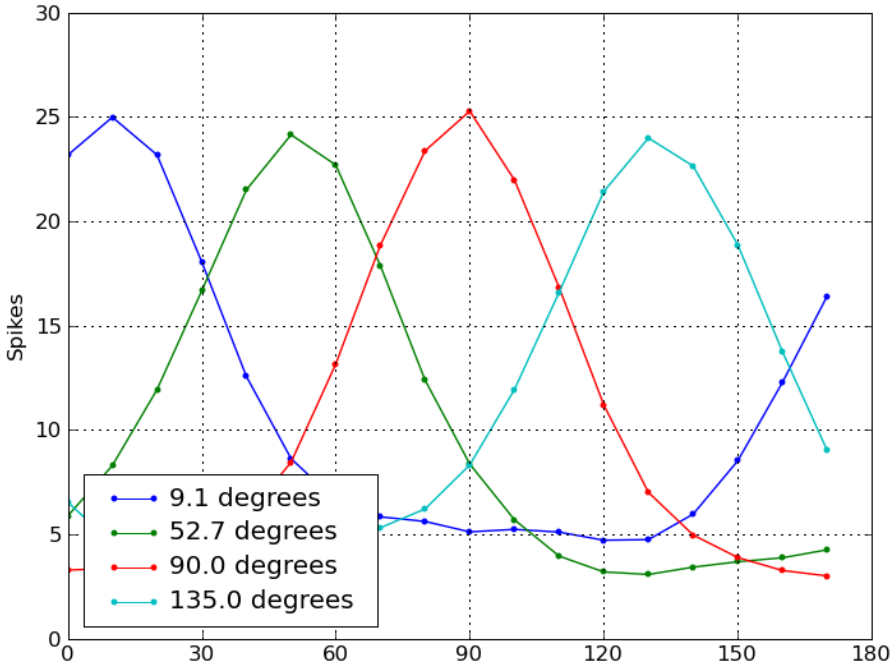

Figure 4.4: Tuning curves: Each curve plots the average response (over 100 trials) of four V1 neurons to 18 different input patterns. The input stimuli are edges oriented between 0 and 180 degrees, equally spaced 10 degrees apart.

and both the center of gravity and maximum likelihood approaches to estimation of arbitrary orientations. Previous work has suggested how the latter two techniques could be implemented with neural circuitry[37], but to limit the scope of my thesis I use straightforward analytical methods.

4.4.1 Perceptron Binary Classifier

Much of the literature concerning these models studies discrimination between two nearby orientations using the output of the V1 neuron array. Specifically, in the early model of [38], psychophysics results [39] are referenced that found that humans can reliably discriminate an orientation angle of around 0.4 degrees. Here "reliably" is defined to mean 75% percent of the time. Much of the work since then [10][32] has continued to use this task, although with differing quantitative results.

I trained a single-layer perceptron binary classifier based on the spike count output. My first attempt, following [10], used the vector of 1024 spike counts to discriminate between two degrees in orientation, using an edge oriented at 89 degrees and one at 91 degrees. I generated 1024 trials at both orientations, then divided each set of trials into two subsets of 768 trials and 256 trials, for training sets and test sets, respectively.

The results are given in Figure 4.5, which shows the increase in testing accuracy with increasing number of training exemplars. The maximum accuracy achieved is shown by the pair of numbers above the highest point. For this set of runs, with this particular learning sequence, the network achieved

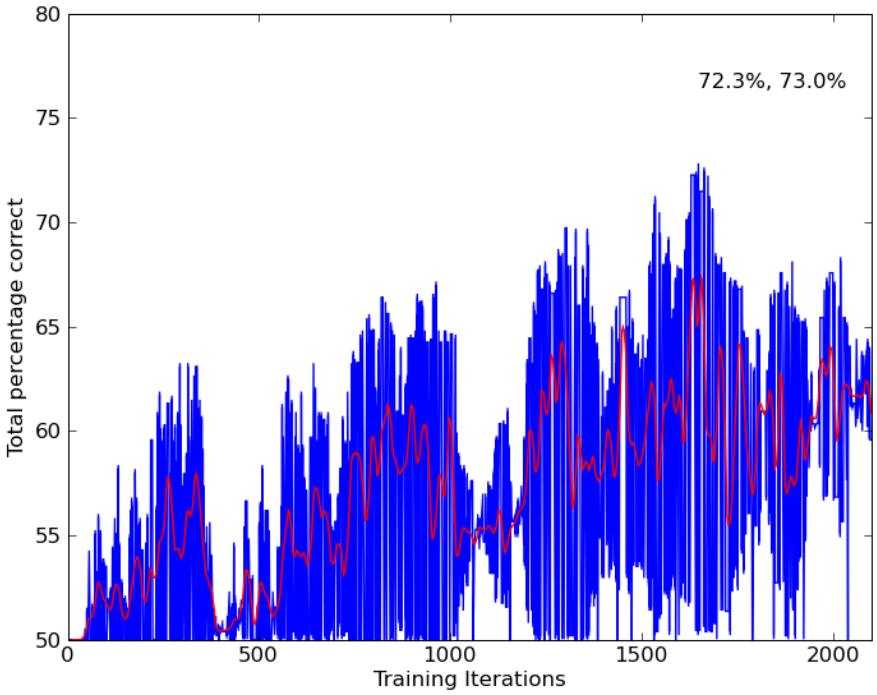

Figure 4.5: Performance of the perceptron classifier on the test set, measured by total percent correct with increasing number of iterations. A low-pass filtered realization is overlaid on the raw samples. The pair of numbers shows the percentage correct in the two orientations at the best iteration.

72.3% accuracy for one angle and 73.0% accuracy for the other, corresponding to 72.65% total accuracy.

The resultant learned weights corresponding to each V1 excitatory neuron, shown in Figure 4.6, have a characteristic pattern. The shape, most evident in the low-pass filtered signal, shows a similarity to Figure 3 of [25] and Figure 6 of [35]. The low weights near 90 degrees are due to the high

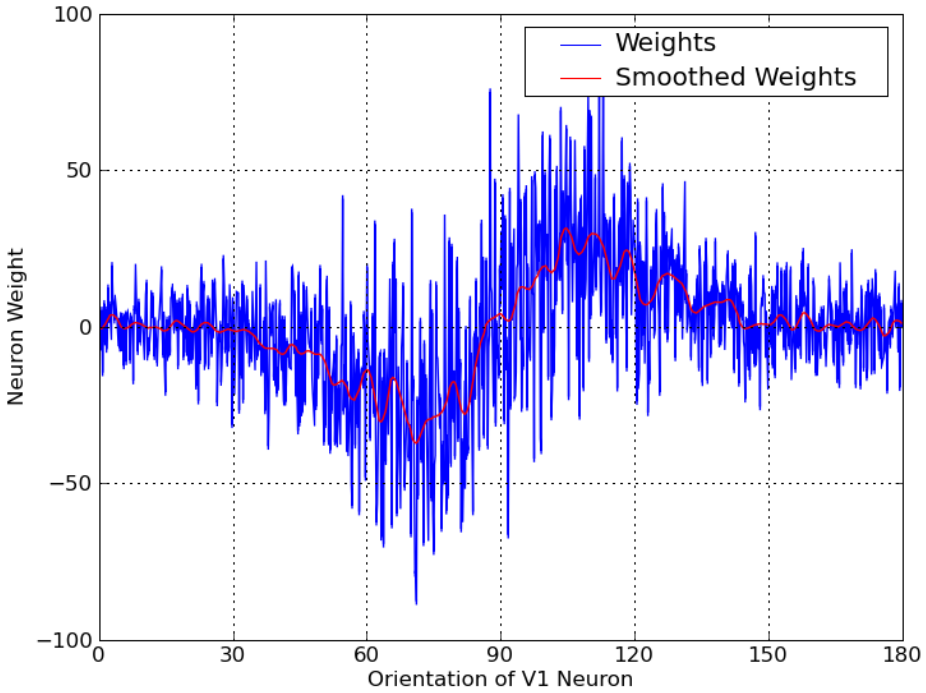

Figure 4.6: Weights of perceptron for discrimination between two nearby orientations. Values correspond to weighting of the spike count for each V1 neuron. The characteristic shape is due to the changing variance of the underlying population response, which is maximal near the peak of the Gaussian population curve. Smoothing was accomplished by low-pass filtering the data.

variance of both detectors at the peak of their Gaussian preference curve, which diminishes the information available from those detectors near the peak. The sinusoidal shape of the weight vector has extrema where there is maximal information about the input stimulus, at either side of the underlying Gaussian population response curve, diminishing to zero as the bell curve of the population response falls off.

4.4.2 Arbitrary Orientation Estimation

Next, I explored the ability to estimate an arbitrary orientation using the spike counts from a single trial. First, I applied the center of gravity technique as specified in Equation (3.5), comparing several network sizes. Figure 4.7 and Figure 4.8 depict the results of performing the center of gravity estimate over multiple trials for networks of size 512, 1024, and 2048 excitatory neurons, at a range of possible orientations. For all three networks, equal sized populations of excitatory and inhibitory neurons were used. Orientations from 0 to 170 degrees, equally spaced at 10 degrees, are presented to each of the networks for 100 separate trials. The mean is estimated using center of gravity weighting of the spike counts. The plots show the deviation of the estimate from the actual value, in degrees, along with the standard deviation of the estimator for each angle.

This estimator performs well only near the center of the orientation range, since boundary effects dominate at the upper and lower parts of the orientation range. The mathematical construction is unable to handle the circular

nature of the angular preference, leading to overestimation of the orientation when the actual angle is less than 90 degrees and underestimation when the actual angle is greater than 90 degrees. The two larger networks provide fairly good estimates (within 1 degree, having 2 degrees standard deviation) near 90 degrees. The 512 neuron network has large variance, although it does provide a good estimate at 90 degrees. In general, variance is inversely proportional to network size.

Next, using the same spike count data, I fit a Gaussian kernel using the `scipy` library's least-squares fit routine, which, in this context, is a form of maximum likelihood (ML) estimation. An example is shown in Figure 4.9. Note that the Gaussian curve is an estimated fit from a single sample, rather than a multiple trial average. Since starting parameters are required for the Levenberg-Marquardt iterative method, I hardcoded reasonable values for the height, width, and offset of the Gaussian of Equation (3.6), and used the center of gravity estimate to "guess" the initial parameters.

Figure 4.10 shows the results of this estimation process for the input stimuli, network sizes, and trials described previously. Due to the dependence on the center of gravity estimation, boundary effects are evident for this estimator as well, but not as egregious as the raw center of gravity estimator itself. Particularly with a greater number of neurons, the range of reasonable estimates is much greater, since the curve fitting algorithm is able to recover from the poor initial estimate. For input stimuli between 50 and 140 degrees, all network sizes yielded good performance, well within half a degree of the

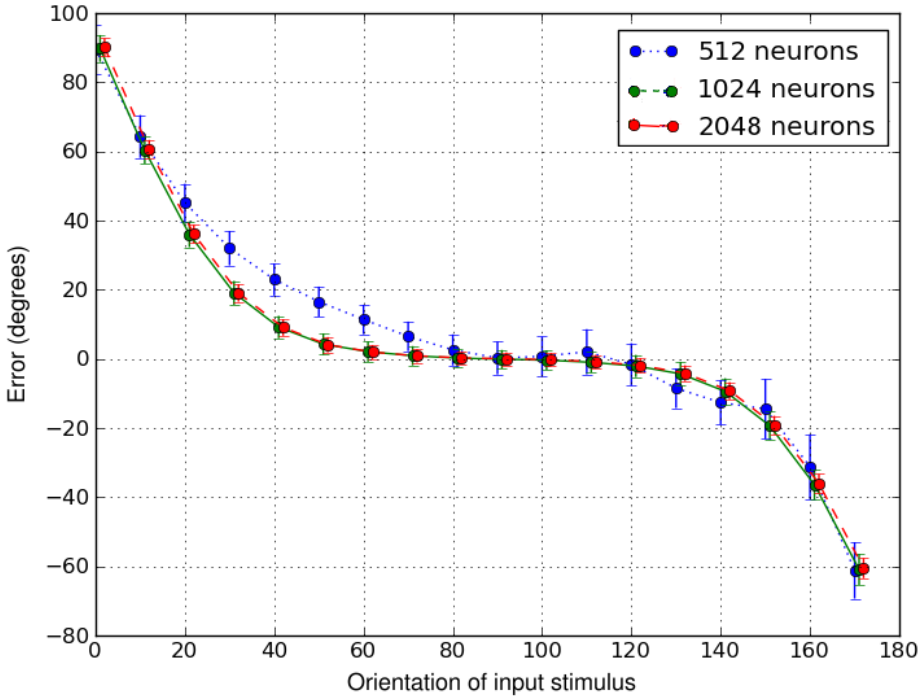

Figure 4.7: Results of the center of gravity estimates for three excitatory V1 neuron population sizes at a variety of input stimulus orientations. 100 trials at each of 18 orientations (shown on the horizontal axis) are presented to the network, and the resultant spike counts are used for single trial center of gravity estimation. The average error in orientation estimation over the 100 trials is plotted on the vertical axis. Error bars represent standard deviation from the mean at each input stimulus. Figure 4.8 shows greater magnification.

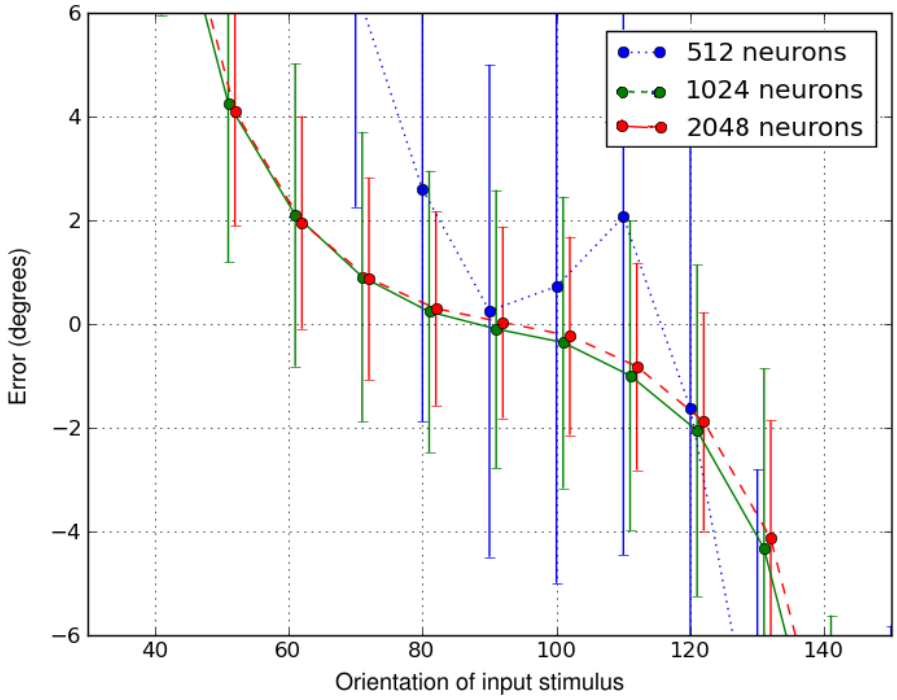

Figure 4.8: Detail from Figure 4.7. See Figure 4.7 caption and text for details.

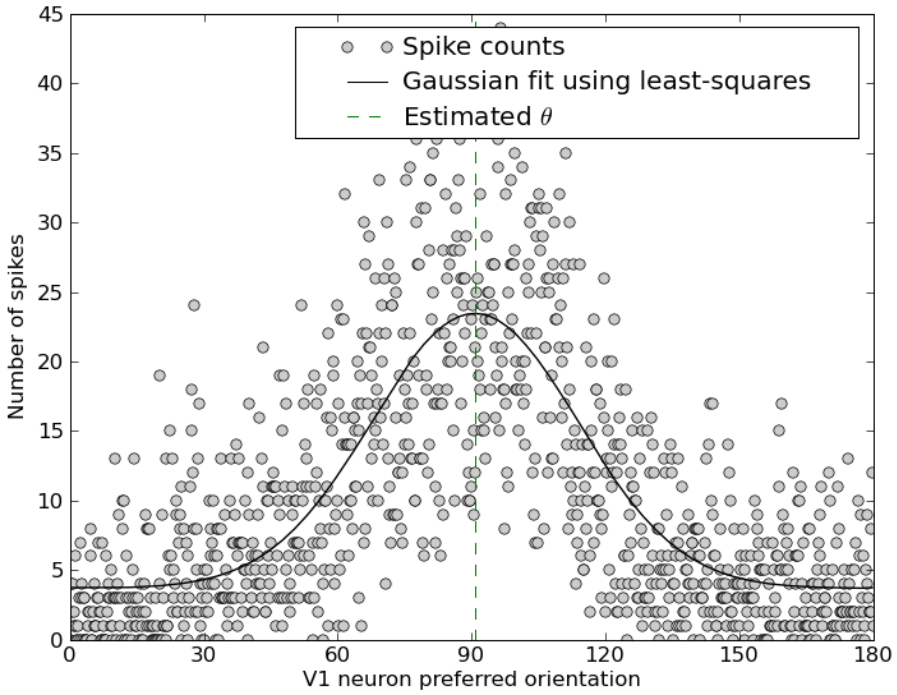

Figure 4.9: Results of performing least-squares optimization of a Gaussian kernel with the spike counts of 2048 excitatory neurons responding to presentation of a vertical bar. Estimated orientation, as indicated by dashed line, is 90.65 degrees.

true estimate, with variance inversely proportional to the number of neurons. Note the much smaller scale of Figure 4.10 compared to Figure 4.8. The variance of the ML estimator was about half that of the center of gravity estimator.

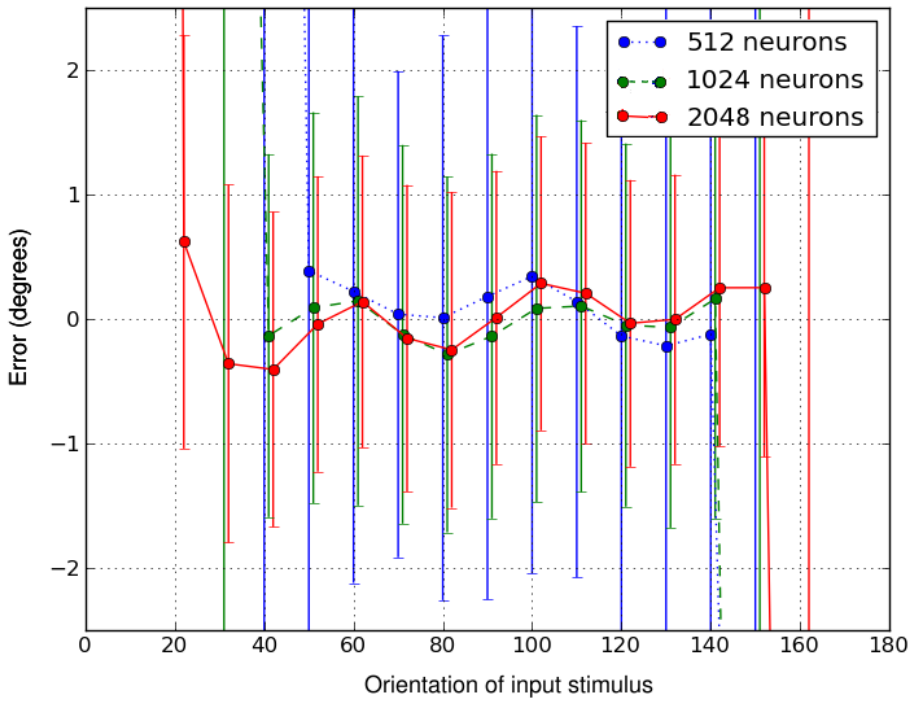

Figure 4.10: Results of the maximum likelihood (ML) estimation, using least-squares optimization of Gaussian kernel. 100 trials at each of 18 orientations stimulus orientations are executed for three different V1 population sizes: 512 excitatory neurons, 1024 excitatory neurons, and 2048 excitatory neurons. The trials are averaged, and the mean and standard deviation are plotted for each network size at each orientation.

4.5 Parallelization Results

To facilitate extensive experimentation, a significant effort was spent parallelizing the code. With large V1 populations and multiple trial runs, this effort became indispensable. Once the results were consistent across architectures, I performed cross platform profiling, summarized in Figure 4.11.

The CPU measurements were performed on an Ubuntu Linux workstation containing two Intel Core i7 920 CPUs, providing a total of eight cores. The 8GB of memory was sufficient to contain the data structures used by the code for all network sizes I tested. Both CPU versions are identically compiled with gcc-4.3 using default optimization options and OpenMP capability enabled. For profiling, concurrency is constrained using the `OMP_NUM_THREADS` environment variable.

I tested the CUDA version on two NVIDIA PCI cards, an NVIDIA GeForce 8600 GTS, and an NVIDIA Tesla 1060C. The GeForce card is a mid-range consumer graphics card with 256M of memory and 32 processing cores. The Tesla card is a high-end card designed for parallel computation with 4GB of memory and 240 processing cores. Each core on the Tesla runs at 1.3 Ghz, while each core on the GeForce runs at 1.46 Ghz. Both GPU cards were housed in desktops running Ubuntu Linux.

The total runtimes in Figure 4.12 are divided into algorithm components, which map directly to those in Table 3.3. The portion labeled "Poisson Spike Generation" corresponds to Step 1 of Table 3.3, while the "Synaptic

Conductances" segment corresponds to Step 2. "Neuron Update" consists of the remaining steps, Steps 3, 4, and 5. The results are discussed further in the next chapter, but there are a few general observations to be made. First, the runtime of the Poisson generation code on the CPU *increased* as more threads were added, and in general this portion of the code did not scale as well as the other components, particularly the synaptic conductance updates. The neuron loop did scale, but not as well as the synapse portion.

I ran additional experiments on the Tesla to further quantify the scaling performance of the CUDA version of the algorithm, summarized in Fiqure 4.12. In this plot the five steps from Table 3.3 are profiled separately. I averaged multiple trials for four problem sizes: 512 excitatory neurons, 1024 excitatory neurons, excitatory neurons, and 4096 excitatory neurons. Generally, the performance scaled linearly with the network size. For the Poisson portion of the code, however, performance was flat until 4096 neurons.

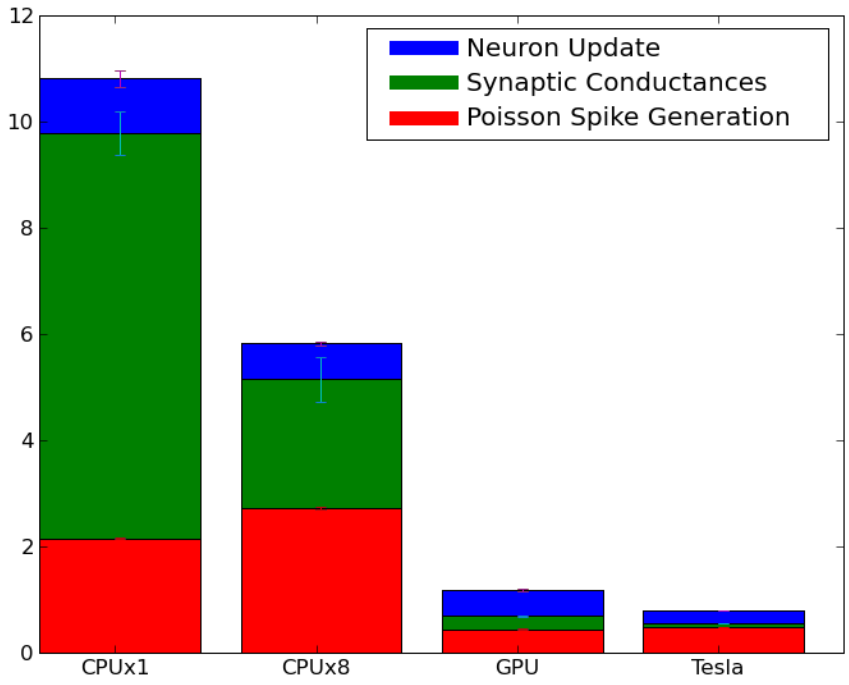

Figure 4.11: Average single-trial runtimes (in seconds), for population size of 1024 V1 excitatory neurons for a variety of architectures, broken down by algorithm component. CPU runtimes are from OpenMP version on 8-core desktop, with `OMP_NUM_THREADS=1` and `OMP_NUM_THREADS=8`, respectively. GPU denotes an NVIDIA GeForce 8600 GTS, and Tesla indicated a NVIDIA Tesla 1060C. Error bars denote a standard deviation in runtime, which was negligible for all architectures, with only small variability on the single CPU.

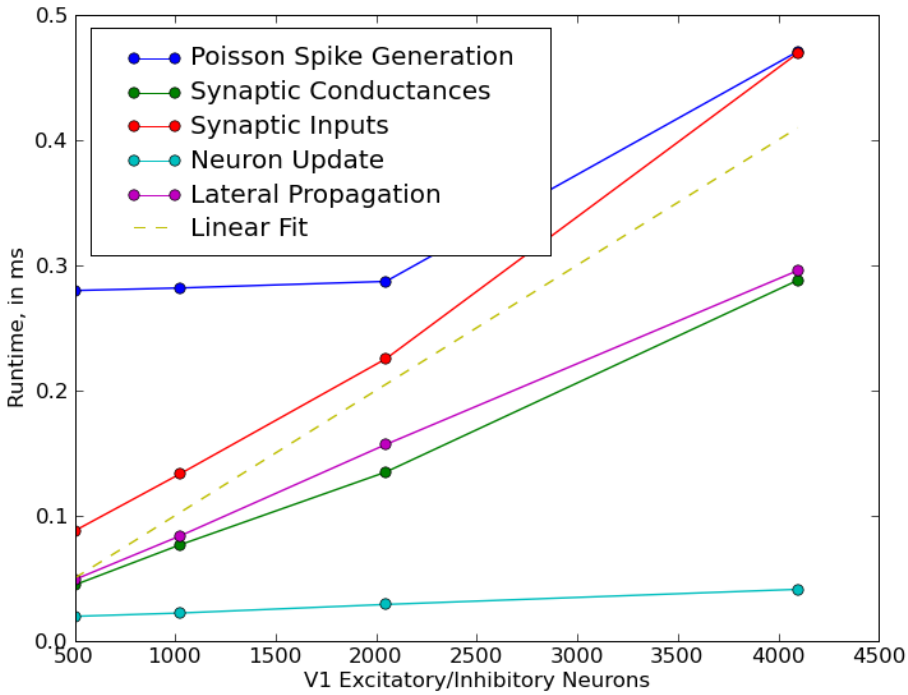

Figure 4.12: Average single-trial runtime of a single iteration of the CUDA version, running on the Tesla, for 4 different problem sizes: 512+512 V1 neurons, 1024+1024 V1 neurons, 2048+2048 V1 neurons, and 4096+4096 V1 neurons. Each algorithm component is shown as a separate line. Dashed line indicates linear slope.

Chapter 5

Discussion, Conclusions, and Future Work

In this chapter, I discuss the results of the experiments and present plans for extension of this thesis. First, I give an itemized summary of the original work. Then, in the remaining sections, I consider functional aspects individually, including possible future research.

5.1 Summary of Original Contributions

- I reimplemented an established theoretical model of visual cortex, in Python and C.
- I built extensive post-processing and analysis tools in Python, for various types of publishable figures.
- I quantified the performance of several statistical classifiers based on the output spike counts, combining several proposals from the neuroscience literature.

- I ported the simulation to OpenMP and CUDA concurrent architectures. Without detailed optimization, this yielded modest speedup on multiple cores, and a 20x speedup on a top of the line GPU.

5.2 Reproduction of the V1 Model

My implementation of the published V1 model faithfully reproduces many important characteristics of neural behavior in the visual cortex. This is confirmed by the good match of the results to comparison with multiple reference points from existing literature. The desired behavior was achieved even without implementing all of the details of the published model, including, but not limited to: the spike refractory period, alpha functions with smooth rise and decay, arbitrary spike delays, and more sophisticated numerical analysis techniques for the differential equations.

There are many promising future directions with the model itself, some of which are already being explored by various researchers. Random perturbation of the various activity constants is one avenue that has been pursued in [10]. This paper showed that modulation of certain parameters, which manifests in tuning curves variability, may actually lead to improved detection performance in the exact task I looked at. Reproducing such a result with my simulation will be straightforward. Prior work has explored the theoretical consequence of tuning curve widths [32][40], but it will be enlightening to *empirically* study the result of LGN to V1 connectivity statistics, particularly with an emphasis on overcoming pixel aliasing.

Finally, this model is a very simplistic realization of V1. In real V1 there are many more lateral and recurrent loops, as well as additional neuron types and top-down influence from higher cognitive processing[34]. These components are not yet well understood, and computational investigations with different connectivity patterns will continue to contribute to theories about possible guiding principles of cortical architecture. The incorporation of more sophisticated interconnections could also inform our understanding of how "context" is utilized by neuronal networks, particularly when the task includes real two-dimensional images with a richer set of features.

5.3 Practical Image Processing

The benchmark image processing task I chose, identification of a single line orientation, was deliberately unambitious, since building up the simulation and analysis platform consumed the bulk of my efforts. Extending this model to operate on real two-dimensional images is a primary future goal, which leverages an additional notion from the architecture of the visual cortex. In the implemented model, each V1 neuron prefers a line of a certain orientation, with all neurons centered on the same spatial point in the retinal input space. In the brain, each V1 neuron is sensitive to a different region of the retinal input, in a characteristic two-dimensional pattern known as a *pinwheel pattern*[41]. Understanding how the array of V1 neurons is able to code both orientation and spatial position with a single population is an important research question that also touches upon theoretical coding theory concerning

the simultaneous transmission of multiple data dimensions. Extension of my simulation to study this phenomena could occur completely in the Python code by merely changing the connectivity pattern between LGN and V1.

5.4 Orientation Discrimination and Spike Coding

I demonstrated that the network output, determined by the spike counts of the V1 excitatory neurons, provided adequate information to discriminate the orientation of the input stimuli. There was a slight disparity between the perceptron performance versus the results from the literature. This could be due to several factors, including ambiguity in published classification methods, and/or biological details I omitted in my implementation. Since the binary classification of two angles has limited practical utility and questionable neural implementation, I focused more on the identification of arbitrary orientations using the two estimation methods. It is not surprising that additional units provide more accurate estimates. This has been studied theoretically by the references cited in Section 2.3.

There are several additional questions to pursue related to estimation from spike counts. From an engineering perspective it is crucial to quantify the relative precision these techniques achieve. This analysis would undoubtedly include the total number of spikes necessary for accurate transmission. The relation of required spikes to information capacity would yield great insight into principles of efficient coding with arbitrary noisy binary channels. The analysis of the noise tolerance is another worthwhile research question.

That question can be studied by analysis of the performance of the estimators in conjunction with random perturbation of the spiking communication channels.

5.5 Parallelism Effort

The parallelism effort was both fruitful and enlightening. Without parallel acceleration of the code, it would have been much harder to perform the variety of experiments I tried.

Several observations related to Figure 4.11 deserve discussion. The fact that the runtime increased with the multicore version of the Poisson spike generator indicated that my usage of the standard C library `random()` function was not optimal for a concurrent architecture. Similarly, the additional cores of the Tesla versus the GeForce GPU did not translate to greater performance on this portion of the algorithm. I believe this was due to nonoptimal scaling in the problem decomposition. Figure 4.12 shows flat performance for the Poisson portion until 8192 total V1 neurons are included. This could be addresses in future work.

The benefit of parallelizing the synaptic conductance calculations was obvious. This part of the algorithm includes several multiplies and adds for each *synapse*, as quantified in Table 3.3. The neuron update part of the code did not benefit as greatly from concurrency, most likely because I iterated over each *neuron* in parallel, for ease of handling shared variable contention issues. There are certainly other ways to structure the algorithm which could

present greater speedups. I believe my version struck a good balance of high performance with few data flow assumptions. The latter characteristic is important for future experimentation with arbitrary network structures.

References

[1] J. von Neumann. *The Computer and the Brain*. Yale University Press, 1958.

[2] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman, March 1983.

[3] T. Serre, A. Oliva, and T. Poggio. A feedforward architecture accounts for rapid categorization. *Proceedings of the National Academy of Sciences*, 2007.

[4] M. Carandini and D. J. Heeger. Summation and division by neurons in primate visual cortex. *Science (New York, N.Y.)*, 264(5163):1333–1336, May 1994.

[5] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133, December 1943.

[6] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, August 1952.

[7] C. Koch. *Biophysics of Computation: Information Processing in Single Neurons (Computational Neuroscience)*. Oxford University Press, USA, 1 edition, November 1998.

[8] T. W. Troyer and K. D. Miller. Physiological gain leads to high isi variability in a simple model of a cortical regular spiking cell. *Neural Comput.*, 9(5):971–983, 1997.

[9] F. Worgotter and C. Koch. A detailed model of the primary visual pathway in the cat: Comparison of afferent excitatory and intracortical inhibitory connection schemes for orientation selectivity. *J. Neuroscience*, 11:1959–1979, 1991.

[10] M. I. Chelaru and V. Dragoi. Efficient coding in heterogeneous neuronal populations. *Proceedings of the National Academy of Sciences*, 105(42):16344–16349, 2008.

[11] B. A. Olshausen and D. J. Field. How close are we to understanding V1? *Neural Computation*, 17:1665–1699, 2005.

[12] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J Physiol*, 160:106–154, January 1962.

[13] D. C. Somers, S. B. Nelson, and M. Sur. An emergent model of orientation selectivity in cat visual cortical simple cells. *J. Neuroscience*, 15(8):5448–5465, August 1995.

[14] C. Enroth-Cugell and J. G. Robson. The contrast sensitivity of retinal ganglion cells of the cat. *The Journal of Physiology*, 187(3):517–552, December 1966.

[15] J. G. Daugman. Two-dimensional spectral analysis of cortical receptive field profiles. *Vision research*, 20(10):847–856, 1980.

[16] J.M. Alonso, W. M. Usrey, and R. C. Reid. Rules of connectivity between geniculate cells and simple cells in cat primary visual cortex. *J. Neurosci.*, 21(11):4002–4015, June 2001.

[17] M. Carandini. Melting the iceberg: contrast invariance in visual cortex. *Neuron*, 54(1):11–13, April 2007.

[18] C. M. Gray, P. König, A. K. Engel, and W. Singer. Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties. *Nature*, 338(6213):334–337, March 1989.

[19] M. Shamir and H. Sompolinsky. Nonlinear population codes. *Neural Comput.*, 16(6):1105–1136, 2004.

[20] H. P. Snippe. Parameter extraction from population codes: a critical assessment. *Neural Comput*, 8(3):511–529, April 1996.

[21] D. E. Rumelhart, J. L. Mcclelland, and the PDP Research Group. *Parallel Distributed Processing, Vol. 1: Foundations*. The MIT Press, July 1987.

[22] A. P. Georgopoulos, J. F. Kalaska, R. Caminiti, and J. T. Massey. On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex. *J Neurosci*, 2(11):1527–1537, November 1982.

[23] F. E. Theunissen and J. P. Miller. Representation of sensory information in the cricket cercal sensory system. ii. information theoretic calculation of system accuracy and optimal tuning-curve widths of four primary interneurons. *J Neurophysiol*, 66(5):1690–1703, November 1991.

[24] P. Baldi and W. Heiligenberg. How sensory maps could enhance resolution through ordered arrangements of broadly tuned receivers. *Biological Cybernetics*, 59(4):313–318, September 1988.

[25] H. S. Seung and H. Sompolinsky. Simple models for reading neuronal population codes. *Proc Natl Acad Sci U S A*, 90(22):10749–10753, November 1993.

[26] E. Salinas and L. F. Abbott. Transfer of coded information from sensory to motor networks. *J. Neurosci.*, 15(10):6461–6474, October 1995.

[27] A. Pouget, K. Zhang, S. Deneve, and P. E. Latham. Statistically efficient estimation using population coding. *Neural Comput.*, 10(2):373–401, 1998.

[28] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. Bower, M. Diesmann, A. Morrison, P. Goodman, F. Harris, M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, Thierry Vieville, E. Muller, A. Davison, S. El Boustani, and A. Destexhe. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398, December 2007.

[29] http://www.gimp.org/.

[30] D. E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd Edition) (Art of Computer Programming Volume 2)*. Addison-Wesley Professional, 3 edition, November 1997.

[31] P. Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 1st edition, December 2001.

[32] P. Seriès, P. E. Latham, and A. Pouget. Tuning curve sharpening for orientation selectivity: coding efficiency and the impact of correlations. *Nature Neurosci*, 7(10):1129–1135, October 2004.

[33] E. M. Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting (Computational Neuroscience)*. The MIT Press, 1 edition, November 2006.

[34] G. M. Shepherd, editor. *The Synaptic Organization of the Brain*. Oxford University Press, USA, 5 edition, November 2003.

[35] A. Pouget, S. Deneve, J. C. Ducom, and P. E. Latham. Narrow versus wide tuning curves: What's best for a population code? *Neural Computation*, 11(1):85–90, 1999.

[36] NVIDIA Corporation. NVIDIA CUDA Compute Unified Device Architecture Programming Guide, Version 1.1, 2007.

[37] S. Deneve, P. E. Latham, and A. Pouget. Reading population codes: a neural implementation of ideal observers. *Nature neuroscience*, 2(8):740–745, August 1999.

[38] M. A. Paradiso. A theory for the use of visual orientation information which exploits the columnar structure of striate cortex. *Biol. Cybern.*, 58(1):35–49, January 1988.

[39] G. Westheimer. Diffraction theory and visual hyperacuity. *American journal of optometry and physiological optics*, 53(7):362–364, July 1976.

[40] H. P. Snippe. Parameter extraction from population codes: a critical assessment. *Neural Comput*, 8(3):511–529, April 1996.

[41] T. Bonhoeffer and A. Grinvald. Iso-orientation domains in cat visual cortex are arranged in pinwheel-like patterns. *Nature*, 353(6343):429–431, October 1991.