VARIABLE BINDING IN BIOLOGICALLY PLAUSIBLE NEURAL NETWORKS

MSc Thesis (*Afstudeerscriptie*)

written by

Douwe Kiela
(born June 7th, 1986 in Amsterdam, the Netherlands)

under the supervision of **Prof. Dr. Michiel van Lambalgen** and **David Neville, M.Sc.**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
August 30, 2011	Prof. Dr. Michiel van Lambalgen
	Prof. Dr. Frank Veltman
	Prof. Dr. Benedikt Löwe
	David Neville, M.Sc.

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Contents

Chapter 1

Introduction

One of the most essential aspects of logic is the ability to bind variables. Without that ability, we cannot represent generic rules in a logical form. In the past decades there has been a lot of interest in implementing logic on neural networks, but this was largely restricted to propositional logic. There have been attempts to implement more sophisticated logics on neural networks, with mixed success. So far, there have been no results that conclusively (and non-theoretically) show that first-order logic can successfully be implemented on neural networks. The most important step towards doing just that, or at least doing so for a fragment, is implementing variable binding on neural networks.

In the study of logic and neural networks, the connectionist paradigm has been of pivotal importance all across the domains of cognitive science. However, one might consider traditional connectionism slightly outdated or oversimplified compared to our current knowledge of the workings of neurons and the brain. Since the advent of the original artificial neural networks, the discipline of computational neuroscience has made significant progress. The fine-grained dynamics in such models may provide new insight into the relative standstill that the study of logic (using variables) and neural networks has suffered from in recent years. This interrelation has, so far, been largely unexplored for these biologically more plausible models.

The purpose of the thesis at hand is to unite these two points, insofar as that it aims to show that biologically plausible neural networks are capable of representing predicates and are capable of performing variable binding on the predicate's variables. The results presented in the current thesis are preliminary, meaning that they are merely meant to show that exploring the combination between logic and computational neuroscience can be a fruitful approach and that it should be pursued much further. The outline of the thesis is as follows. First we will establish a solid foundation, covering the background of the study of logic and neural networks in depth, from the inception of artificial neurons to the representation of predicates. In chapter 3, the binding problem–one of the major hurdles to be taken before variable binding can even be attempted–is discussed, along with the solutions that have been proposed over the years.

Chapter 4 makes explicit the distinction between connectionism and computational neuroscience, in order to draw attention to biologically plausible neurons. Chapter 5 briefly describes the methodology. The correlation between spike times as a measure for binding is described and the software chosen to perform the simulations is introduced, as well as the software used to analyze the results. Since biologically plausible networks are necessarily highly parameterized, the chosen parameters are discussed as well. The next chapter will turn to the actual experiments used to show that such networks are indeed capable of performing variable binding. Several criteria have been set in the preceding chapters, which will be addressed in the models. Chapter 7 will briefly discuss these results and shed light on some of the remaining issues. Moreover, it discusses the conception of logic that is at the base of what the thesis proposes; and discusses some important aspects of extending the research presented. The closing chapter will look back on what has been covered in the thesis and provides a brief roadmap towards achieving the final goal of implementing first-order logic in full.

1.1 Acknowledgments

The author would like to thank a number of people without whom this thesis would have been impossible. Many thanks go to Michiel van Lambalgen for suggesting the topic, providing many useful suggestions, giving invaluable feedback and showing that there is much more to logic than “just logic”. To David Neville for never giving up on the project even when the author almost did, steering me in the right way and being a tremendous help with getting to grips with the simulation software and the peculiarities of computational neuroscience. To Reinhard Blutner for not only awakening my interest in neural networks but also showing that there is so much still to be done. To Alistair Isaac and Thomas Icard for our many discussions on anything neural network-related during my stay at Stanford University’s CSLI–which would not have been possible without Johan van Benthem’s support. Special thanks go to Ted, Alice, Heleen and Peter for their support and for just being there. Finally, this thesis would have been inconceivable without Kattria: her positive attitude, undying support and infectious enthusiasm have been what has kept me going and have been a great inspiration.

Chapter 2

Background

From a logical standpoint, the development of artificial neural networks has always been heavily intertwined with propositional logic. The very first artificial networks introduced by McCulloch and Pitts in their seminal 1943 paper [58] equate individual nodes with propositional formulae. The discovery by Minsky and Papert that McCulloch-Pitts networks could not implement exclusive disjunction [61] resulted in a decline in artificial neural networks research, but the subsequent development of the backpropagation algorithm [101][73] led to a renewed interest from the early 80's onwards.

The line of thought that emerged from this renewed interest was at first predominantly an artificial intelligence affair, but soon awakened the interest in philosophers. The *Stanford Encyclopedia of Philosophy* describes connectionism–as the line of thought is known–as "a movement in cognitive science which hopes to explain human intellectual abilities using artificial neural networks", which "promises to provide an alternative to the classical theory of the mind: the widely held view that the mind is something akin to a digital computer processing a symbolic language" [30]. For our current purposes it is not necessary to go into the full details of the so-called connectionist-symbolic debate (but see e.g. [72]). However, it is important to note that this distinction and the alleged incommensurability [18] of the two paradigms has inspired prolific research into the general topic of the thesis at hand, namely the implementation of logic on neural networks.

The rationale behind such approaches is as follows. If we can implement logic, which is after all the archetypical symbolic language, on neural networks, then the two paradigms can be united as different sides of the same coin. Or in other words, if such an approach is successful, it shows that it is possible to implement reasoning on the symbolic level using processing on the sub-symbolic level. Unsurprisingly then, the interface between logic and neural networks has become one of the focal points of connectionist research. With the advent of Hopfield networks [40] and Elman networks [24], and the increase in available computational power, ever more complex networks were used to study logic and language in neural networks. Influences from physics inspired a study in the energy distribution of networks, with Pinkas arguing that the global minimum in the energy function of symmetric networks corresponds to finding a model for a propositional formula [67] (another example is [11]). Gärdenfors & Balkenius showed that neural networks perform non-monotonic reasoning [8]. Even recently, new models of neural networks have been developed that use logic as a base [76].

The popularity of the logic programming paradigm [54] in the study of artificial intelligence opened a way to exploit the ability of networks to learn facts (i.e., the declarative side of logic programs) and at a later stage to process the network and examine the inputs and outputs (i.e., the procedural side) in a logically rigid way [37]. What is more, logic programs are ideally suited to incorporate non-monotonic reasoning, through for example negation as failure or completion algorithms [29]. The study of propositional logic programs and artificial neural networks yielded impressive results. For example, Hölldobler et al. were able to develop a connectionist system for computing the least model of propositional logic programs, provided such a model exists [38]. However, all studies mentioned above have in common that they are focused on propositional logic. The lack of research into more expressive logics famously led McCarthy very early on to pose overcoming "propositional fixation" as one of the major challenges facing connectionism [57].

Answering McCarthy's challenge, however, has proven extremely difficult. The vision presented by Smolensky [83], where connectionist systems fully incorporate the power of (first order) symbolic logic, is still far from complete. Numerous attempts have been made, with mixed results. Unary relations were modeled in [9] and predicates with higher arity and inference rules were modeled in [79] and [47]. Although these attempts were successful within a limited scope, nobody has come up with a theory as of yet that conclusively settles the issue once and for all. Quite obviously, classical logic would be the most desirable, but the apparent inability to come up with any conclusive results has caused some researches to move away from classical first-order logic and towards things like "connectionist modal logic" [20]. However, positive results have been obtained for first-order fragments of classical logic: Hölldobler et al. were able to prove that the least model of first-order logic programs can be approximated arbitrarily well by multi-layer recursive networks with a feed-forward core [39]. First-order logic programs are an especially suitable candidate, not only because of the aforementioned properties of logic programs in general, but also because they do not require an external (and potentially infinite) domain, like classical logic would. Instead of a Tarskian semantics based on objects in the domain, first-order logic programs apply unification to satisfy clauses, using substitution of identical (or equal) terms [4].

One of the first questions any attempt to answer McCarthy's challenge has to ask itself is; how does one represent the basic building blocks of the logic? In the context of first-order logic, this comes down to making explicit how terms, predicates and logical connectives are implemented. Whereas the logical connectives need not necessarily be "represented" as such because they may be implicit in the organizational structure, the same cannot be said for terms, predicates (i.e., atomic formulae) and complex formulae that consist of combinations of atomic formulae and logical connectives.

The issue of representation coincides with another heated philosophical debate, namely, whether "meanings" are represented at all. The sentence *John loves Mary* may be encoded using a distributed representation that does not contain any explicit representation of the constituents John, loves, and Mary [84]. Even though these constituent representations can be retrieved from the network, the network itself does not need to retrieve them in order to process the entire sentence correctly [17]. These distributed representations are the opposite of localist representations, where every constituent is assigned a single node that represents it (like McCulloch-Pitts neurons, for example). In the past these localist representations have been criticized on the basis of the Grandmother-cell analogy [18]: if my grandmother is represented by one single neuron, what happens when that neuron is damaged? Do I forget my grandmother? Psychological and neurological evidence suggests that this is not the case and that seeing your grandmother triggers a pattern of neural activation distributed all over your brain.

Distributed representations are often portrayed as incompatible with the symbolic paradigm, which is said to be inherently localist. In this sense, distributed representations have several advantages: they allow for the combination of not-necessarily-propositional content, such as olfactory or kinesthetic data, with propositional content and they are robust in the sense that minor damage does not destroy the representation–which allows distributed representations to degrade gracefully. The idea is based on a theory proposed by Donald Hebb, which proposes that neurons that are distributed all over the brain form assemblies (i.e., neurons become associated with each other) by firing together [36]. Thus, your grandmother is represented by a Hebbian assembly, which was formed through Hebbian learning–under the motto "fire together, wire together".

However, the explicit distributed-localist dichotomy has to be recognized as at least slightly misguided. First and foremost, distributed representations are incompatible with symbolic representations only under the assumption that symbolic reasoning can never be implemented on neural networks in a distributed fashion–which depends in turn on the level of description you apply to symbolism and connectionism, respectively. Given the topic of the current thesis, we quite obviously have to reject this conception. Furthermore, localist representations are not incompatible with distributed representations either. Again, this is a matter of the level of description and the level of explanation. When we have three localist neurons encoding for John, loves and Mary, what is their collective representation as *John loves Mary*, if not a distributed representation? What if the three neurons together are used in a higher-order representation, such as *"John loves Mary" is a sentence*? Apart from some of the connectionist hardliners, localist representations are typically considered as a good way of representing atomic data which can then be combined. What is more, a localist neuron can be taken to be a representation of a neural assembly, such that when we say we assign John to neuron X, neuron X might be, without loss of generality, interpreted as a neural assembly–a distributed representation, in other words. Thus, when we want to represent terms, predicates and formulae in a neural network, our primary decision is what we will choose as our atomic data, which will be given a localist representation. In our case, of course, the most primitive constituent is a term; specifically, constants and variables. However, there is more to it than that. Consider the following predicate:

$P(x_1, x_2, ..., x_n)$

Where constants and variables are the fillers for the predicate's arguments, we also have to keep track of which filler is associated with which argument. In other words, we have to know the role of the constants and variables in the predicate. Taking these considerations into account, the most basic spatial representation of an n-place predicate would be something along the following lines[1]:

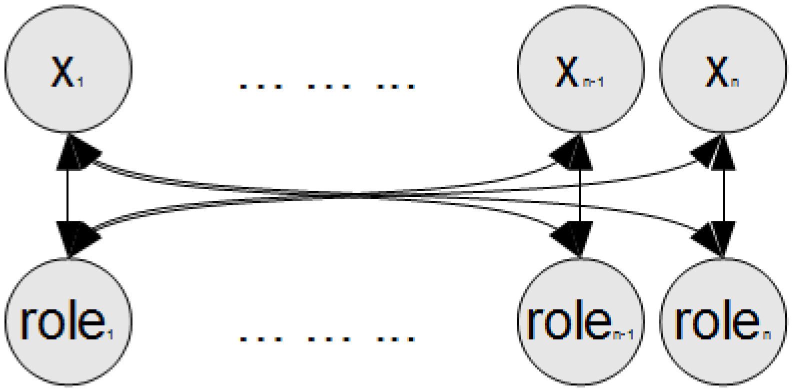

Figure 2.1: Representing a Predicate in a Neural Network

All available constants and variables are connected to all the available roles of the predicate, because all constants and variables can bind to all these available roles. However, this leads to a problem: how can we represent these bindings in such a way that we can distinguish multiple encodings and find the correct bindings? That question is the topic of the following chapter.

[1]As we shall see, an important aspect of neural network models that is lacking in this picture is inhibition. However, for the current purpose, namely, showing that the variables need to bind to roles for predicate representation, the figure is clearer without inhibition, which will be addressed later on.

Chapter 3

The Binding Problem

A single predicate is arguably one of the simplest forms of a combinatorial structure we can possibly encode in a neural network. However, combinatorial structures have been the root of a substantial challenge to connectionism and neuroscience, known as the binding problem. The binding problem occurs in many different shapes and guises. As a result, the term is applied to a wide variety of not necessarily related instances where "binding" can occur and cause problems. In its simplest form, the binding problem concerns the way in which neural instantiations of parts can be bound together in a manner that preserves the structure of the whole that is made up out of the parts. To give an easy and well-known example which was first given by Rosenblatt [71], consider a simple neural network that is given the task to encode and retrieve two colored shapes: a green triangle and a blue square.

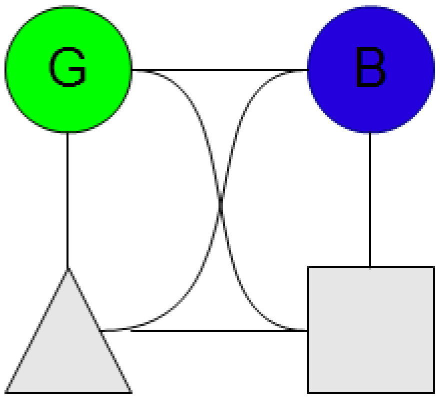

Figure 3.1: Rosenblatt's Binding Problem

The neural instantiations of the parts, the constituents of the color-shape combinations, are green, blue, triangle and square, respectively. The problem arises when we want to distinguish between the two shapes: the color green has to be bound to the shape triangle, whereas the color blue has to be bound to the shape square. We are unable to represent both structures in the same network and be able to tell them apart, because there is no way to determine which color belongs to which shape.

For example, consider the two color-shape combinations encoded in a simple Hopfield network with a binary activation function, with the nodes in Figure 3.1 as the nodes in the network. In order to represent the green triangle, the nodes Green and Triangle have to be activated. For the blue square, we need Blue and Square to be active. When both are active at the same time, the triangle and square could be either green or blue: there is no way to tell them apart.

A problem by the same name has been heavily debated in philosophical discussions of the so-called unity of consciousness. The problem roughly comes down to the issue of how all our perceptual experiences can be combined–that is, bound together–into one big picture, a "Cartesian theatre", which we call conscious experience. A similar problem is found in binding data from different modalities, i.e. auditive and visual perception. Although an important issue in its own right, this type of binding problem is not what is at stake when we are looking into encoding predicates on neural networks. Hence, in order to be as clear as possible, it is important to clearly demarcate the particular instance(s) of the binding problem that we are faced with.

In the context of the neural instantiation of linguistic structures, i.e. the combinatorial and rule-based structure of language, Jackendoff [42] has presented "four challenges for cognitive science", all pertaining to a particular type of binding problem. As Jackendoff himself has acknowledged, although his book is about linguistic structures, the same problems apply to any combinatorial structure we wish to give a neural instantiation. The four challenges are (see also [55]):

1. The massiveness of the binding problem
2. The problem of multiple instances
3. The problem of variables
4. The relation between binding in working memory and binding in long-term memory

The first problem corresponds to the simplest form of the binding problem that was mentioned above. Simple though it may be, Jackendoff notes that this problem has a massive impact specifically on linguistics. For example, when we want to neurally instantiate the sentence *John loves Mary*, how can we distinguish that sentence from *Mary loves John*? Another favorite example is *The little star is beside the big star*, which involves a much higher number of bindings than the typical example of the visual binding problem sketched above [31]. This sentence also confronts us with Jackendoff's second challenge, namely, the word "star" is used twice, but binds to different attributes (little and big). The problem is, when we have multiple instances of the word star, how do we ensure that the right binding is used at the right moment, and how do we avoid erroneous bindings such as *little big star*? Even if we are able to solve these two problems, we have only solved the binding problem for the knowledge of specific facts. But how about systematic facts, such as the fact that gives(x,y,z) entails owns(y,z)? We could encode it for the case of *John gives Mary a book*, but then what happens when Bob gives Mary a book? Obviously, we would want our neural instantiation of such a rule to be systematic enough to be able to deal with any case in which the rule applies. In other words, the rule has to be generically encoded, such that we can instantiate the variables at a later stage. It is easy to see that the amount of binding necessary for such a case is much larger than in the first two cases, because objects or attributes need to bind to variables, which in turn need to bind to their respective arguments in give and own. This, the third challenge, is known as the variable binding problem, which is an important problem since rule-based relationships play such an important role in human cognition. And then there is also the question of where and how this binding takes place in the brain. Working memory is typically assumed to consist of activity in neurons that encode the information that is currently active in the working memory [26]. That is, a piece of information only stays in working memory for so long as the activity encoding that information is sustained [27]. On the other hand, long term potentiation, which is associated with long-term memory, results from synaptic modification in the connections between neurons. Jackendoff's last challenge is about understanding how the interface between these two types of memory occurs, specifically when it comes to the type of binding that happens.

All four challenges are important for incorporating first-order logic, or first-order fragments, on neural networks, but the central issue is the variable binding problem. In order to solve the variable binding problem, we first need a good solution for the first two problems. Jackendoff's last challenge is a little different. It does not directly relate to our current topic of interest, but the way you choose to implement variable binding for predicates is directly related to how you want to answer that question. Any theory that claims to "solve the binding problem" needs to answer at least the first two challenges. The answer to the fourth challenge is a consequence of the choices made for the implementation, so biological plausibility in that sense would give the theory as a whole more credibility.

Browne and Sun [14], give several reasons why "the ability to perform variable binding is essential if connectionist systems are ever to perform complex reasoning tasks" in their nice overview of variable binding in connectionist networks. Like Jackendoff, they also list the ability to reason with combinatorial structures and the ability to have generic rules, but they also add that the strict constraints variable binding imposes on rule-matching may be necessary for modeling human language acquisition (see also [68]). Browne and Sun characterize variable binding as a form of complex pattern matching, with three main forms, from simplest to most complex:

1. Atomic symbol matching. When we teach green(triangle) and then ask the network whether the triangle is green, it should reply yes.
2. Standard pattern matching. Teaching green(triangle) and then asking the network *what is green*, i.e., green(X), it should reply X=triangle
3. Unification. When both pattern and datum can contain variables. When we teach dog(X) and ask dog(Y), then it should give us Y=X, even though both are variables.

Taking that characterization of variable binding as a roadmap for coming up with a possible solution, we should start off with the simplest form, atomic symbol matching. This corresponds to the simplest binding problem. From there on out, we can try adding more and more complexity.

3.1 Solutions to the Binding Problem

Over the years, many different solutions to the variable binding problem have been proposed, all in varying detail and coming from very different disciplines in the spectrum of cognitive science. Broadly speaking, two main strands can be discerned in all these different theories: the temporal and the spatial solution.

At first glance, the spatial solution seems a very natural answer to solving the binding problem: if we have to encode a blue square and a green triangle, why don't we simply have one neuron for a blue square and one for a green triangle? However, this quickly leads to the neural equivalent of combinatorial explosion. Psychological studies suggest common knowledge in humans amounts to as much as 10^8 items [51]. Given that we only have about 10^{12} neurons in the brain by most estimates and that most of these neurons are likely to be involved in other processes than knowledge processing, there is no way we could encode combinatorial structures like this–let alone when we want to encode more properties, such as a *big equilateral flashing dotted soft furry green triangle*, just to name something.

Much more elegant spatial solutions have been suggested than this simple view, of course. The most famous one is probably Smolensky's tensor product representation [84], which uses the tensor product of distributed representations in order to represent binding between the two. The tensor product of an n-place vector V and an m-place vector W is the $n*m$-place vector whose value consists of the pairwise products of V and W. Another spatial solution is holographic reduced representations [69], which uses circular convolution on vectors that represent associated items. As opposed to tensor products, holographic reduced representations have actually been used to model the third and most difficult type of variable binding–unification [97], as have Kohonen maps [44][98]. However, in general these spatial approaches have been faced with much criticism. It has been noted that the vectors are not of a fixed dimension, making them very impractical computationally [31]. Apart from the combinatorial explosion that renders the spatial models neither biologically plausible nor computationally feasible, it has also been noted that they tend to operate in a serial fashion, whereas cognition is known to be parallel [79].

Due to the spatial and quantitative constraints on the human brain and research in the 1980s that suggested synchronous firing–"gamma oscillations"–in the cat's visual cortex, temporal binding was presented by von der Malsburg as a viable alternative to solving the binding problem [95]. Specifically, the timing of spikes in neural networks, or the local field potential of neural assemblies, could be used to bind the combinatorial aspects of a structure. For example, if green and triangle fire together, they are bound together. This idea is also strongly related to Hebbian learning and Hebbian assemblies [36]. Traditional rate models, such as the Hopfield network with binary activation that we discussed earlier, do not take the actual firing times of neurons into account. If we look at the firing times of neurons encoding green, blue, triangle and square, we can see that the respective neurons for green triangle and blue square fire in synchrony, but that they have identical firing rates.

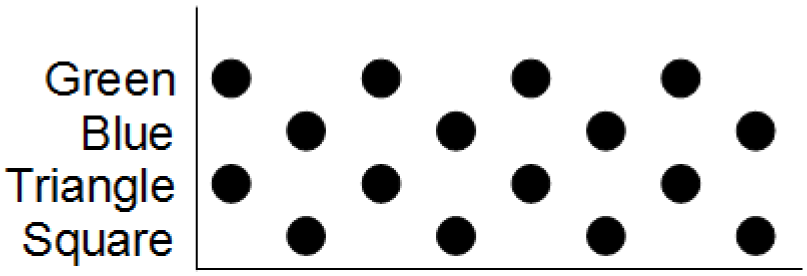

Figure 3.2: Different Spike Times with Identical Firing Rate

The temporal model has been suggested pretty early on as a particularly good way to solve the variable binding problem [25][96]. The approach has met with quite some success in a large range of problems of brain function [34] [82]. But it is important to note that there are also constraints on the temporal model, mostly derived from psychological experiments. A system that applies a temporal solution to the binding problem has to be able to perform binding at a reasonable time interval, one that is comparable to humans.

Von der Malsburg's work and the spatial and temporal constraints inspired Shastri & Ajjanagadde in their seminal paper [79] to model what they called "reflexive reasoning" using a quite successful approach of temporal binding on localist representations. Shastri & Ajjanagadde use the synchronous firing of specially constructed neuron-like elements as a way of representing dynamic binding. These representations were subsequently used to form rules, where the synchronous oscillations propagate through interconnected representations.

Their neuron-like elements are idealized oscillators of four different types:

1. τ-and nodes: produce an oscillatory pulse with same period and pulse width as its input
2. ρ-btu nodes: produce a periodic spike train in phase with the input spike train[1]
3. τ-or nodes: become active when it receives one or more spike signals within one oscillatory period
4. Multiphase τ-or nodes: become active when it receives more than one spike trains, in different phases, during the same oscillatory period

Using these elements, Shastri & Ajjanagadde are able to encode an n-ary predicate P using two τ-and nodes and n ρ-btu nodes. By connecting these nodes in specific ways, facts and rules can be encoded that amount to a signicant fragment of Horn-clause logic. Using these nodes, synchrony between John and Lover and Mary and Lovee encodes which role (lover, lovee) in the predicate is taken by which filler (john, mary). In a similar fashion to Abeles' synchronous firing chains [2], the synchronous pattern propagates through the chain, allowing an antecedent predicate to enable the consequent predicate. It is beyond the scope of the thesis at hand to go into full detail, so suffice it to say that the network is constructed such that the nodes form a pathway towards a certain inference.

Shastri & Ajjanagadde's results were enthousiastically received and still form the basis for many explorations of the binding problem that make use of temporal solutions. The model has also been extended to model larger fragments of Horn clauses and a larger set of inferences, such as in [63] and [64]. Oscillators have also been used in completely unrelated ways but much to the same purpose, such as in [89] and in [91].

In addition to what we may call the strictly-spatial and strictly-temporal approaches, many hybrid approaches have emerged which do not clearly subscribe to either of these paradigms. Fortunately a number of excellent overviews and surveys exist, most notably [5], which provide an adequate excuse not to discuss these in detail. Approaches that fall in this category are theories that do not acknowledge the necessity of representing combinatorial structures at all (see [92] ch. 4). Theoretical work that exemplifies this approach and that deserves distinct mention is recent work by Hölldobler et al. [7] which tentatively and preliminarily extends their so-called Core model to first-order logic. Furthermore, hybrid approaches such as blackboard models [92], signature-based processing (e.g. [14]) and combinations of these approaches [10] have been proposed in recent years.

[1]It is unclear what the letters *btu* are supposed to stand for. Suffice it to say that these types of nodes play the key part in the propagation of the signal.

In the end, when evaluating all these results and different theories, much progress has been made, but there are still many open challenges [6] on the way to fully realizing the “logic on neural networks” paradigm. This will be the topic of the next chapter.

Chapter 4

Connectionism and Computational Neuroscience

As was already briefly discussed in the previous chapter, spatial solutions to the binding problem were inadequately equipped to fit within the known constraints of the human brain. It is important to note that this fact does not at all disqualify any such theory, as indeed there are many more interesting applications of connectionism outside the particular domain of understanding human cognition. For example, as Bader & Hitzler make clear in their survey of neural-symbolic integration, integrating connectionist and symbolic knowledge representation may also be motivated from a more technical perspective, for example by trying to find optimal ways to combine the advantages of both approaches [5]. The same constraints of course do not apply in this domain, which is particularly relevant for computer science-related applications of knowledge representation. For the thesis at hand, however, we are predominantly concerned with biological plausibility, because a more biologically-oriented approach towards logic on neural networks is advocated.

Some of the temporal solutions we saw are much more in line with aiming for biological plausibility. Shastri & Ajjanagadde's work, for example, specifically addresses the question how biological networks can perform reasoning–reflexive reasoning as they call it, as opposed to reflective reasoning–very quickly. As a result, they spend a significant amount of time in defending the biological plausibility of their system.

4.1 Connectionism

Bader & Hitlzer distinguish between these two approaches as "connectionist" and "neuronal", where the latter is aimed towards achieving biological plausibility. To be fair, this distinction deserves a lot more attention than it is awarded in their survey. Shastri & Ajjanagadde, for example, would be classified as being in the neuronal category, whereas they frequently refer to their own approach as connectionist. At the same time, people like Smolensky certainly, though according to critics unsuccessfully, pay a significant amount of attention to the biological underpinnings and neuronal implications of their models. It seems that something is awry if we choose to bluntly classify these approaches as neuronal and connectionist, respectively.

As mentioned before, connectionism started as a study of human cognition. Notably, this was the objective of McCulloch and Pitts, with other champions of connectionism such as Rumelhart and McClelland closely following this goal. However, whereas our understanding of the inner workings of the biological neuron has greatly improved since McCulloch and Pitts published their paper, the artificial neurons that we find in modern day connectionism–in the sense of Bader & Hitlzer–are still very similar. For example, Hölldobler et al. use binary threshold functions, with occasional sigmoidal functions, on their nodes, while the teaching happens through the backpropagation algorithm [6]. Shastri & Ajjanagadde, on the other hand, use what they call neuron-like elements that allow for precisely fixing the oscillatory behavior, which is obviously very different from the original connectionist neurons. In that sense, it is correct to classify the approaches as distinct.

However, these neuron-like elements are still a far cry away from what we know about biological neurons. As Bader and Hitzler duly note, relatively recent developments in computational models of biologically plausible neurons, such as spiking neurons [53], have "hardly been studied so far" when it comes to encoding and retrieving symbolic knowledge ([5] p. 15) (with the exception of [86]). Furthermore, they "perceive it as an important research challenge to relate the neurally plausible spiking neurons approach to neural-symbolic integration research" [ibidem].

Biologically plausible neurons belong to the domain of neuroscience, with the mathematical description of the neurons falling under the field of theoretical- or computational neuroscience. Comparing computational neuroscience to Bader & Hitzler's neuronal and connectionist approaches, we are able to draw a much more distinct line. In order to make this distinction clear, let us stipulate that in what follows, connectionism in the case of neural-symbolic integration will be interpreted in a broader sense than Bader and Hitlzer do:

Definition 4.1.1. Connectionism is the approach in neural-symbolic integration that can be characterized as follows:

1. Neurons or nodes are idealized units in a network, that compute very simple functions only
2. Analysis of the network happens through studying the network's input-output function
3. Networks are construed in a designated connection-pattern, arranged such that the input-output function delivers the desired result

4. Learning in the network, if present, is idealized, usually through backpropagation or sometimes simple forms of Hebbian learning

5. Neurons, connections, networks and learning need not have a plausible biological basis

According to this definition, both neuronal and Bader & Hitlzer's "connectionist" neural-symbolic integration fall under connectionism, because they are characterized by the same approach. Both approaches use idealized neurons–either neuron-like oscillators or binary activation neurons–and both approaches typically take the behavior of the input-output function as a measure of adequacy of their network's configuration. Furthermore, both approaches use specific connection patterns to enable their input-output function to work correctly, such as connecting particular oscillation-propagating nodes in Shastri & Ajjanagadde, or recurrently connecting input to outputs to ensure that the consequence operator finds the least model in Hölldobler et al. Shastri & Ajjanagadde do not incorporate learning[1], whereas Hölldobler et al. and Smolensky use the backpropagation algorithm.

Although Shastri & Ajjanagadde explicitly focus on biological feasibility, in the sense that synchronized oscillators have a biological basis, the way they construct their network is far from biologically plausible [77]. Furthermore, their approach is successful because they are able to exhibit full control over firing rates of their idealized neuron-like elements, which is much harder to achieve on biologically plausible neurons [52] (see also e.g. [1], [81]). Thus, they fall firmly within the connectionism definition above. All in all, connectionism can be more generically seen as an approach that constructs networks out of idealized nodes to study the input-output function.

4.2 Computational Neuroscience

Where we might term connectionism "constructivist", the type of research that happens in computational neuroscience is decisively "empiricist". The physiological properties of the neuron are fixed mathematical descriptions derived from empirical research. In other words, for computational neuroscience "it is important to bear in mind that the models have to be measured against experimental data; they are otherwise useless for understanding the brain" ([88], p2). Thus, there is no possibility of directly tweaking something like a biological neuron's oscillations–because its oscillatory behavior depends on a large set of physiological and physical parameters.

The fundamental model in computational neuroscience is the Hodgkin-Huxley model. The HH-model is a so-called conductance-based model, which incorporates the behavior of the chemical synapses of neurons, the response of the postsynaptic membrane potential to synaptic events and the behavior of action

[1]More recent enhancements to the original model allow for the support of negation and inconsistency [78], very basic learning capabilites [99] and re-use of rules [100].

potentials, using an elegant set of differential equations. Since the computational model that we will be using is a simplification of the HH-model, we will have to cover its basics in order to justify the simplifications. After all, if we are advocating the use of biologically plausible neurons in the study of neural-symbolic integration in favor of the idealized neurons in connectionism, aren't we ourselves committing exactly the same fallacy in simplifying the biologically plausible model?

Since we are interested in the information-processing capabilities of neurons, we first have to gain insight into the mechanisms of information transmission within single neurons and between neurons. As any high school biology textbook tells us, a neuron consists of a soma, dendrites and the axon. Neurons are connected through synapses, which may be excitatory or inhibitory, depending on the type of neurotransmitter and associated ion channels in the synapse. The sending neurons are in contact with the receiving neuron at synapses either at the soma or the dendrites. In the context of synaptic transmission, these two neurons are called the presynaptic and the postsynaptic neuron, respectively. The synapses influence the influx and outflow of certain ions through the ion channels, the exact composition of which determines the membrane potential in the postsynaptic neuron, which is defined as the difference between the electric potential within the neuron and its direct environment. Excitatory synapses increase the membrane potential, in so-called excitatory postsynaptic potential, whereas inhibitory postsynaptic potentials inhibit the rise of membrane potential. The changes in potential can ultimately trigger the generation of an action potential, also known as a spike, which consists of a rapid depolarization of the neuron, followed by a brief spell of hyperpolarization. After a spike the neuron returns to the resting potential due to ion channels that are usually open all the time, causing the neuron to "leak" ions until it returns to an equilibrium with its environs.

In order to make clear the exact behavior of the action potential and to grasp the HH-model, it is useful to turn to a short mathematical exposition of the intricacies involved[2]. The membrane capacitance can be used to determine how much the current changes the membrane potential at a given rate. The time derivative of the basic equation that relates membrane potential and charge is:

$$C_m \frac{dV}{dt} = \frac{dQ}{dt} \tag{4.1}$$

Since the time derivative of the charge dQ/dt is equal to the current passing into the cell, the amount of current needed to change the membrane potential of a neuron with capacitance C_m at rate dV/dt is $C_m dV/dt$. In other words, the rate of change of the membrane potential is proportional to the rate at which charge builds up inside the cell, which is in turn equal to the total amount of current entering the neuron.

[2]The mathematical exposition as it occurs here owes much to [21] and [88], which both present detailed mathematical descriptions that span the entire field of computational neuroscience.

For the different types of ion channels that regulate the current flow we can compute the equilibrium potential, which is the membrane potential at which ion influx cancels the ion outflux and the channel reaches an equilibrium[3]. The equilibrium potential depends on the concentration of ions inside the cell, [inside], and the concentration outside the cell, [outside]. When the ion has an electric charge zq, where q is the charge of one proton, the equation for the equilibrium potential of that ion, which is known as the Nernst equation, becomes[4]:

$$E = \frac{V_T}{z} \ln \left(\frac{[outside]}{[inside]} \right) \tag{4.2}$$

We label the different types of ion channels through an index i. The current increases or decreases approximately linearly when the membrane potential deviates from the value of E_i when $E_i = V$. We give the membrane current resulting from a particular channel of type i as $g_i(V - E_i)$. The total membrane current is then defined by the following equation:

$$i_m = \sum_i g_i(V - E_i) \tag{4.3}$$

Usually, it is easier to explicitly model pump currents. This is denoted as $\bar{g}_i$ with a line over the parameter to make clear that it has a constant value.

The Hodgkin-Huxley model defines the generation of the action potential, in its single-compartment form. It is constructed by summing the leakage current and the K^+ and Na^+ currents:

$$i_m = \bar{g}_L(V - E_L) + \bar{g}_K n^4 (V - E_K) + \bar{g}_{Na} m^3 h (V - E_{Na}) \tag{4.4}$$

The variables n, m, and h are known as the gating variables. We know that ion channels are not necessarily open all the time: they function more like gates that can open and close. The gating variables are described by the following equation:

$$\tau_n(V) \frac{dn}{dt} = n_\infty(V) - n \tag{4.5}$$

where $\tau_n(V)$ and $n_\infty(V)$ are defined in terms of the opening and closing of the gate[5]. Combining the above, the basic equation for the Hodgkin-Huxley model is

[3]Ions flow into or out of channels, changing the current flow, depending on electric forces and diffusion. The equilibrium potential is defined as the membrane potential at which current flow due to electric forces cancels the diffusive flow.

[4]Not all channels are so selective so as to only allow one particular type of ion. To take this into account, the value for E is sometimes computed as an intermediate value between the equilibrium potentials that the ion channel allows to pass through. This is known as the *reversal potential*. V_T is the potential at temperature T where the thermal energy of the ion is high enough to cross the membrane.

[5]Let $\alpha_n(V)$ be the voltage-dependent rate of gating transitions from closed to open, and $\beta_n(V)$ the rate for the reverse. Then $\tau_n(V) = \frac{1}{\alpha_n(V)+\beta_n(V)}$ and $n_\infty(V) = \frac{\alpha_n(V)}{\alpha_n(V)+\beta_n(V)}$.

$$C_m \frac{dV}{dt} = -i_m + I_e \tag{4.6}$$

where I_e is the presynaptic potential, that is, the input current.

An important realisation allows for a great simplification of the HH-model: the form of generated action potentials is highly stereotyped. That is, all spikes have almost exactly the same form. As a result, spike forms probably do not play an important role in information transmission. We can therefore neglect much of the specific ion-channel dynamics in the HH-model. Instead, we only focus on the spike-time dynamics and the effect of the leakage channels. The resulting differential equation of this so-called *leaky integrator* runs as follows:

$$\tau_m \frac{dV}{dt} = E_L - V + R_m I_e \tag{4.7}$$

where R_m is the total resistance of the membrane. The input currrent $I(t)$ is the sum of synaptic currents from presynaptic cells, which depends on the firing time of the presynaptic neuron of synapse j, the *synaptic efficacy* w_j of the synapses and the α-function that describes the stereotyped response:

$$I(t) = \sum_j w_j \alpha(t - t_j) \tag{4.8}$$

The firing time of the postsynaptic neuron is then determined by a threshold θ such that whenever V reaches that threshold, a spike is produced and the potential is reset to V_{reset}. Equation 4.7 indicates that when $I_e = 0$, the membrane potential approaches $V = E_L$ with time step τ_m. Hence, E_L is the resting potential of the neuron. The membrane potential is determined by integrating equation 4.7 and applying the threshold and reset rules.

These equations describe the *leaky integrate and fire model*, which is more commonly known as the IF-model. This model is the foundation for several other biologically plausible models, which add extra features–i.e., extra biological plausibility–on top of this basis. The IF-model is a simplified version of the HH-model that is extremely useful, because it is computationally not feasible to fully incorporate all our knowledge of every single parameter involved in a mammalian neuron. That is, if we were to do so, we would require a supercomputer with enormous computational power just to calculate the membrane potential of a single neuron. The simplifications do allow us to perform computations using larger groups of neurons, and to do so within a reasonable time-frame. Modeling will always be a matter of weighing your options and the choices you make have to reflect what it is you are trying to model. The point is, however, that as should be clear to anyone who is familiar with the connectionist paradigm, the neurons described through the IF-model are much more complex than the idealized neurons we find in connectionist models.

4.3 Biological Plausibility and Logic

Now, a hardline connectionist might still ask, what do we gain from the added biological plausibility? When the objective of connectionism is to study human cognition, the answer is, more biological plausibility in the models indicates that successful models are closer to the way human cognition functions. But that answer is trivial and may not be quite satisfactory, because there is much to learn from idealizing neurons and abstracting over their behavior in order to find out what properties of neurons and neural networks really matter for information processing. Thus, it should be stressed beyond all doubt that the thesis at hand is not meant to disparage connectionist models: au contraire, they remain highly valuable for cognitive science. The crux is, however, that one of the ways to find out which properties really matter is to study them in biologically plausible networks (see for a good example of this [93]). Furthermore, differences between connectionist and biologically plausible models may point us towards the limitations of the former in comparison to the latter. When Shastri and Ajjanagadde learned about oscillators in the brain, they took the basic approach of connectionism and turned the nodes into oscillators. The reason they did this was because the lack of oscillatory ability in traditional connectionism acted as a limitation on temporal binding.

Of course, studying biologically plausible networks does also have its disadvantages. The primary reason, and I surmise this is also the reason why relatively little research has been done on logic in combination with these types of networks, is the loss of control. Biologically plausible models are highly parameterized, as we shall see in what follows, and it is exceedingly difficult to grasp the dynamics, even when one is dealing with only a couple of neurons. However, as the work on temporal binding has shown and as Bader and Hitzler acknowledge in their survey, there is a lot of potential in the type of dynamics that emerges from these more complex models–and the territory is largely unexplored, particularly in the context of logic.

Which turns us to another question: why aren't computational neuroscientists interested in logic? One reason might be that most of the people that are interested in logic tend to have had a "connectionist upbringing". After all, connectionism has been around for quite some time. Another reason, that may have been an influence on the first, is that computational feasibility has long stood in the way of using more complex models. That, however, is no longer the case, as studies of other phenomena using these models have shown (see e.g. [60]). Another possible reason, which revolves around the common conception of logic and reasoning in the brain, is that it is a higher-level phenomenon. This view might be summed up as, either there is a "central logic processing unit" or we need an extremely complex network to model reasoning. Both views are explicitly denied by the thesis at hand, as we shall see in what follows. Forays into logic have been made from a computational neuroscience perspective, such as [93], but these are only preliminary results and do not directly relate to one of the most important problems to conquer first: the variable binding problem.

To sum up, the considerations above are taken to indicate that the study of logic on neural network suffers–to put it as a McCarthyism–from *"connectionist fixation"*. The discussion above is hoped to indicate the strong argument in favor of studying logic on computational neuroscience models as well, next to the ongoing research in connectionism.

Chapter 5

Methodology

The models in all experiments have been performed using the same software. For the actual neural simulation, the NeoCortical Simulator (NCS) was used [102][35], in combination with a frontend specifically designed for NCS, named Brainlab [23]. Furthermore, since spike train correlation is an actively researched field in computational neuroscience, there are some very good libraries available for performing such statistical analyses. The statistics library from the Brian simulator was used [32]. The advantage of this particular library is that it is written in python and hence easy to incorporate in combination with Brainlab. Furthermore, a big advantage is that the mathematics with relation to spike trains has been precisely described (in e.g. [12]). Next to these tools, several self-written analysis tools were used, all written in python and using the numpy, scipy and matplotlib modules. The full source code of all performed experiments, including detailed installation instructions for NCS and Brainlab, can be acquired online at https://github.com/dkiela/thesis. Also refer to Appendix B, which goes into more detail on this subject.

5.1 Spike-Time Correlation

Since we will be using spiking models, the synchrony in spikes provides a good measure of temporal binding between individual nodes. This is due to the fact that Hebbian learning–recall the motto "fire together, wire together"–will cause two neurons that are bound together to fire at related time intervals by heightening the synaptic efficacy of the postsynaptic neuron. Exactly how one measures synchrony in the behavior of neurons is an important topic in computational neuroscience, and several different approaches have been suggested (see [41] for a nice analysis). In order to keep our model as simple as possible, we will choose a relatively blunt measure of synchrony, namely spike train correlation. The rationale behind this choice is that if we obtain positive results using this measure, more finer-grained measures of synchrony will yield even better results.

Correlation is a much-used tool in computational neuroscience and has been actively studied (see e.g. [90][74][75][45]). Spike train correlation was already used as far back as the late 60s to analyze the peri-stimulus time histogram (PSTH) of neural assemblies, which was acquired through EEG scans [65][66].

A spike train is defined as a sum of Delta functions:

$$S(t) = \sum_{i=1} \delta(t - t_i) \tag{5.1}$$

where t_i is the time of the i^{th} spike. Thus, we use $S(t)$ to re-express sums of spikes as integrals over time.

The firing rate is the time average of S(t), i.e., the average number of spikes:

$$r = \langle S(t) \rangle = \lim_{T \to +\infty} \frac{1}{T} \int_0^T S(t)\, dt \tag{5.2}$$

We will denote the $\langle S(t) \rangle$ function with a subscript t to indicate that the variable t is bound by the integral and is not a free variable. To quantify the temporal relation between two spike trains, our first measure is a cross-correlation function (CCF):

$$CCF_{i,j}(s) = \langle S_i(t) S_j(t+s) \rangle_t \tag{5.3}$$

A better measure is a cross-covariance function (CCVF), which substracts the "baseline" of the cross-correlation function:

$$CCVF_{i,j}(s) = \langle S_i(t) S_j(t+s) \rangle_t - \langle S_i(t) \rangle_t \langle S_j(t) \rangle_t \tag{5.4}$$

Using this CCVF, we define the total correlation function for two spike trains i and j, which is what we will us as our measure of synchrony:

$$\lambda_{i,j} = \frac{1}{\langle S_i(t) \rangle_t} \int CCVF_{i,j}(s)\, ds \tag{5.5}$$

The spike-time correlation described here is exactly what the statistics library from the Brian simulator does, and allows us to perform binding through correlation.

5.2 The NeoCortical Simulator

NCS is one out of many spiking neural network simulators available (see [13] for an overview). It has seen several versions over the last decade, with the latest incarnation being NCSv5, which is specifically designed with parallel processing in mind (using MPI). The objective of NCS was "to create the first large-scale, synaptically realistic cortical computational model" (idem, p.2). NCS has an advantage over the two most popular biologically plausible simulators, NEURON and GENESIS, in that it uses the simplifications on the Hodgkin-Huxley model that were described above. Although NEURON and GENESIS have recently also developed into parallel simulators, the fact that they compute the

HH-model in its full details renders them unusable for large-scale experiments or repeated sets of experiments with many different parameters.

NCS models neurons in very close biological detail, including extensive control over the neuron's individual compartments; the membrane channels; synapse dynamics such as facilitation, depression and augmentation–and Hebbian spike-time dependent plasticity [85]. Particularly this latter property is important, because it allows us to apply Hebbian learning to the network on the level of spikes, as opposed to the simpler types of Hebbian learning we see in most connectionist models. Since NCS uses large ASCII files that describe all the different parameters, a minor change in the network's architecture requires a lot of change in the ASCII files. Therefore, a frontend was written in python which allows the setting of the parameters through libraries, which are converted to an ASCII file in NCS syntax when the simulation is run, called Brainlab [23]. The advantage of using python is that it is very well-suited for data analysis, with several actively maintained scientific analysis modules like numpy, scipy and matplotlib available as open source software[1]. Although Brainlab also features some rudimentary data analysis functionality of itself, the data analysis was done using self-written code using these scientific libraries, in combination with the spike train statistics library from the Brian simulator.

Because NCS is a cortical simulator for the study of the mammalian brain, it also allows for a 3D layout grouping where neurons and neuronal assemblies can be part of a designated cortical grouping, column or layer.

5.3 Parameters

The NCS networks that we have studied consist of localist representations, meaning that we use a single node to represent either a constant, variable, or predicate role. Hence, the number of neurons per simulation is relatively low compared to some other studies (such as [60]), but as we have discussed previously, this is not necessarily a problem at all. There are other good reasons for this: spike-time correlation is not particularly well-suited for the analysis of larger groups of neurons. Many different methods for analyzing the collective output of neural populations have been used, with the best-known one probably being the local field potential (LFP) [22]. However, this has the side effect of "flattening out" the spikes, which does not make the LFP suitable for a spike correlation analysis. It is possible to measure synchrony in LFP's, but these methods are much more computationally intensive than our correlation function. Additionally, since spike correlation is a relatively blunt measure, synchrony in LFP's, if anything, will yield finer-grained results than correlation. Meaning that when correlation turns out to be a successful measure the same will be the case for the more advanced synchrony measures. Every node is a leaky integrate-and-fire neuron characterized by a resting membrane potential

[1]Numpy and Scipy are available from http://www.scipy.org. Matplotlib is available at http://matplotlib.sourceforge.net.

$V_{rest} = -80$mV and a spiking threshold of -50mV. There are excitatory and inhibitory neurons. Inhibition has been incorporated in the more advanced models using an inhibitory pool of neurons, as opposed to distributing the inhibitory nodes internally, laterally and globally [16]. This abstraction is justifiable, because it keeps the model simple and removes the parameters for the placement of inhibitory neurons from the equation (see also [15]). The connection probability is set to 1 for all nodes, because we are not dealing with large randomly connected nodes: two nodes failing to connect results in undefined behavior of the model, because NCS unfortunately does not allow us to track which neuron was connected to which other neuron.

Each experiment consists of two stages: a learning stage and a testing stage. These stages are incorporated in the same trial, because of limitations in Brainlab. The typical simulation, varying on the model, lasts between 2 and 12 seconds, with Hebbian learning activated for 1 second, which is based on findings reported in e.g. [103].

In order for the network to become active, it has to be given a stimulus, which consists of a Poisson input spike train with a fixed firing rate. The stimuli are chosen such that it allows for spike-time dependent Hebbian learning to occur in the learning stage–i.e., the firing rate of two stimuli is such that stimulated neurons become associated, meaning that their synaptic efficacy with respect to each other increases. After the learning stage, one of the stimuli remains active, while the other stimulus drops. The correlation pattern is examined only for the testing stage, where the spike trains produced by two or more neurons is analyzed through the total correlation of their respective spike trains. In order to guarantee that the obtained results are caused by spike-time dependent learning, short-term dynamic plasticity was switched off.

The physiological parameters characterizing the behavior of the individual compartments, which are largely responsible for the leaky integrate-and-fire behavior of the soma [35], were taken from [103]. Based on the same findings, the absolute synaptic efficacy was set to 0.250.

Exploiting NCS's hierarchical organization, the neurons are assigned to the layer that represents their class, such that we have distinct layers for constants, variables and predicate roles. As we shall see, this hierarchy can potentially be expanded further to include truth-values and implication.

Whether or not two spike trains are correlated and whether or not a signal propagates in the network is dependent largely on three parameters:

- Excitatory conductance
- Inhibitory conductance
- Rate of one binding stimulus compared to that of another

The models and their results are presented in the following chapter.

Chapter 6

Models and Results

6.1 Model I: Testing Correlation

In order to verify that the correlation function works properly and that spike time correlation actually occurs in the setup of our choice, the first model consists of a simple experiment that shows some of the behavior that we would typically expect from such a function. It has been known in neuroscience that inhibitors are necessary to prevent a network from displaying erratic firing, i.e., going “on a rampage” [59]. This happens because excitatory synapses keep activating each other when they are recurrently connected without any inhibition taking place. As a result, we should see that as a trial lasts longer, or alternatively, as the excitatory conductance becomes higher over trials, the correlation should quickly reach a maximum and then drop, because even though the firing rates of the neurons increase, they become less and less related to each other. In order to test whether this is correct for our network, and our correlation function, consider the model in Figure 6.1. Random firing rates were chosen for the stimuli.

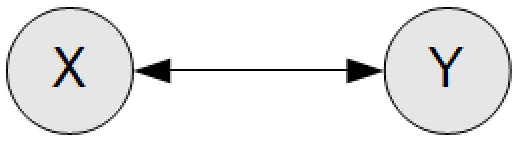

Figure 6.1: Correlation Test with Two Neurons

In a simulation run of 1,000 trials, with excitatory conductance increasing from 0.0 to 1.0 and an increase of 0.001 over every time step, we can clearly see that the hypothesis is correct. Even though the firing rate in Y increases as X is continuously stimulated, the correlation between the two neurons' spike times decreases.

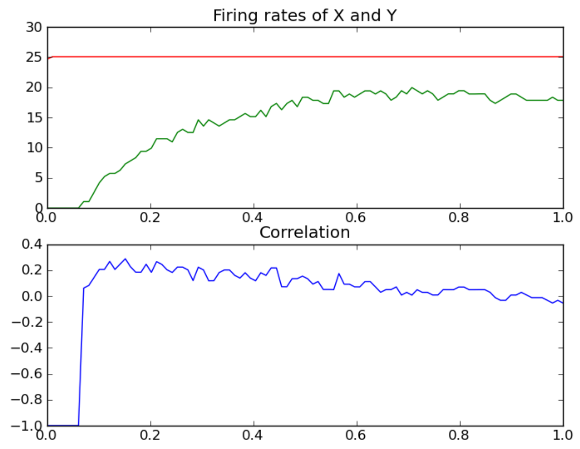

Figure 6.2: Firing Rate versus Correlation for Two Excitatory Neurons

For your benefit, all plots are reproduced in a bigger size in Appendix A.

6.2 Model II: The Impact of Inhibition

Thus, it is clear that inhibitory neurons are quite necessary, to stabilize the pattern and to ensure that the correlation that we find is in fact because the spike trains are correlated. Of course, even random firing patterns can show some correlation, so we want to make sure that we exclude this possibility.

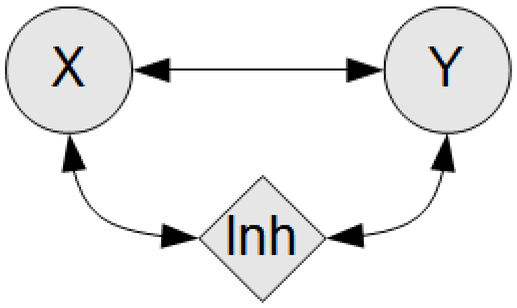

Figure 6.3: Excitatory - Inhibitory Test with 2:1 Ratio

For a fixed excitatory conductance and varying inhibitory conductance, we can already clearly see that the inhibitory conductance has a large impact on the correlation.

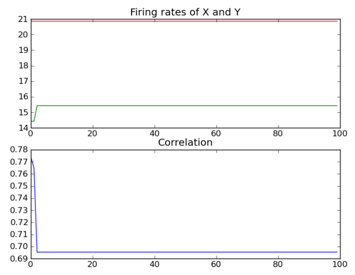

Figure 6.4: Impact of Inhibition of Firing Rates and Correlation

In a simulation run of 10,000 trials, the total correlation was examined with varying excitatory and inhibitory conductances. Typically, the inhibitory conductance has to be one order of magnitude larger than excitatory neurons [93][94], so the range of the excitatory conductance was between 0.0 and 0.1, whereas that of inhibitory conductance was 0.0 and 1.0, with 100 time steps each.

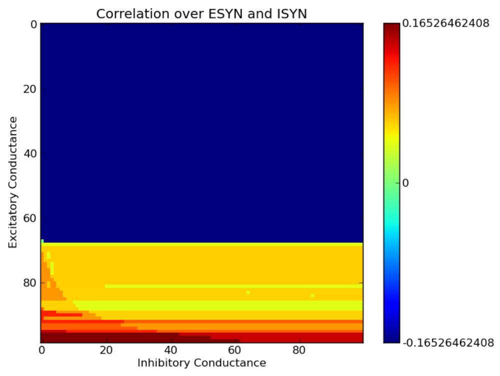

Figure 6.5: Average Correlation Difference

We can see that the inhibitory neuron and inhibitory conductance have a large impact on the correlation. In a large segment of possible combinations, there is no correlation whatsoever. This is the blue part of the graph. The reason appears to be that the excitatory conductance there is not high enough to overcome the inhibition. Only when the excitatory conductance value becomes large enough, do we see correlation. The higher the excitatory conductance, the higher the correlation, as is indicated by the color shifting from yellow to dark red. We can conclude that the inhibitory conductance plays an important role in correlation: without inhibition, as we saw in the previous model, the correlation decreases as excitatory conductance goes up and as the network starts to fire erratically; with inhibition, the correlation is kept in check.

It seems likely that an increase in the number of excitatory neurons will also have an impact on this. The excitatory-inhibitory ratio varies between 4:1 and 10:1 in the literature [35][93]. Since Vogels & Abbott use the 4:1 ratio and the simplest representation of a binary predicate requires at least 4 neurons, we will choose that ratio.

6.3 Model IIIa: Conductance

The model for the more advanced case where we maintain a 4:1 ratio and represent the predicate *Loves* for John and Mary is very similar to what we saw in the chapter on the binding problem. Rosenblatt's example used Triangle, Square, Green and Blue, which is identical to a case where we use John, Mary, Lover and Lovee. Thus, showing binding in this instance of the model solves Jackendoff's first case of the binding problem.

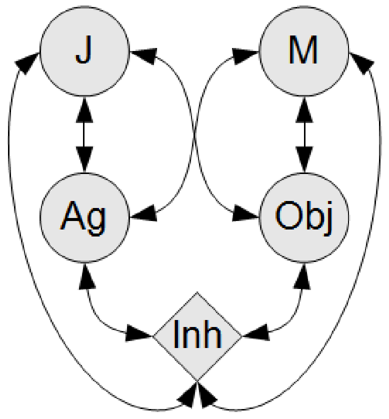

Figure 6.6: Representation of John, Mary, Lover and Lovee

The representation of a binary predicate John loves Mary then becomes as follows. John and Mary are both connected to the role-encoding neurons, which we have termed Agent and Object. We can stimulate John and Mary, or Agent and Object, and get similar correlation outputs since the connections are recurrent. This is important to stress, because this need not necessarily be the case in advanced models like this.

One might think that there should also be an inhibitor between the nodes for John and Mary. This is correct, in the sense that this would introduce the right type of inhibition for this case, but as we have mentioned above, the inhibitors in the models are used as an inhibition pool. Hence, all nodes in the network are connected to the inhibitor pool, which in this case contains only one inhibitory node. The reason for implementing inhibition this way is consistency: the placement of local, lateral and global inhibitors would add a sense of arbitrariness to the models and introduce even more parameters, which we've explicitly tried to avoid[1].

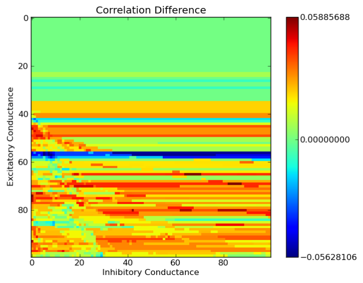

Figure 6.7: Correlation Difference for Conductance over John loves Mary

The same test of 10,000 trials was repeated for the network of four excitatory neurons, which shows that the ratio is indeed very important. The plot shows the average correlation difference between the taught bindings. For example, when we teach Agent(John) and Object(Mary), the correlation difference indicates how distinguishable these facts are from their alternatives. In other words, it is a measure of correctness of the correlation.

Interestingly, we can see in figure 6.7 that the output is rather peculiar: there appear to be horizontal lines drawn, indicating the the correlation stays the same for some excitatory conductance values, disregarding the inhibitory conductance. It is unclear why this is the case. Luckily, the highest correlation differences occur in places where this pattern is not found, so we can neglect it for our current purposes.

6.4 Model IIIb: Firing Rate

Another important factor in correlation is the firing rates we give to the stimuli. When teaching Agent(John) in the teaching phase, they are given the same firing rates in order to make sure that they bind through spike-time dependent plasticity. In order to avoid overlap between spike-times, the test rates for the trials were all prime numbers below 100, ranging from 2 to 97. The same measure of average correlation difference was used, in order to find an optimum firing rate. The chosen value for excitatory conductance and inhibitory conductance

[1]But see [15] for a thorough discussion of the effect of different types of inhibition on a neuron's oscillatory behavior.

over these trials was one of the optimal values for the previous model, namely an excitatory conductance of 0.085 and inhibitory conductance of 0.55.

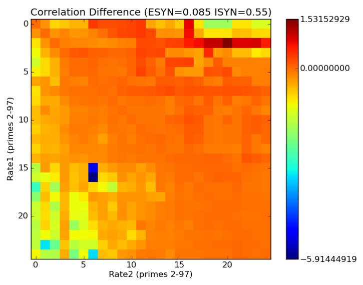

Figure 6.8: Correlation Difference for Firing Rate over John loves Mary

The blue and light green colors in figure 6.8 indicate no correlation or negative correlation difference. The dark red in the top right corner shows what values exhibit the highest correlation difference. The optimum rate thus turned out to be the 3rd and 20th prime, giving us 5 and 73 for the firing rates.

6.5 Model IV: Lover and Lovee

With the correct values found for this case, a simple version of the logical model was implemented in Python, so that we can teach the network facts and retrieve the information through queries. A sample run of the trivial programming language based on the logical model, indicates that the correct results are obtained. The responses are based on a comparison between the alternative possibilities. Because there are only two possibilities per case, no threshold value was necessary.

```
>>> from vb4 import *
>>> Loves(john,mary)
>>> run()
Done.
>>> Loves(john,mary)
True
>>> Loves(mary,john)
False
>>> Loves(john,john)
False
>>> Loves(mary,mary)
False
>>> Loves(john,X)
X = mary
```

```
>>> Loves(X,mary)
X = john
```

Unfortunately, limitations in Brainlab did not allow for the encoding of multiple facts at the same time, but experiments show that it is possible to encode for example Loves(john,mary) and Loves(mary,john) in the same simulation and acquire the correct correlation values to indicate that it is in fact the case that they love each other. Similarly, the correlations that have been found indicate that it is also possible to do something like:

```
>>> Loves(mary,X)
X = none
```

However, this can only be accomplished when we have determined a way to define a threshold which determines whether or not the correlation is sufficient to show full binding. There are several ways in which this threshold can be implemented, which is an important issue that we will address in the next chapter.

The results obtained for this model show that the network can successfully solve the first of Jackendoff's problems.

6.6 Model V: Little Big Star

Jackendoff's second problem is a little harder. First of all, there is the issue of how we measure correlation and between what nodes. When Little and Big both bind to star, but with different firing rates, we will be able to distinguish between them, but what about Agent(Little(star)) and Object(Big(star))?

The easiest way is to compare the total correlation values with each other. That way, when the difference between the correlation of Little(star) and Agent(star) is similar, the network encodes Agent(Little(star)). The similarity between the correlations is indeed apparent from trial runs, as we can see in a sample output with excitatory conductance of 0.085, inhibitory conductance of 0.55 and firing rates of 5 and 73 for the bindings, respectively:

```
>>> vb5.run()
Little star: 3.53700033637
Big star: 23.657132429
Agent star: 4.48106837019
Object star: 26.3286247906
```

Thus, Jackendoff's second problem can also be resolved. One objection, of course, is that having similar correlation values does not necessarily imply that the correct result has been obtained. After all, when are the correlations

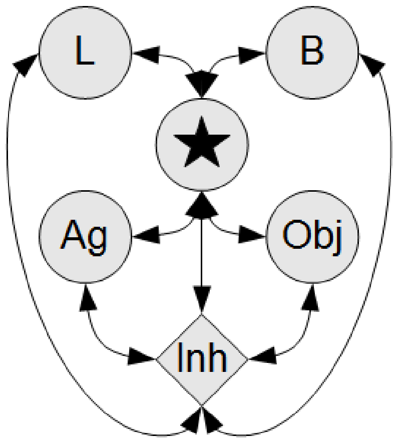

Figure 6.9: Binding over the Little and the Big Star

similar enough to say that binding has occurred? Since we are using very simple networks, this issue is not very apparent here, but it will become more important once the network is used to encode more facts.

6.7 Model VI: Single Binary Predicate Variable Binding

Although model IV successfully encodes the predicate Loves(john,mary), that is not the way one would ultimately want to encode a predicate. Whatever way we choose, the possibility of instantiating variables in predicates at a later state is of pivotal importance, because otherwise we will never be able to encode rules using our predicates. One way to do this would be to have multiple constants bound to the roles of the predicate, but further thought shows that this is not enough: we need to be able to re-use variables over different predicates. Something like $Loves(X,Y) \rightarrow Loves(Y,X)$ cannot be encoded if we just use the predicate roles. What we need, is instantiable variables that constants can bind to.

It is easy to see that this model is very similar to the previous one, except that we have two variables in the middle, as opposed to the single star. As it turns out, the results are the same as what we saw for the star, so that we can encode Agent(X), X=john and Object(Y), Y=mary. The exact values exhibit similar characteristics to what we have found in the previous model, and are ommitted for the sake of brevity.

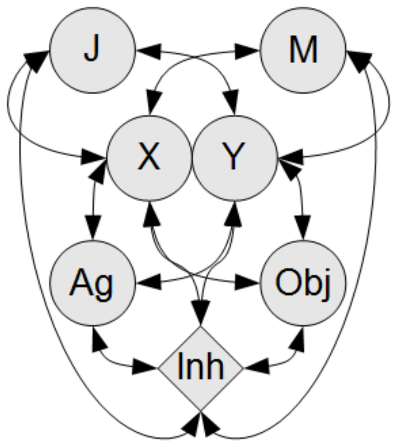

Figure 6.10: The Complicated Case Using Actual Variables

6.8 Model VII: Multiple Predicate Variable Binding

The real test for variable binding is, of course, whether we can in fact re-use the variables. In order to check whether this is the case, we encode two binary predicates, bind variables to their roles and bind constants to these variables. It is important to note that this does not necessarily constitute any particular logical connective, we are just encoding two predicates in the same network. For the sake of clarity, let us call these predicates Loves and Hates, and encode Loves(X,Y) and Hates(Y,X).

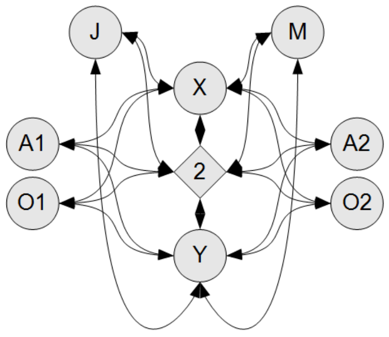

Figure 6.11: Two Predicates in One Network

The initial excitatory and inhibitory conductance values that we used for previous experiments yielded very different results in this network. This is presumably due to the fact that there is a large number of recurrent excitatory connections. Instead of simplifying the model so that it is more likely to reach an equilibrium (for example, by making it a recurrent feed-forward network), an attempt was made to find the optimal values for this network.

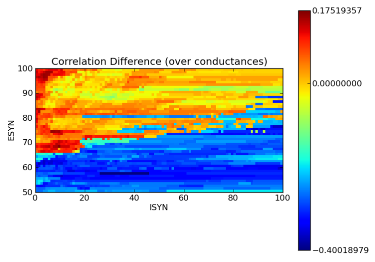

Figure 6.12: Correlation Difference over Conductance

A simulation run of 5,000 trials was done, with varying excitatory and inhibitory conductance values, like in Model IIIa, but only with the higher half of the excitatory values. The plot in figure 6.12 was vertically inverted to make clear that we are not starting from 0, but going up to 0.1 from 0.05. The firing rates were set to 5 and 73. The results show that the difference in correlations is much lower than in the simpler case, as one might expect.

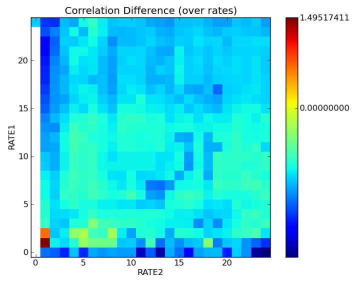

Figure 6.13: Correlation Difference over Firing Rates

However, the results clearly show that it is possible to have the correlation differences such that they are distinguishable, only to a lesser extent than what we saw in the earlier cases. A series of trials with fixed conductances and varying rates showed that the correlation between John and X together with X and Agent1 is only apparent for a few rates. We can see in figure 6.13 that only a few firing rates qualify, and that–for the chosen excitatory and inhibitory conductance values–they cannot be flipped so as to encode two different facts using the same firing rates, as was the case in the previous models. It is still possible to encode different facts, but not in such an elegant way.

There are a lot of ways in which this research can be expanded, as will be discussed in what follows. However, the important point here is not finding the optimal values for which correlation is the best tool to determine binding. The current purpose is to show that it is possible to use correlation as a measure of binding on biologically plausible neural networks. As the above results show, that is most certainly possible, so our goal has been achieved.

Chapter 7

Discussion

Variable binding is one important problem, but as goes without saying, it is only the beginning. In order to get clear what any full approach would look like, it is fruitful to highlight some of the features of what has been introduced in the thesis at hand and discuss the obtained results in a little more detail.

7.1 Conception of Logic

As some logicians jokily say to each other, “logic is everywhere”. In fact, logic may be found even in one single neuron: consider a spiking neuron with a firing threshold that requires at least two spikes in close temporal proximity. A single spike alone is not enough to initiate a spike. Only if two spikes are present within the required proximity can a postsynaptic spike be triggered–which corresponds to a logical AND function ([88], p. 85). A more detailed study of logic gates, notably including XOR and a flip-flop circuit, was done by Vogels & Abbott [93]. They discovered that given particular physiological parameters pertaining to the synapses, logic gates can spontaneously form anywhere in a randomly connected neural architecture. Furthermore, these logic gates can be used to propagate their signal over large sequences of neurons. To paraphrase these results in more logician-friendly terms: there does not need to be a central logic processing unit, nor does logic processing require enormously complex networks.

At least, whether you agree with that depends on your conception of logic. The distinction that Shastri and Ajjanagadde make between reflexive and reflective reasoning can be of help here. They explain what they take reflexive reasoning to be through the folktale of Little Red Riding Hood (LRRH) and the Big Bad Wolf. Unbeknownst to LRRH, the wolf has followed her into the woods and is about to attack her. The reader then reads: “The wolf heard some wood-cutters nearby and so he decided to wait” ([79] p. 1). Human beings understand effortlessly what has happened and why the wolf decides to wait, but underlying this inference is a chain of (unconscious) reasoning that leads to the conclusion:

> To eat something you have to be near it, so the wolf has to approach LRRH. LRRH will scream upon seeing the wolf, because she is scared. The wood-cutters will hear this scream and come to the rescue of the child. The wood-cutters will try to prevent the wolf from attacking LRRH. In doing so they may hurt the wolf physically. The wolf does not want to be hurt physically. So, he decides to wait.

This type of rapid, spontaneous reasoning without conscious effort is what Shastri and Ajjanagadde call reflexive reasoning. Reflexive reasoning is contrasted with reflective reasoning, which requires reflection, conscious effort and "an overt consideration of alternatives and weighing of possibilities" ([79], p.1). A good example of the reflective kind of reasoning is solving a cross-word puzzle in the newspaper. If we understand logic as reflexive reasoning–and it is a logical inference after all–a "locus of logic" suddenly becomes a lot more improbable, if only for the constraints this would put on how rapidly we can reason. Similarly, if such a network would be enormously complex, rapidly and spontaneously processing the LRRH-inference would be complicated. Thus, there is even an evolutionary argument for this conception of logic, namely that rapid inference is more likely to lead to survival: reflectively drawing the inference "wolf-dangerous-run" would be potentially lethal. It is precisely this kind of logic that we are interested in, because it seems to be an important factor in general human behavior.

More evidence from this conception of logic comes from research in logic and the psychology of reasoning by Stenning and van Lambalgen [87], who have analyzed a range of instances where human reasoning deviates from what classical logic prescribes. They "claim for logic a much wider role in cognition than is customarily assumed, in a complete reversal of the tendency to push logic to the fringes" ([87], p. 347). Stenning and van Lambalgen's analysis of logic in this context is too sophisticated to explicate in full detail here, but suffice it to say that according to them information processing happens with reference to logical form, which consists of an idealized competence model that is "as such not directly applicable to the real world" ([87], p. 350) in combination with constraints–that is, hypotheses about the world–imposed on that competence model to determine the strictly underdetermined input. Because human beings never have access to all data, they perform a type of non-monotonic reasoning called closed-world reasoning in order to solve reasoning tasks. This process of arriving at logical form is called reasoning to an interpretation. Notably, this type of reasoning does not require awareness, as indeed is illustrated in one of the reasoning tasks that they closely examine (the suppression task).

On the basis of their analysis of reasoning, Stenning and van Lambalgen see a large role for logic in information processing, as it also provides "a good format in which to represent information-processing tasks which are not traditionally conceived of as logical" ([87], p.354). One reason for this is that they show how some of the strictly nonverbal reasoning tasks that they discuss can successfully be analyzed using competence models formulated using logical notions. Furthermore, they recognize that cognition is fundamentally concerned with setting goals and achieving them, which if it is to be successful requires a form of planning, which in turn requires causal information about the consequences of particular actions. Logic, especially non-monotonic logic, lends itself naturally to representing goal-oriented behavior, planning and consequences of actions.

The above considerations are meant to show that, under this conception of logic, it makes perfect sense to implement logic in a biologically plausible network. Logical reasoning networks can potentially be all over the brain, as Vogels and Abbott's results indicate. Reflexive reasoning is something that is the most likely candidate for this type of logical reasoning, as is fundamental cognitive behavior such as goal-orientedness, planning and closed world reasoning, which can all be captured in logic. The reasoning has to be rapid, spontaneous and does not require awareness. Thus, these considerations suggest that any implementation of first-order logic on neural networks should be as simple as possible and reproducible in any neural structure of significant complexity, without adhering to a locus of logic.

7.2 Logic Programs and Neural Networks

One may wonder why there has been such a focus on logic programs as the logic of choice. An easy answer would be that this is simply because logic programs have been used successfully in previous work, but that is not sufficient. It is true that logic programs have been used quite successfully, most notably in the works of Hölldobler and in Stenning and van Lambalgen's study of human reasoning. However, other logics have also been used, such as Pinkas' penalty logic [67], to name but one. Luckily, there are other reasons that make logic programs such a suitable candidate, which we will briefly address in what follows.

When artificial intelligence researchers first tried to implement human-like reasoning in artificial agents, they soon discovered that classical logic is far from the best candidate. Human reasoning must be defeasible if we want to be able to deal with the changes in our environment; we must be able to revise our beliefs. As Stenning and van Lambalgen phrase it: "credulous reasoning [where the hearer tries to accommodate the truth of all the speaker's utterances in deriving an intended model] is best modeled as some form of nonmonotonic logic" ([87], p. 176). This view is closely related to the conception of logic that we discussed previously: classical logic only arises from conscious reflection on human reasoning. Actual human reasoning is much more related to planning and goal-oriented behavior, driven by the evolutionary necessity of rapid reasoning towards a credible model of the environment. This strengthens the case for logic programs as the best candidate to model reflexive reasoning.

Logic programs are just one out of many non-monotonic logics. However, logic programming does have some decisive advantages over its competitors. The fact that it allows for having a declarative and a procedural side means that it is highly suitable to be used for knowledge representation and modeling planning strategies. Moreover, it is syntactically simple and it is computationally efficient: whereas other non-monotonic logics, such as Reiter's default logic [70] and McCarthy's circumscription [56] are mostly NP-complete, logic programming is in P [33].

Additionally, the type of non-monotonicity that is displayed by logic programs is in line with results that have been obtained about the non-monotonic behavior of neural networks. Kraus et al. [46] have developed a monotonicity hierarchy which allows us to make this point clear. The hierarchy provides positive definitions of five different families of consequence relations and provides representation theorems for all five, based on the proof-theoretic study of non-monotonic consequence relations first suggested in [28] on the one hand, and model-theoretic considerations for non-monotonic inference proposed in [80] on the other. Summarized in a table, the resulting hierarchy looks as follows:

System	**Model**	**Postulates**
C	cumulative	Reflexivity, Left Logical Equivalence, Right Weakening, Cut and Cautious Monotonicity
CL	cumulative loop	C + Loop
P	preferential	C + Or
CM	cumulative monotonic	C + Monotonicity
M	monotonic	C + Contraposition

The first system, C, corresponds to the system that Balkenius and Gärdenfors [8] used in their original paper which spawned the interest in non-monotonic logic and neural networks. Leitgeb has developed representation theorems for both neural networks and logic programs with respect to this hierarchy. Using dynamical systems theory, he proves that neural networks correspond to C and that a specific subset, so-called hierarchical neural networks, belongs to CL [50] (see [49] for the full proofs). In the same paper, Leitgeb shows that logic programs can also be viewed as interpreted systems–the type of dynamical system that corresponds to CL. These results are especially interesting for our current discussion because they fit into Leitgeb's program to investigate what he calls *inference on the low level* [48].

However, there are some problems when it comes to what Leitgeb shows with relation to logic programs. Whereas it seems to be a good idea to approach neural networks from the perspective of dynamical systems, the same is not necessarily true for logic programs. Furthermore, the semantics that Leitgeb uses in his proofs unnecessarily restricts the class of logic programs that is applicable. What is more, Leitgeb pays no attention to the fact that for logic programs, the CL and P classes coincide. The author of the current thesis has addressed these issues in a previous paper [43], where it is shown that logic programs correspond to CL in the same way that hierarchical neural networks do. It needs to be noted that this is restricted to so-called normal logic programs, sometimes called general logic programs, because these are in fact non-monotonic. The approach differs from Leitgeb in that it proves the fact for a larger set of logic programs. While his approach is restricted to stratified logic programs and uses answer set semantics, it is possible to obtain the same result for all normal logic programs using well-founded semantics. Coincidentally, this three-valued semantics for logic programs is exactly the same semantics as is used by Hölldobler and Stenning and van Lambalgen.

As we can see on closer inspection, the hierarchy goes all the way up to cumulative monotonic and monotonic logics. This last level is the level where classical logic is at. Thus, it is important to note that logic programs represent a fragment of classical logic. A non-monotonic fragment, but also one that allows for a different type of semantics, based on unification. The advantage of this, in the context of neural networks, is that we do not require an external domain. As such, assignments of constants to variables reduce to identity relations between constants and variables. As a result, unification for logic programs in neural networks reduces to a problem of combinatorial optimization.

Another property of logic programs that is useful when trying to implement logic on neural networks is that the consequence relation is nicely modeled through connections in neural networks. In biologically plausible networks, the synaptic efficacy determines the strength of the connection and hence the strength of the implication. However, the story is slightly different for the other logical connectives: how about negation and conjunction? As we will see in a bit, negation is a problem that can relatively easily be solved. The story is different for conjunction, because the typical logic gates that Vogels and Abbott found [93] are not applicable for a three-valued logic. That topic, however, is out of the scope of the current thesis.

7.3 Incorporating Rules

Let's focus on the representation of truth values first. Luckily, work on this has already partially been done, specifically with relation to non-monotonic consequence relations. David Neville [62] has worked on modeling this type of three-valued consequence relation and showed that it is possible to successfully model non-monotonic relations. Importantly for the current argument, this was accomplished on biologically plausible networks using NCS. The model that Neville used looks as follows:

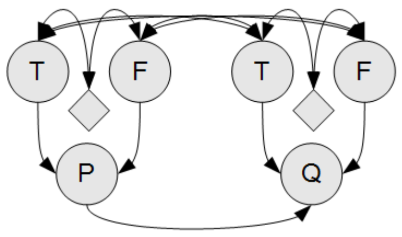

Figure 7.1: Neville's Model of a Non-Monotonic Consequence Relation

Neville used assemblies of neurons and showed that the consequence relation between P and Q tracks the truth values. As such, it is possible to make sure that Q is only true if and only if P is true. The unknown truth value is modeled when neither T nor F is active. Since Neville's model was implemented on the same software, NCS, a suggestion of how we could model implication and this sort of non-monotonic consequence would look as follows[1]:

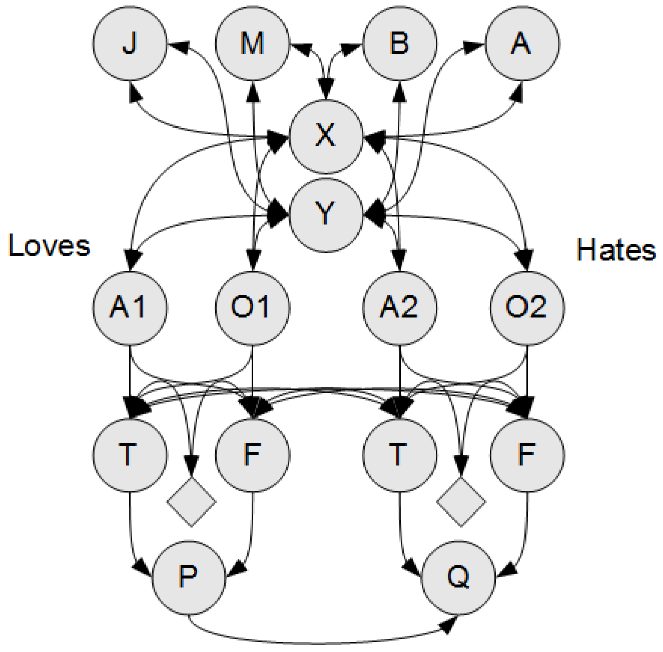

Figure 7.2: Variable Binding with Neville's Model

Whereas the two-predicate network ran into the trouble of expressing the exact relation–whether it be a logical connective, just separate predicates, or something else–between the two predicates, Neville's work provides us with a way out. The correlation determines the binding between the constants and the variables, while the actual implication happens at the lower levels. On closer inspection, this model actually is a good way to show how important unification is for our current purposes, because it provides us with a great advantage: the actual implication and the inference resulting from it is separate from the variable binding. Furthermore, explicitly encoding truth-values like this allows for correlation to determine which bindings are true, false or undecided: e.g., when True correlates with john, x and agent$_1$, it is true that John bound to x

[1]An inhibition pool for all neurons, not just True and False, is implicit here. It is without a doubt an important part of the model, but it was disregarded in the picture in order to make clearer the interconnections between the nodes.

as the agent is true for that predicate.

7.4 Determining the Threshold

Using this information we could attempt to formulate a logical model where we provide a semantics based on biologically plausible neural networks–a neurosemantics, if you will. However, such an attempt is beyond the scope of the thesis at hand, because it requires us to make explicit how we are to define a threshold θ that conclusively indicates binding–something that is worth pursuing, but that easily merits a thesis of its own right. In what follows, we briefly review some of the intricacies involved.

First and foremost, it is hard to come up with a threshold that is so generic that it can be applied to any model whatsoever that uses spike-time correlation. In the experiments, the threshold was mostly unnecessary, because we were aware of all the possibilities, but this will not always be the case when the experiments are scaled up. Without attempting to define the threshold in a general way, we can indicate what factors might play an important role here[2]:

- Connections: when scaling up, the connection probability between individual neurons within the same layer, inbetween layers or inbetween colums, will have a direct impact on the correlation threshold.
- Conductance and firing rates of stimuli: the conductance values and the firing rates given to stimuli have a great impact even on the simpler models. Changing the conductance only slightly, as we have seen, can already change the correlation outcomes significantly.
- Learning: the impact of learning on correlation may be used as a base-case threshold. That is, one can measure the correlation when there is no learning, and use that as an index for correlations where the network has learned a specific relation.
- Time: the time of learning and the time of the total experiment may cause fluctuations in the correlation, depending on the parameters. When the time of learning is too long, the network may overlearn. When it is too short, the correlation may not change significantly enough. When the length of the experiment is extended, the correlation should remain identical, as long as the learning phase stays the same. However, since there is at least some randomness involved in any biologically plausible model, the length of the experiment may also cause the correlation to fluctuate slightly.

[2]An additional complicating factor in determining the threshold is due to the some peculiarities of the NeoCortical Simulator. With a connection probability lower than 1 NCS does not allow us to determine after the fact which neurons have connected with which other neurons. Furthermore, there is a certain randomness introduced which cannot be avoided. Since these and other factors make it very difficult to quickly determine a qualitatively sound threshold, it is better not to go into too much detail on that and leave it for future work.

It is important to note that these factors have been taken into account in the experiments performed for the current thesis. Connection probabilities were set to 1 and they were always recurrent; the conductance values were tested against each other; learning and the time of learning were made such that Hebbian learning in the model was only active when both assemblies were simultaneously stimulated. The ratio of 1 second learning phase followed by 1 second processing phase was chosen deliberately on the basis of previous experiments [103]. All in all, the models were chosen such that if it works for these models, then the results are automatically applicable to finer-grained models where these parameters have been chosen differently. For example, a recurrent feed-forward network with the correct conductance values; exactly the right amount of learning; and precisely the correct ratio between inhibitory and excitatory neurons and their interconnections will result in a much higher distinguishability in spike-time correlation.

7.5 Synchronization

Likewise, the measure of spike-time correlation was chosen exactly because *if* we can obtain results with correlation, then finer-grained measures will yield even better results. Specifically, the measure of synchrony is something that is very actively researched in computational neuroscience and even in physics. The Kuramoto model [3] has been used already in examining the synchrony between neural assemblies on EEG scans. EEG scans are notoriously imprecise, meaning that the Kuramoto synchrony is measured over the local field potential of not-necessarily related neurons [19]. That is, the assemblies are arbitrarily based on the placement of sensors. This is something that computational neuroscience need not suffer from: we can very precisely calculate the local field potential over designated assemblies, knowing what they are supposed to represent.

Chapter 8

Conclusion & Outlook

The purpose of the current thesis has been to show that biologically plausible neural networks can implement variable binding on (multiple) predicates. The results are preliminary, in the sense that the thesis makes the modest claim that more attention should be paid towards uniting our knowledge of logic with our knowledge of computational neuroscience. The current work can be used as a foundation for implementing predicate logic programs on neural networks and shows the way towards achieving that goal. Predicate logic programs are especially important, because they exhibit exactly the right properties that we would want to have when we are implementing logic on neural networks.

It has been made clear that the conception of logic that is at stake here differs significantly from classical logic: it is reflexive reasoning, low-level inferences, that we are concerned with. It is the belief of the author that this type of reasoning represents the fundamental type of reasoning and that it underlies cognition as a whole, as well as the reflective understanding of our own reasoning that ultimately led to classical logic.

The deliberately blunt parameters for connections, conductances, firing rates, inhibitory-excitatory ratios, learning times and the measure of spike-time correlation were chosen together in order to represent such a blunt tool that the obtained results quite definitely show first and foremost that there is a lot of potential in modeling logic on biologically plausible neural networks and, more importantly, that logic in combination with computational neuroscience is a very useful alternative next to the traditional study of logic in relation with connectionism.

Since the results presented here are merely preliminary, there is a lot of space for future work. The previous chapter has already touched upon some of the possibilities: a first step would be implementing truth values, implication and conjunction. Furthermore, tweaking the network according to our current neurophysiological understanding of the brain, as well as chosing a finer-grained measure of synchrony, may allow for even better results. That is, the blunt parameters can be made much more precise, allowing for greater differences in correlation for the desired cases.

Ultimately, the author is confident that this is the way forward. Some sort of biologically plausible neurosemantics which fully models normal first-order logic programs, is likely to be achieved through a combination of logic and connectionism on the one hand, and logic and computational neuroscience on the other.

Appendix A

Detailed Graphs and Plots

The graphs and plots used for the models are reproduced in this appendix in a larger size.

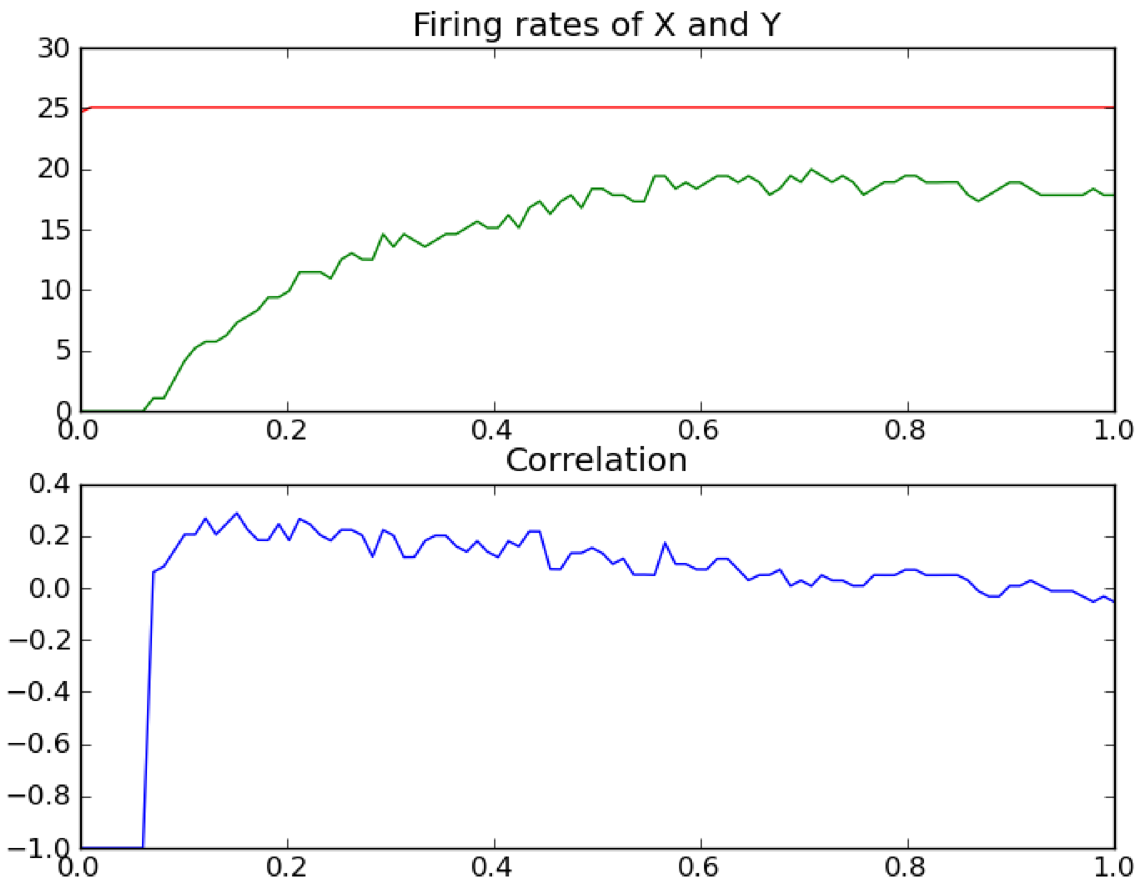

Figure A.1: Figure 6.2

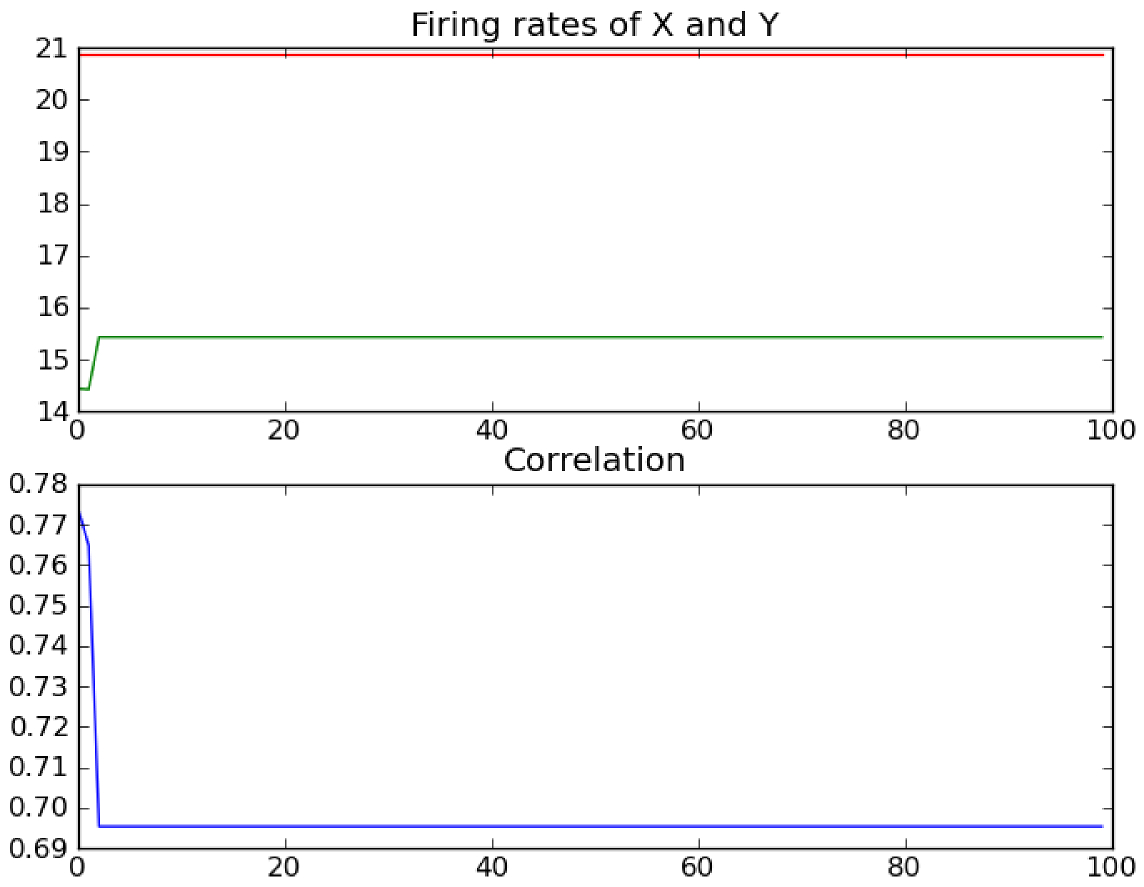

Figure A.2: Figure 6.4

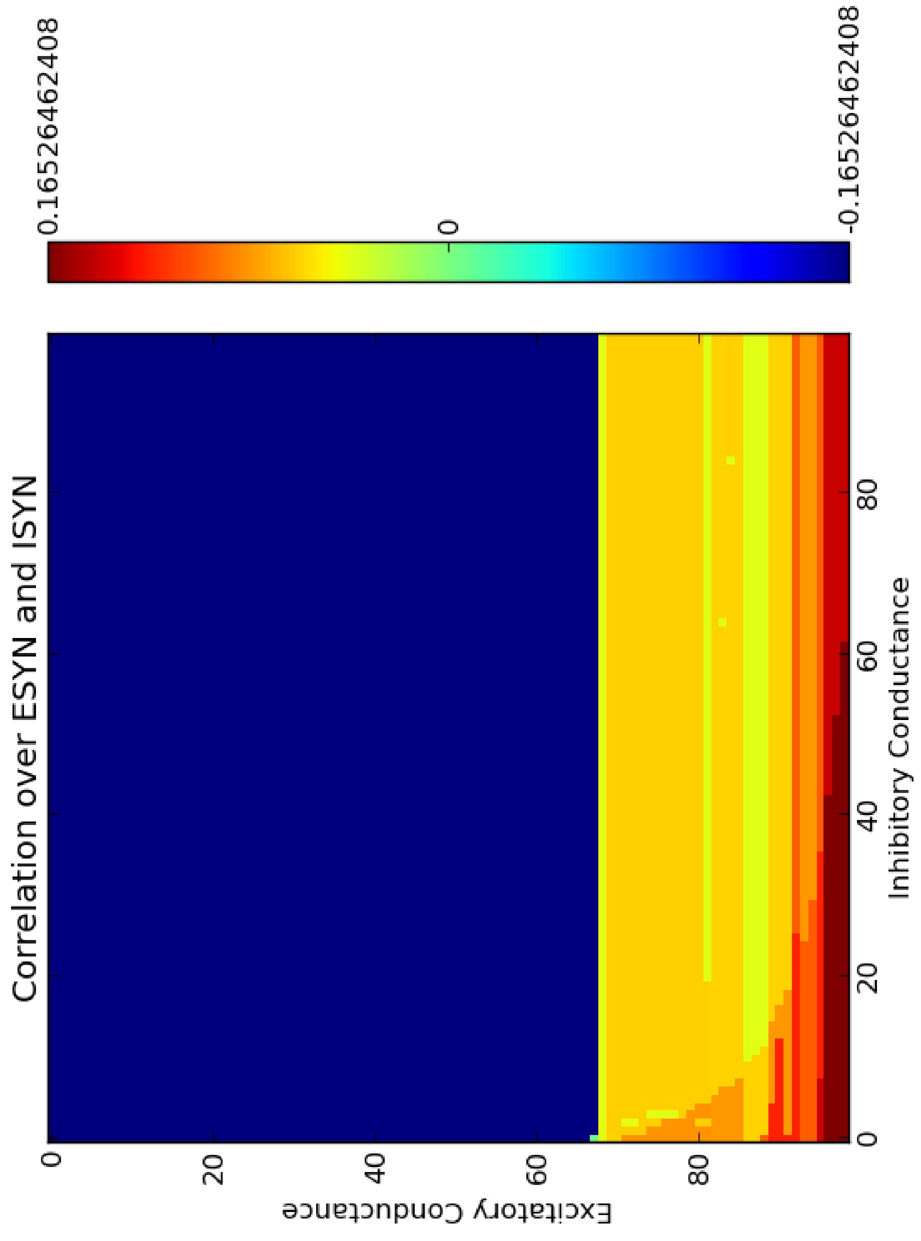

Figure A.3: Figure 6.5

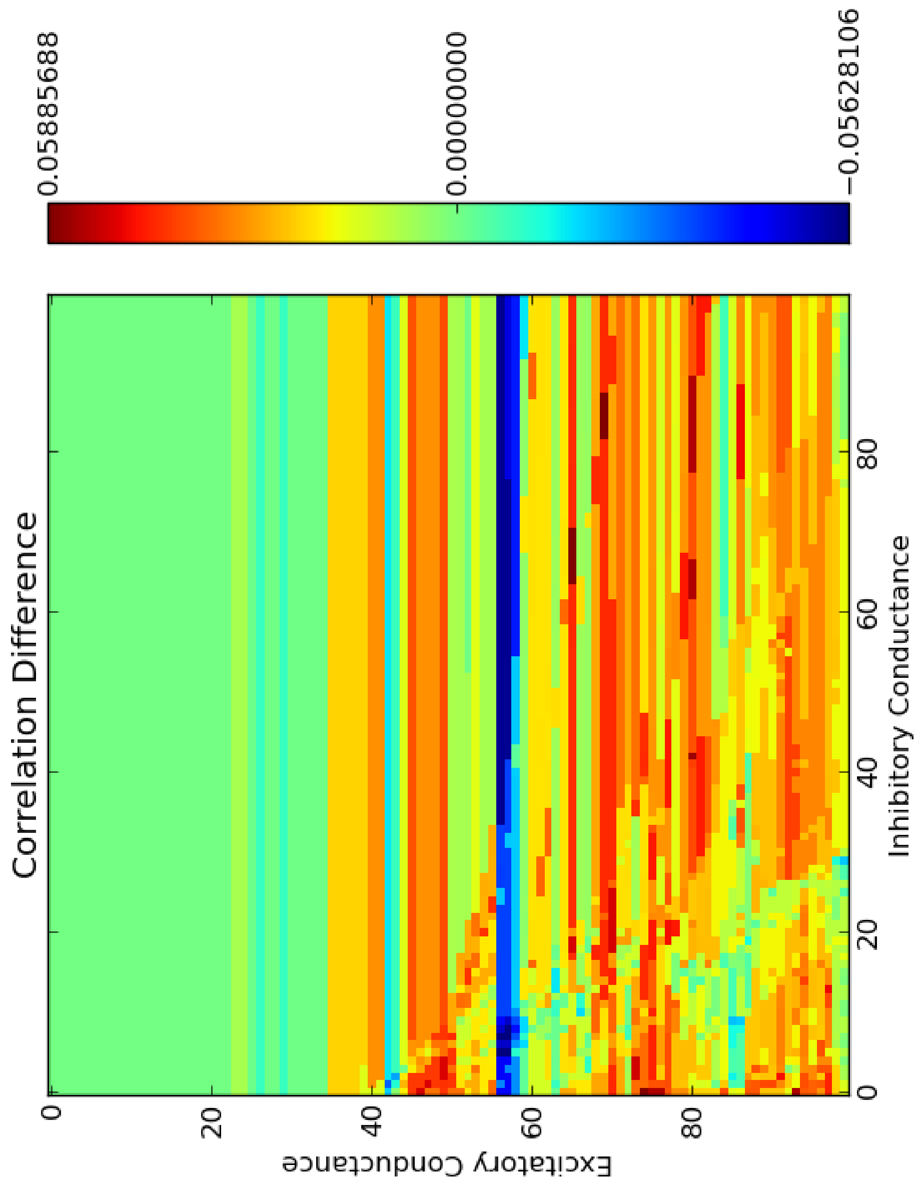

Figure A.4: Figure 6.7

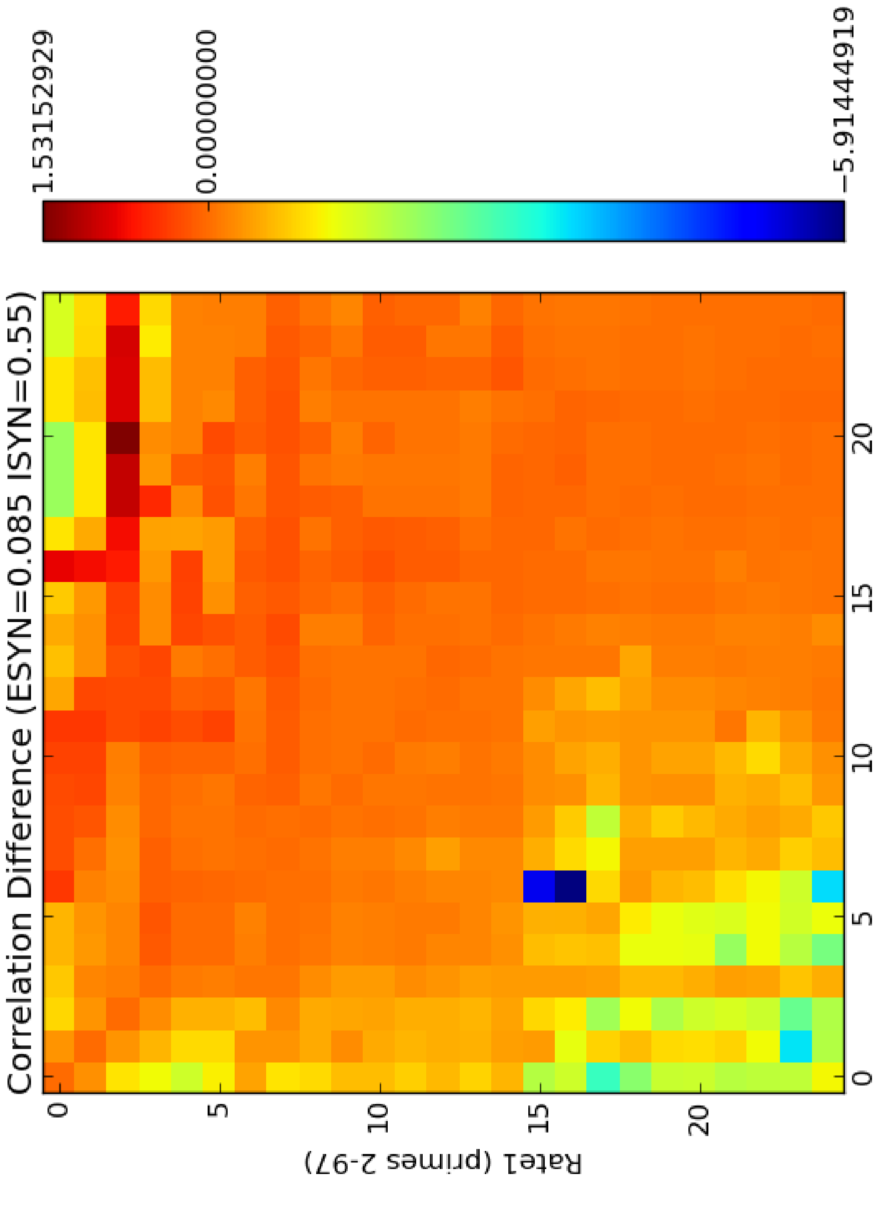

Figure A.5: Figure 6.8

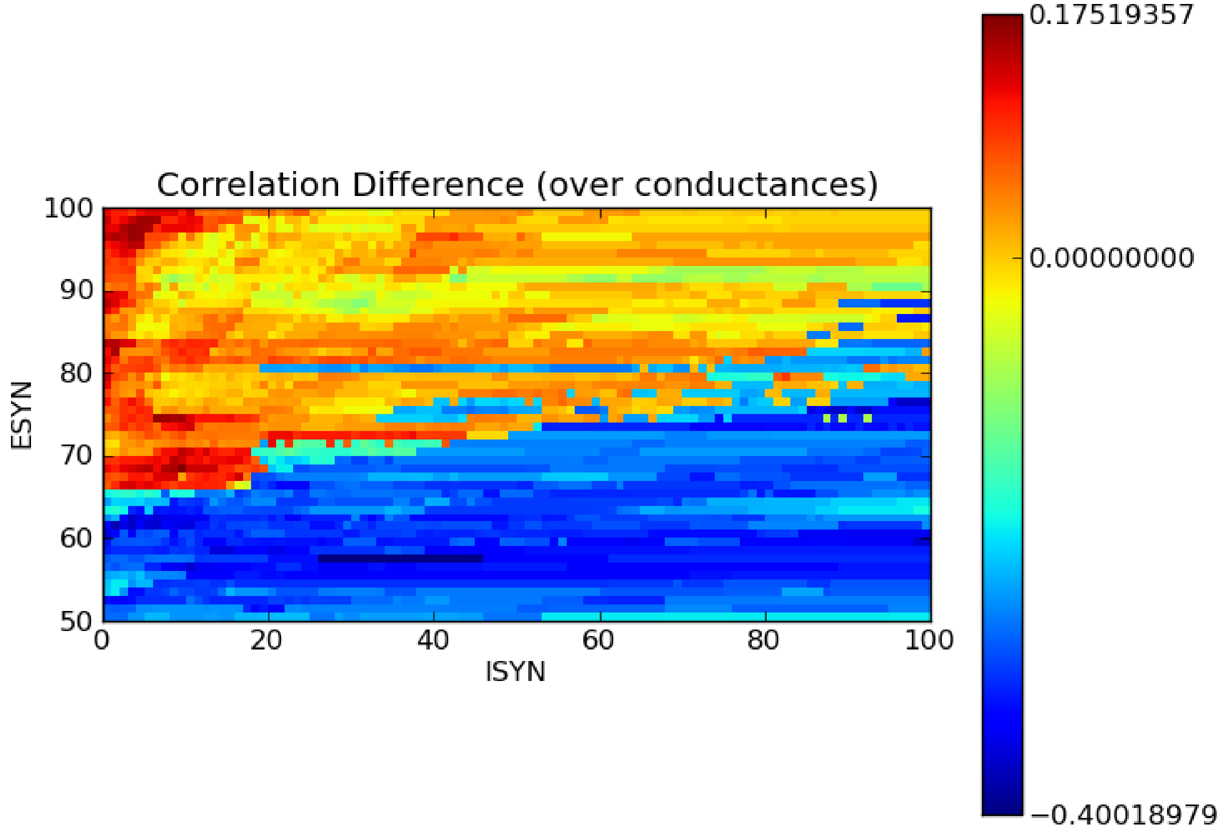

Figure A.6: Figure 6.12

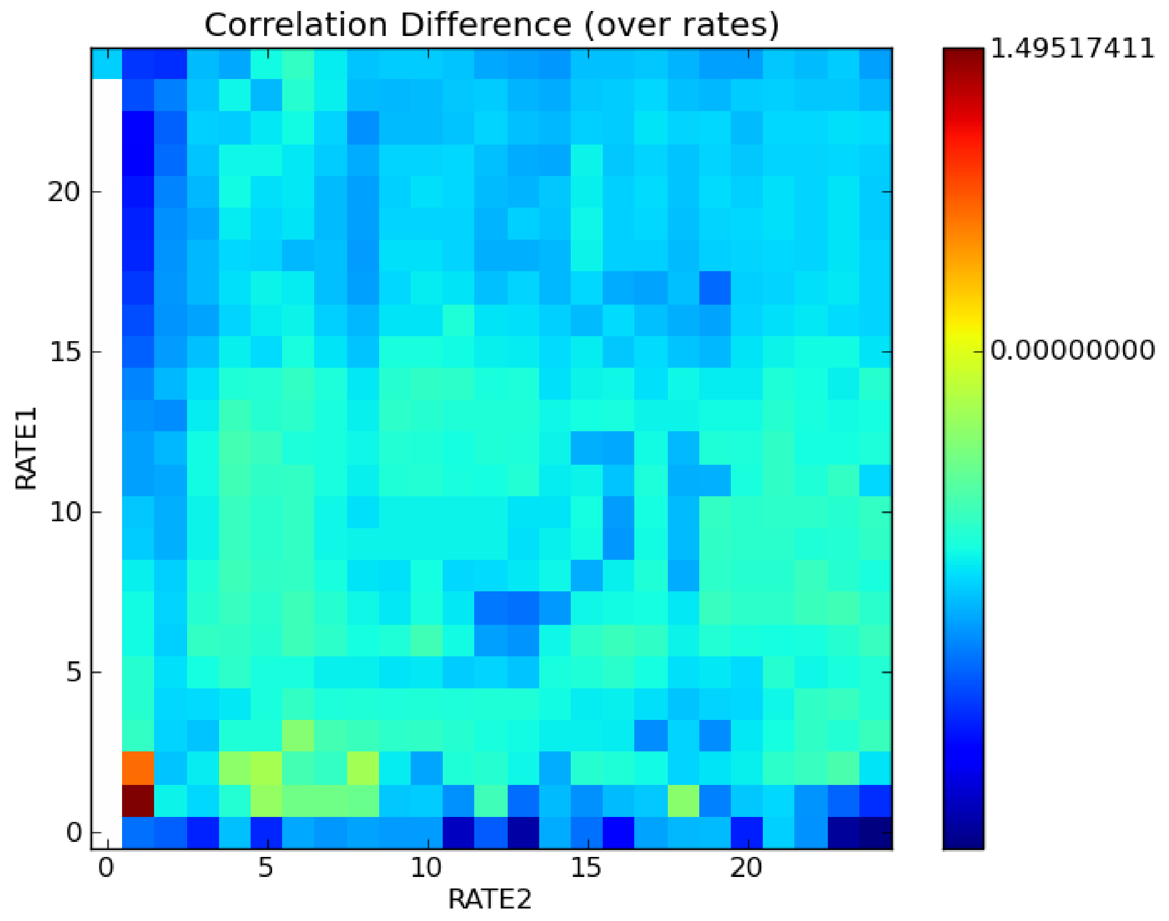

Figure A.7: Figure 6.13

Appendix B

NCS, Brainlab and Code from the Experiments

B.1 Installing NCS and Brainlab

From a fresh Ubuntu install, first install subversion and then use it to acquire the Brainlab package, which includes the NCS source code:

```
sudo apt-get install subversion
cd ~
svn co https://brainlab.svn.sourceforge.net/svnroot/brainlab brain-
lab
cd brainlab
tar xvzf ncs*
```

Then, install the required dependencies for NCS:

```
sudo apt-get install bison mpich-bin libmpich-mpd1.0-dev libmpich-
shmem1.0-dev mpich2 mpichpython flex g++
```

The next step is to compile NCS. Before we can do that, however, we need to edit the Makefile and adjust the following (if we are running it locally):

```
# apt-get installs MPICH in /usr/bin, so set this:
ifeq ($(SYSTEM),local)
MPI = /usr
NAME = ncs5pe
endif


# Make sure this is set:
SYSTEM = local
```

If you're on an AMD64 system, adjust the CFLAGS accordingly. Then compile the code and link to the executable so that Brainlab can find it:

```
make
ln -s ./ncs5pe ..
```

With NCS compiled successfully, we then turn our attention to Brainlab and add the following to .brainlabrc:

```
remoteexec=False
mcmd='/usr/bin/mpirun'
nolocal=False
remdir='./'
```

Install the required dependencies for Brainlab:

```
sudo apt-get build-dep python-matplotlib python-scipy python-scipy
sudo apt-get install python-matplotlib python-scipy python-opengl
python-pyx
```

We can then run the testsuite to see if there are any errors, and fix them in case there are:

```
python testsuite.py
```

Then, we need to make sure that MPI is running and configured properly. For running only locally, the easiest way is to run the following commands:

```
touch ~/.mpd.conf
chmod 600 ~/.mpd.conf
echo "MPD_SECRETWORD=S3CR3TW0RD" > ~/.mpd.conf
mpd &
```

We can then verify that MPD is running properly by doing:

```
mpdtrace
```

The hostname that this outputs should be put into the machinefile in /usr/lib/mpich/share/machines.LINUX. If you want to add multiple machines, repeat the steps and add additional hostnames to the same machinefile. To check that this works properly, run the following command to show your hostname:

```
mpiexec -n 1 /bin/hostname
```

With MPI configured, we need to fix some bugs in the Brainlab code. Change the appropriate line where the command is executed in brainlab.py to the following:

```
machfile="/usr/lib/mpich/share/machines.LINUX"

ncmd="cd "+tmpdir+"; "+mcmd+" -machinefile "+machfile+" "+nlcl+
" -np "+"nprocs"+" "+ncscmd+" "+
brainname+".in -d ./" # end quote after I/O redir
```

Also set appropriate values at the top, depending on your configuration. Next, make sure the IPSC.txt and EPSC.txt files are in your home directory and you should be good to go.

In case you run into any errors, notably segmentation faults with running reports, you might have to set the appropriate library dependency, like so:

```
export LD_PRELOAD=/usr/lib/libstdc++.so.6
```

B.2 Understanding NCS and Brainlab

Running simulations in Brainlab comprises four main steps:

- Setting the parameters
- Constructing the network
- Running the simulation
- Analyzing the results

In this appendix we will briefly go over some of the code used in the experiments, in order to make it easier for others to reproduce the results and build on top of the results obtained in the thesis at hand. Refer to [35] for a more detailed list of available options and a more detailed explanation of the possibilities.

B.2.1 Setting the parameters

Before we initialize a so-called BRAIN object, we need to tell the constructor the length of the simulation, the name of the "brain" and the simulation timesteps per second. We can then initalize the object:

```
bradb=brainlab.BRAIN(simsecs=2.0, jobname="experiment1", fsv=10000)
```

The resultant object consists mostly of a comprehensive library that spans all the possible parameters. For example, if we want to change the spiking threshold for a particular neuron's compartments, we can change the parameters as follows:

```
lib=bradb.libs['standard']
comptypes=lib['comptypes']
comp=comptypes['SOMA1']
comp.parms['THRESHOLD']=['-50,0.0']
```

The most important parameters concern the excitatory and inhibitory synapses. For example, if we want to set the Hebbian learning of the excitatory synapses for one second, starting from the beginning:

```
lib=bradb.libs[‘standard’]
syntypes=lib[‘syntypes’]
esyns=syntypes[‘E’]
esyns.parms[‘HEBB_START’]=[‘0’]
esyns.parms[‘HEBB_END’]=[‘1’]
```

B.2.2 Constructing the network

Once all the parameters are set, we can start constructing the actual networks. The associated objects here are COLUMN, LAYER and STIMULUS. One column can contain multiple layers. When applying a stimulus (a so-called stimulus injection), we provide the column name, the layer name, the synapse type, the cell name and the number of cells being stimulated, in that order.

Connections can be formed between layers in different columns, layers in the same column and between cells within the same layer. This is best explained through a simple example that contructs a column, adds a layer to it, connects the cells in the layer to each other and applies a stimulus:

```
col=bradb.COLUMN(“TYPE”:“COL1”)
bradb.AddColumn(col)
layer=bradb.LAYER(“TYPE”:“LAYER1”)
layer.AddCellType(ecell, num_neurons=100)
col.AddLayerType(layer)
bradb.AddConnect((col,layer,ecell),(col,layer,ecell), esyns, prob=1)
stim1=bradb.STIMULUS(parms=‘MODE’: ‘VOLTAGE’, ‘PATTERN’:‘NOISE’)
# you can set the parameters for stim1 here
stim_layer1=bradb.STIMULUS_INJECT()
stim_layer1.parms[‘STIM_TYPE’]=stim1
stim_layer1.parms[‘TYPE’]=‘inject_layer1’
stim_layer1.parms[‘INJECT’]=[‘COL1 LAYER1 E SOMA1_name 100’]
bradb.AddStimInject(stim_layer1)
```

Lastly, we can tell Brainlab what to report on. In most cases, we do not need all available data and we are only interested in specific information about one population of neurons, for example. For this reason, before we run the brain we tell Brainlab that we only want to get a report of the voltage in Layer 1 for the first two seconds:

```
bradb.AddSimpleReport(“Layer1Report”, (col,layer1,ecell), reptype=“v”,
dur=(0.0,2.0))
```

B.2.3 Running the simulation

In order to run the simulation, we pass our constructed network and associated parameters to the Run function. Here we can also tell it how many processes can be used, which can be important on multi-core systems or on clusters:

```
brainlab.Run(bradb, nprocs=1)
```

B.2.4 Analyzing the results

Brainlab itself provides some basic functions for the analysis of the results, but in most cases the functions are unsatisfactory for a number of reasons. If you want to analyze the results in more detail than looking purely at the voltage plots of one single neuron, it is best to familiarize yourself with the Python libaries *matplotlib* and *numpy*.

Before you can start using these libraries, you need to know how the datafiles that Brainlab/NCS generates are organized. This is in fact pretty straightforward:

```
0 -60.0877 -60.0894 -60.0911 -60.0947 -60.0874 -60.0879 -60.0897
```

The first number, 0, represents the timestep. Every column that follows it represents the membrane potential of one neuron, so that in the data above we can see the membrane potentials for 7 neurons at timestep 0. The best way to go about this is to load the entire file into a two-dimensional numpy array and remove the leftmost column (which represents the timesteps). We can then plot the results using matplotlib:

```
import numpy as np
import matplotlib.pyplot as plt
space=np.linspace(0.0,2.0,2.0*10000)
results_2darray=np.array(data)
plt.plot(sp,results_2darray,'b')
plt.show()
```

The fact that the information is stored in a numpy array means you can do advanced analysis of the whole dataset using functions in the *numpy* and *scipy* libraries. For example, in one of the experiments performed for the thesis, the whole sequence above was repeated 10.000 times for different parameters. The resulting datasets were analyzed for spike-time correlation one-by-one, which was in turn plotted on a 100 by 100 color plot to show where correlation was highest.

B.3 Code from the Experiments

In most of the more advanced experiments, particularly the repetitive ones that searched for the right parameters, an additional layer of code was used to make things more organized. Typically, the dynamic parameters were set from a loop, then passed to the simulation run in brainsim.py and then analyzed using functions from brainalyze.py. For example, checking the rates for two stimuli, we would run the repetitive experiments like so:

```
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59,
61, 67, 71, 73, 79, 83, 89, 97]
rates = np.array(primes)
x=0
y=0
while x < len(rates):
    while y < len(rates):
        parameters['RATE1']=rates[x]
        parameters['RATE2']=rates[y]
        parameters['BRAINNAME']='vb-exp'+str(rates[x])+'-'+str(rates[y])
        brainsim.simBrain(parameters)
        y=y+1
    y=0
    x=x+1
```

The resulting data files can then be analyzed from another loop, for example, storing the correlation between John and X for all possible rate values in a numpy data file which can be further analyzed or plotted at a later stage:

```
x=0
y=0
store = np.zeros((len(rates), len(rates)))
while x < len(rates):
    while y < len(rates):
        lJ=brainsim.loadBrain('vb-exp-JReport.txt')
        _,spikesJ=brainalyze.countSpikes(lJ,10000*(parameters['ENDSIM']/2))
        lX=brainsim.loadBrain('vb-exp-XReport.txt')
        _,spikesX=brainalyze.countSpikes(lX,10000*(parameters['ENDSIM']/2))
        cJX=brainalyze.corr(spikesJ,spikesX) # correlation John
and X
        store[x][y]=cJX
        y=y+1
    y=0
    x=x+1
np.save('ratestore.npy', store)
```

The full code of the experiments can be acquired from the following URL: https://github.com/dkiela/thesis and from the ILLC website.

Bibliography

[1] L. Abbott. Firing-rate models for neural populations. In *Neural Networks: From Biology to High-Energy Physics*, pages 179–196. ETS Editrice, Pisa, 1991.

[2] M. Abeles. Local cortical circuits: An electrophysiological study. 1982.

[3] J. A. Acebrón, L. L. Bonilla, P. Vicente, J. Conrad, F. Ritort, and R. Spigler. The kuramoto model: a simple paradigm for synchronization phenomena. *Reviews of Modern Physics*, 77:137185, 2005.

[4] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[5] S. Bader and P. Hitzler. Dimensions of neural-symbolic integration - a structured survey. In S. Artemov, H. Barringer, A. d'Avila Garcez, L. Lamb, and J. Woods, editors, *We Will Show Them: Essays in Honour of Dov Gabbay, Volume 1*, pages 167–194. International Federation for Computation Logic College Publications, 2005.

[6] S. Bader, P. Hitzler, and S. Hölldobler. The integration of connectionism and first-order knowledge representations and reasoning as a challenge for artificial inteligence. *Information*, 9, 2006.

[7] S. Bader, P. Hitzler, S. Hlldobler, and A. Witzel. The core method: Connectionist model generation for first-order logic programs. In *Proceedings of the ICANN'06*, pages 1–13, 2006.

[8] C. Balkenius and P. Gärdenfors. Nonmonotonic Inferences in Neural Networks. In J. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning*, pages 32–39. Morgan Kaufmann, San Mateo, CA, 1991.

[9] D. Ballard. Parallel logic inference and energy minimization. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, page 203 208, 1986.

[10] L. Barrett, J. Feldman, and J. Dermed. A (somewhat) new solution to the variable binding problem. *Neural Computation*, 20:2361–2378.

[11] R. Blutner. Nonmonotonic Inferences and Neural Networks. *Synthese*, 142:143–174, 2004.

[12] R. Brette. Generation of correlated spike trains. *Neural Computation*, 21:188–215, 2009.

[13] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, M. Zirpe, T. Natschlger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Viville, E. Mueller, A. P. Davison, S. E. Boustani, and A. Destexhe. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23:349–398, 2007.

[14] A. Browne and R. Sun. Connectionist variable binding. *Expert Systems: The International Journal of Knowledge Engineering and Neural Networks*, 16:189–207, 1999.

[15] G. Buzsáki. *Rhythms of the Brain.* Oxford University Press, Oxford, 2006.

[16] G. Buzsáki, K. Kaila, and M. Raichle. Inhibition and brain work. *Neuron*, 56(5):771–83, 2007.

[17] D. Chalmers. Syntactic transformations on distributed representations. *Connection Science*, 2:53–62, 1990.

[18] P. Churchland. *A Neurocomputational Perspective: The Nature of Mind and the Structure of Science.* MIT Press, Cambridge, MA, 1989.

[19] D. Cumin and C. P. Unsworth. Generalising the kuromoto model for the study of neuronal synchronisation in the brain. *Physica D*, 226(2):181–196, 2007.

[20] A. d'Avila Garcez, L. Lamb, and D. Gabbay. *Neural-Symbolic Cognitive Reasoning.* Springer-Verlag, Berlin, 2009.

[21] P. Dayan and L. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems.* MIT Press, Cambridge, MA, 2001.

[22] G. Deco, V. Jirsa, P. Robinson, M. Breakspear, and K. Friston. The dynamic brain: from spiking neurons to neural masses and cortical fields. *PLoS Computational Biology*, 4, 2008.

[23] R. Drewes, Q. Zou, and P. H. Goodman. Brainlab: a python toolkit to aid in the design, simulation, and analysis of spiking neural networks with the neocortical simulator. *Frontiers in Neuroinformatics*, 3, 2009.

[24] J. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[25] J. A. Feldman and D. H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6(3):205–254, 1992.

[26] J. Fuster. *Memory in the cerebral cortex.* MIT Press, Cambridge, MA, 1995.

[27] J. Fuster. The prefrontal cortex-an update: Time is of the essence. *Neuron*, 30:319–333, 2001.

[28] D. Gabbay. Theoretical foundations for non-monotonic reasoning in expert systems". In K. Apt, editor, *Logic and Models of Concurrent Systems.* Springer-Verlag, Berlin, 1985.

[29] D. Gabbay, C. Hogger, and J. Robinson, editors. *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1-5. Oxford UP, Oxford, 1994-1999.

[30] J. Garson. Connectionism. *Stanford Encyclopedia of Philosophy*, 2010.

[31] G. Gayler. Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience. In P. Slezak, editor, *ICCS/ASCS International Conference on Cognitive Science*, pages 133–138, 2003.

[32] D. F. M. Goodman and R. Brette. The brian simulator. *Frontiers in Neuroscience*, 3:192–197, 2009.

[33] G. Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2:397–425, 1992.

[34] C. M. Gray. The temporal correlation hypothesis of visual feature integration: Still alive and well. *Neuron*, 24(1):31–47, 1999.

[35] F. C. Harris, M. C. Ballew, J. Baurick, J. Frye, L. Hutchinson, J. G. King, P. H. Goodman, and R. Drewes. A novel parallel hardware and software solution for a large-scale biologically realistic cortical simulation. In *Computer Applications in Industry and Engineering*, pages 72–77, 2006.

[36] D. Hebb. *The organization of behavior.* Wiley & Sons, New York, 1949.

[37] P. Hitzler, S. Hölldobler, and A. K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 2, 2004.

[38] S. Hölldobler and Y. Kalinke. Towards a massively parallel computational model for logic programming. In *Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pages 68–77. ECCAI, 1994.

[39] S. Hölldobler, Y. Kalinke, and H. Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–58, 1999.

[40] J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the USA*, 79(8):2554–2558, 1982.

[41] E. M. Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting.* MIT Press, Cambridge, MA, 2007.

[42] R. Jackendoff. *Foundations of Language: Brain, Meaning, Grammar, Evolution.* Oxford University Press, 2002.

[43] D. H. Kiela. Classifying non-monotonic representation in neural networks: In defense of the middle level. *Unpublished*, 2010.

[44] T. Kohonen. *Self-Organization and Associative Memory.* Springer, Berlin, 1984.

[45] P. Konig, A. Engel, and W. Singer. Integrator or coincidence detector? the role of the cortical neuron revisited. *Trends Neuroscience*, 19(4):130–137, 1996.

[46] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Artificial Intelligence*, 44:167–207, 1990.

[47] T. Lange and M. Dyer. High-level inferencing in a connectionist network. *Connection Science*, 1:181217, 1989.

[48] H. Leitgeb. *Inference on the Low Level: An investigation into deduction, nonmonotonic reasoning and the philosophy of cognition.* Springer, Berlin, 2004.

[49] H. Leitgeb. Interpreted Dynamical Systems and Qualitative Laws: From Neural Networks to Evolutionary Systems. *Proceedings of an Unknown Conference?*, ?, 2004.

[50] H. Leitgeb. Interpreted Dynamical Systems and Qualitative Laws: From Neural Networks to Evolutionary Systems. *Synthese*, 146:189–202, 2005.

[51] D. B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project.* Addison-Wesley, 1990.

[52] W. Maass. Networks of spiking neurons: The third generation of neural networks models. *Neural Networks*, 10:1659–1671, 1997.

[53] W. Maass. Paradigms for computing with spiking neurons. In J. C. J.L. van Hemmen and E. Domany, editors, *Models of Neural Networks.* Springer, Berlin, 2002.

[54] D. Makinson. General Patterns in Nonmonotonic Reasoning. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming 3*, pages 35–110. Clarendon Press, Oxford, 1994.

[55] G. Marcus. *The algebraic mind.* MIT Press, Cambridge, MA, 2001.

[56] J. McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[57] J. McCarthy. Epistemological challanges for connectionism. *Behavioural and Brain Sciences*, 11, 1988.

[58] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115 – 133, 1943.

[59] C. Mehring, U. Hehl, M. Kubo, M. Diesmann, and A. Aertsen. Activity dynamics and propagation of synchronous spiking in locally connected random networks. *Biological Cybernetics*, 2003.

[60] P. Miller, C. Brody, R. Romo, and X. Wang. A recurrent network model of somatosensory parametric working memory in the prefrontal cortex. *Cerebral Cortex*, 13(11):1208–18, 2003.

[61] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

[62] D. Neville. Non-monotonic reasoning in biologically oriented neural networks. Master's thesis, Universiteit van Amsterdam, 2007.

[63] N. S. Park and D. Robertson. A localist network architecture for logical inference based on temporal synchrony approach to dynamic variable binding. In *IJCAI-95 Workshop on Connectionist-Symbolic Integration: From Unified to Hybrid Approaches*, pages 63–68, 1995.

[64] N. S. Park, D. Robertson, and K. Stenning. An extension of the temporal synchrony approach to dynamic variable binding in a connectionist inference system. *Knowledge-Based Systems*, 8, 1995.

[65] D. H. Perkel, G. L. Gerstein, and G. P. Moore. Neuronal spike trains and stochastic point processes. i. the single spike train. *Biophysics Journal*, 7:391–418, 1967.

[66] D. H. Perkel, G. L. Gerstein, and G. P. Moore. Neuronal spike trains and stochastic point processes. ii. simultaneous spike trains. *Biophysics Journal*, 7:419–440, 1967.

[67] G. Pinkas. Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge. *Artificial Intelligence*, 77:203–247, 1995.

[68] S. Pinker and A. Prince. On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. *Cognition*, 28:73–193, 1988.

[69] T. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6:623–641, 1995.

[70] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[71] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1961.

[72] D. Rumelhart and J. McClelland. *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, MA, 1986.

[73] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[74] E. Salinas and T. Sejnowski. Impact of correlated synaptic input on output firing rate and variability in simple neuronal models. *Journal of Neuroscience*, 20:6193–6209, 2000.

[75] E. Salinas and T. Sejnowski. Correlated neuronal activity and the flow of neural information. *Nat. Rev. Neurosci.*, 2(8):539–550, 2001.

[76] U. Sandler and L. Tsitolovsky. *Neural Cell Behavior and Fuzzy Logic*. Springer, New York, 2008.

[77] M. N. Shadlen and J. A. Movshon. Synchrony unbound: a critical evaluation of the temporal binding hypothesis. *Neuron*, 24, 2005.

[78] L. Shastri. Advanced in shruti — a neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Applied Intelligence*, 11:78–108, 1999.

[79] L. Shastri and V. Ajjanagadde. From associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioural and Brain Sciences*, 16:417494, 1993.

[80] Y. Shoham. *Reasoning about Change*. MIT Press, Cambridge, MA, 1988.

[81] O. Shriki, D. Hansel, and H. Sompolinsky. Rate models for conductance-based cortical neuronal networks. *Neural Computation*, pages 1809–1841, 2003.

[82] W. Singer. Neuronal synchrony: A versatile code for the definition of relations? *Neuron*, 24(1):49–65, 1999.

[83] P. Smolensky. On variable binding and the representation of symbolic structures in connectionist systems. *Technical Report CU-CS-355-87*, 1987.

[84] P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. In G. Hinton, editor, *Connectionist Symbol Processing*, pages 159–216. MIT Press, Cambridge, MA, 1991.

[85] S. Song, K. D. Miller, and L. F. Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, 3, 2000.

[86] J. Sougne. Binding and multiple instantiation in a distrubted network of spiking nodes. *Connection Science*, 13(1):99–126, 2001.

[87] K. Stenning and M. van Lambalgen. *Human Reasoning and Cognitive Science*. MIT Press, Cambridge, MA, 2003.

[88] T. Trappenberg. *Fundamentals of Computational Neuroscience*. Oxford University Press, Oxford, 2002.

[89] M. Ursino, E. Magosso, and C. Cuppini. Object segmentation and recovery via neural oscillators implementing the similarity and prior knowledge gestalt rules. *BioSystems*, 85:201–218, 2006.

[90] W. Usrey, J. Alonso, and R. Reid. Synaptic interactions between thalamic inputs to simple cells in cat visual cortex. *Journal of Neuroscience*, 20(14):5461–5467, 2000.

[91] E. Vassilieva, G. Pinto, J. A. de Barros, and P. Suppes. Learning pattern recognition through quasi-synchronization of phase oscillators. *IEEE Transactions on Neural Networks*, 22(1):84–95, 2011.

[92] F. V. D. Velde. Neural blackboard architectures of combinatorial structures in cognition. In *Behavioral and Brain Sciences*, pages 1–72, 2006.

[93] T. P. Vogels and L. Abbott. Signal propagation and logic gating in networks of integrate-and-fire neurons. *Journal of Neuroscience*, 25:10786–10795, 2005.

[94] T. P. Vogels, K. Rajan, and L. F. Abbott. Neural network dynamics. *Annual Review of Neuroscience*, 28:357–376, 2005.

[95] C. von der Malsburg. The correlation theory of brain function. *MPI Biophysical Chemistry, Internal Report 812*, 1981.

[96] C. von der Malsburg. Synaptic plasticity as basis of brain organization. In J. Changeux and M. Konishi, editors, *The Neural and Molecular Bases of Learning*. Wiley, 1987.

[97] V. Weber. Connectionist unification with a distributed representation. In *Proceedings of the International Joint Conference on Neural Networks*, pages 555–560, 1992.

[98] V. Weber. Unification in prolog by connectionist models. In *Proceedings of the Fourth Australian Conference on Neural Networks*, 1993.

[99] C. Wendelken and L. Shastri. Acquisition of concepts and causal rules in shruti. In *Proceedings of Cognitive Science*, 2003.

[100] C. Wendelken and L. Shastri. Multiple instantiation and rule mediation in shruti. *Connection Science*, 16:211–217, 2004.

[101] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[102] E. C. Wilson, P. H. Goodman, and F. C. H. Jr. Implementation of a biologically realistic parallel neocortical-neural network simulator. In *Parallel Processing for Scientific Computing*, 2001.

[103] M. Zirpe. Rain and ncs 5 benchmarks. Master's thesis, University of Nevada, Reno, 2007.