

Towards a Computational Theory of the Brain: The Simplest Neural Models, and a Hypothesis  
for Language

Daniel Mitropolsky

Submitted in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy  
under the Executive Committee  
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2024

© 2024

Daniel Mitropolsky

All Rights Reserved

## **Abstract**

Towards a Computational Theory of the Brain, and Language in the Brain

Daniel Mitropolsky

Obtaining a computational understanding of the brain is one of the most important problems in basic science. However, the brain is an incredibly complex organ, and neurobiological research has uncovered enormous amounts of detail at almost every level of analysis (the synapse, the neuron, other brain cells, brain circuits, areas, and so on); it is unclear which of these details are conceptually significant to the basic way in which the brain computes. An essential approach to the eventual resolution of this problem is the definition and study of theoretical computational models, based on varying abstractions and inclusions of such details. This thesis defines and studies a family of models, called NEMO, based on a particular set of well-established facts or well-founded assumptions in neuroscience: atomic neural firing, random connectivity, inhibition as a local dynamic firing threshold, and fully local plasticity. This thesis asks: what sort of algorithms are possible in these computational models? To the extent possible, what seem to be the simplest assumptions where interesting computation becomes possible? Additionally, can we find algorithms for cognitive phenomena that, in addition to serving as a “proof of capacity” of the computational model, otherwise reflect what is known about these processes in the brain? The major contributions of this thesis include:

1. The formal definition of the basic-NEMO and NEMO models, with an explication of their neurobiological underpinnings (that is, realism as abstractions of the brain).
2. Algorithms for the creation of neural *assemblies*, or highly dense interconnected subsets of

neurons, and various operations manipulating such assemblies, including reciprocal projection, merge, association, disassociation, and pattern completion, all in the basic-NEMO model. Using these operations, we show the Turing-completeness of the NEMO model (with some specific additional assumptions).

3. An algorithm for parsing a small but non-trivial subset of English and Russian (and more generally any regular language) in the NEMO model, with meta-features of the algorithm broadly in line with what is known about language in the brain.

4. An algorithm for parsing a much larger subset of English (and other languages), in particular handling dependent (embedded) clauses, in the NEMO model with some additional memory assumptions. We prove that an abstraction of this algorithm yields a new characterization of the context-free languages.

5. Algorithms for the blocks-world planning task, which involves outputting a sequence of steps to rearrange a stack of cubes in one order into another target order, in the NEMO model. A side consequence of this work is an algorithm for a chaining operation in basic-NEMO.

6. Algorithms for several of the most basic and initial steps in language acquisition in the baby brain. This includes an algorithm for the learning of the simplest, concrete nouns and action verbs (words like "cat" and "jump") from whole sentences in basic-NEMO with a novel representation of word and contextual inputs. Extending the same model, we present an algorithm for an elementary component of syntax, namely learning the word order of 2-constituent intransitive and 3-constituent transitive sentences. These algorithms are very broadly in line with what is known about language in the brain.

## Table of Contents

Acknowledgments . . . . .	vi
Chapter 1: Introduction and Background . . . . .	1
1.1 Why construct abstract theories? . . . . .	2
1.2 An Approach Inspired by Theoretical Computer Science . . . . .	3
1.3 Basic neurobiological underpinnings for NEMO . . . . .	4
1.4 Contribution of this Thesis . . . . .	7
1.5 The Theory of Assemblies, and Comparisons to Other Models . . . . .	9
1.5.1 Hebb and early theory of assemblies . . . . .	9
1.5.2 Marr’s theories of cortex and hippocampus . . . . .	10
1.5.3 Hopfield networks . . . . .	15
1.5.4 “Arbitrary” computation with Hopfield networks . . . . .	17
1.5.5 ANNs . . . . .	21
1.5.6 Valiant’s model . . . . .	22
1.6 Experimental study of assemblies. . . . .	22
1.7 Techniques in neurolinguistics . . . . .	24
1.7.1 The main methodologies . . . . .	24
1.7.2 Songbirds . . . . .	26

Chapter 2: The NEMO model . . . . .	28
2.1 The basic-NEMO model . . . . .	28
2.1.1 Parameter settings . . . . .	31
2.2 NEMO . . . . .	32
2.3 Other extensions of NEMO . . . . .	33
Chapter 3: Computing with NEMO: Creation of Assemblies and Basic Operations . . . . .	35
3.1 The computational power of basic-NEMO . . . . .	35
3.1.1 Assembly creation . . . . .	37
3.1.2 Association . . . . .	39
3.1.3 Pattern completion . . . . .	40
3.1.4 Reciprocal project . . . . .	40
3.1.5 Merge . . . . .	41
Chapter 4: Parsing in NEMO . . . . .	43
4.1 Overview . . . . .	43
4.1.1 Goals . . . . .	46
4.1.2 Background . . . . .	47
4.2 The NEMO model: Review and Additional Assumptions . . . . .	49
4.3 The Parser . . . . .	54
4.3.1 Parser Architecture . . . . .	54
4.3.2 Operation . . . . .	55
4.3.3 Readout . . . . .	56
4.4 Experiments . . . . .	59

4.5	Details of the experiment . . . . .	60
4.6	Extensions and Discussion . . . . .	61
4.6.1	Conclusion . . . . .	67
Chapter 5: Parsing embedded clauses . . . . .		68
5.1	Introduction . . . . .	68
5.1.1	The original Parser . . . . .	70
5.1.2	Neuroscience . . . . .	71
5.2	Constituency . . . . .	71
5.3	Embedded Sentences . . . . .	72
5.4	A Little Formal Language Theory . . . . .	77
5.5	Discussion . . . . .	80
5.6	Proof of Theorem 1 . . . . .	81
Chapter 6: Planning in the Blocks World . . . . .		86
6.1	Introduction . . . . .	86
6.2	Background and related work . . . . .	87
6.3	The Blocks World NEMO Program . . . . .	88
6.3.1	The Parser . . . . .	89
6.3.2	Removing the Top Block . . . . .	90
6.3.3	Putting a Block on Top of the Stack . . . . .	92
6.3.4	Computing the Intersection of Two Stacks . . . . .	93
6.3.5	Multiple Stacks . . . . .	93
6.4	Experiments . . . . .	94

6.5	Chaining, and its limits in NEMO. . . . .	94
6.6	Conclusions and Future Directions . . . . .	97
Chapter 7: Planning in the Blocks World . . . . .		98
7.1	Introduction . . . . .	98
7.2	On Language in the Brain . . . . .	100
7.3	Psycholinguistic theories . . . . .	101
7.4	The Language Organ . . . . .	102
7.5	The learning experiment . . . . .	104
7.5.1	Individual word tutoring . . . . .	106
7.6	Future Work . . . . .	108
7.7	Conclusion . . . . .	109
References . . . . .		110
Appendix A: Experimental Equipment . . . . .		120
Appendix B: Data Processing . . . . .		121

## List of Figures

2.1	A NEMO instance . . . . .	29
3.1	Assembly creation, and operations of basic-NEMO . . . . .	36
4.1	Example of a lexical assembly's action . . . . .	55
4.2	Execution of the Parser on an example sentence . . . . .	58
5.1	Example of the extended Parser on an embedded phrase . . . . .	76
6.1	An example of block worlds configurations, and the planning algorithm . . . . .	91
6.2	Experiments on the chaining primitive . . . . .	95
7.1	The architecture of the NEMO language organ . . . . .	104
7.2	Experimental results for learning concrete nouns and verbs . . . . .	107

## **Acknowledgements**

First and foremost, my deepest gratitude lies with my advisors, Christos Papadimitriou, Tal Malkin, and earlier Michael J. Collins. I thank them for their mentorship at all stages of my PhD, their dedication to my growth and success as a researcher and academic, and for the friendship I have forged with each of them. I feel extremely, extremely grateful and indebted to them. I want to sincerely thank my closest collaborators, including Alon Rosen, Prahladh Harsha, Max Dabagia, and Santosh Vempala; my PhD would not have been successful without them. I also wish to thank all of my co-authors throughout the years, including Adiba Ejaz, Mirah Shi, Mihalis Yannakakis, Francesco D'Amore, Pierluigi Crescenzi, and Emanuele Natale.

My gratitude also lies with my friends in the PhD cohort, including Tim Randolph, Clayton Sanford, Shivam Nadimpalli, Miranda Christ, and Marshal Ball, among many others.

## Chapter 1: Introduction and Background

Understanding the computational basis of the brain is one of the most important problems in basic science. How do neurons, in their collective action, beget cognition, as well as intelligence and reasoning? As Nobel laureate neuroscientist Richard Axel eloquently put it, “we do not have a logic for the transformation of neural activity into thought and action. I view discerning [this] logic as the most important future direction of neuroscience” [1].

The brain is an incredibly complex organ, and neurobiological research has until now uncovered enormous amounts of detail at almost every level of analysis. One large direction within neuroscience focuses on the physiology of the neuron, the individual cell making up the brain; this includes a very deep theory of electrochemical neuronal membrane, synapses, neurotransmitters, and the precise nature of the action potential, or neuronal firing. Indeed, much of a classic neuroscience textbook is dedicated to these phenomena [2]. From our perspective, this is on the “fine-grained” end of the spectrum. Another body of work is concerned with studying small neuronal circuits, where we can understand how neurons are wired together, and hence even their functional roles. For instance, in the case of the *C. elegans* nematode, where individuals of the species have an identical set of neurons (exactly 302) and neuronal connections, neuroscientists have mapped out the entire neuronal circuit of the organism, constructing a connectome or neuronal wiring diagram of the organism [3]. *Connectomics* is the field concerned with constructing complete maps or diagrams of neurons and their connections for organisms beyond *C. elegans*.

On the other end of the spectrum, that is on the “coarse-grained” end, we have rich theories of higher-level cognitive phenomena and entire, often independent scientific disciplines devoted to their study, such as psycholinguistics, psychology, and cognitive science. These disciplines include the study of macro-level brain physiology using methods such as fMRI and EEG. These are very useful for telling us which broad brain regions are implicated in various cognitive processes, but

are at relatively coarse level of resolution. What remains poorly understood is the zone between these two ends of the spectrum; how do these higher level cognitive phenomena “compile” down to firing neurons? Which details of neuroscience, and from which levels of analysis of the brain, are significant to a naturalistic and conceptual understanding of the way in which the brain computes? A satisfying explanation of the computer would be a model of logical gates and how they are used compositionally to construct registers and other arithmetic operators, but not a detailed description of, say, the electromechanics of electrons of wires, or the transistors underlying the gates, nor would higher-level facts about how compilers or operating systems are organized be helpful for this kind of conceptual description. Similar analogies can be made for genetics, physics, and other fields.

## **1.1 Why construct abstract theories?**

One of the main approaches in neuroscience has been to try and mine the ever-increasing supply of neural data and to attempt to induce the basic logic of the brain from the data. That is, statistical techniques and data-analysis methods are developed that, applied to neurobiological data, hope to elucidate some fundamental principal of neural computation. Yet, even ignoring the issue that much of neural data may not be at the right level of resolution (e.g. brain areas combining over many neurons as opposed to individual neurons, etc), that is, even assuming we could record from all neurons individually and simultaneously, there is strong evidence that it will be very hard to understanding anything meaningful in this way. In a recent paper titled “Could a Neuroscientist Understand a Microprocessor?” [4], the authors recorded the electrical signals from every transistor in a simulated microprocessor as it was used in various downstream tasks (computer programs), in some sense equivalent to the aforementioned scenario in which we were able record from all neurons individually and simultaneously. Although the tasks varied, the transistors of the chip were always part of a larger circuit that computed arithmetic operations that made up the rest of the computation (they implemented logical gates that made up registers, and so on). When the authors applied a barrage of the most popular and modern data analysis tech-

niques used in neuroscience to the microprocessor data, they found that the results revealed nothing about the fundamental, compositional structure of the device (logical gates), let alone registers and arithmetic operations. On the other hand, one could determine some patterns and regularities in the data (often of the kind that certain transistors were active at regular intervals, i.e. had some sort of periodicity) but they convincingly argue that this does not help us understand the device. Analogizing their findings to the case of the brain, one is left with the impression that perhaps the greatest problem in neuroscience is that we do know not *what* to even look for in neural data.

## 1.2 An Approach Inspired by Theoretical Computer Science

In the case of a system as complex as the brain, perhaps an approach is needed in the *opposite* direction of the classic data-to-theory paradigm. That is, what if we started by setting aside the overabundance of neural data, and directly defined and studied mathematical *models of computation* based on only the most ubiquitous and seemingly fundamental principles of neuron and brain biology. Of course, selecting “ubiquitous and fundamental” principles is no easy task (and is hard to disentangle from *the* task we are trying to solve). However, until we have reasons to eliminate theories, the author sees it as worthwhile to define *any* model that is broadly plausible neurobiologically. As we study these models and understand their capabilities better, as well as extract predictions and compare them with the evolving experimental evidence, we will, eventually, be able to “whittle down” the space of candidate descriptions of the brain.

This thesis is a vertical column in this theoretically-inspired approach: we define and initiate the theoretical study of one specific “family” of models, known as the NEMO models, based on a particular set of neurologically-plausible assumptions. At a very high level, these assumptions can be summarized as: random connectivity, a constant-number-of-winners-takes-all notion of local inhibition, and Hebbian plasticity. Initially, these particular assumptions gained our attention because of work showing their usefulness in limited pathways in the brain, namely in the olfactory system of the fruit fly [5] and of the mouse [6]. However, it was the discovery that densely interconnected *neuronal assemblies* or *ensembles* emerge organically from these principles in an

extremely natural way (Chapter 3) that inspired a deeper and sustained interest in understanding the potential of this computational system.

That is, we will define the NEMO family of models and ask: what kinds of algorithms can be implemented in these models? What is their *computational power*, especially relative to one another? If we found that some cognitive task is *not* achievable in a model we defined (or at least not in any realistic way), we would learn that our particular set of assumptions is not a good candidate for a model of the brain. The ultimate goal is to find a *minimal set of assumptions* that are sufficient for algorithms for the hardest problems that brains can do. To complete the connection with theoretical computer science, studying algorithms in these computational models of the brain corresponds to studying *upper bounds* (or algorithms) in traditional models of computation in computer science. While it is very difficult, another desideratum would be, especially given an algorithm for a certain cognitive task at a particular level of the hierarchy, a demonstration of the *impossibility* of an algorithm if the model is weakened in any way (i.e. some neurobiological detail is removed). The study of such “lower bounds” is notoriously difficult, and very few such impossibility results are known even in the simplest models of computation. However, it is an important idea, if just an aspiration, to keep in mind in our approach to studying the brain. It will be at least philosophically suggestive if a natural and simple algorithm for a particular phenomenon seems to “fall out” of the definition of a particular brain model, but removing any assumption results not only in the algorithm losing its implementability, but it becomes unclear how to approach the problem at all. This is certainly the impression I have with several of our algorithms, and we will point out such musings when they so arise.

### **1.3 Basic neurobiological underpinnings for NEMO**

This section is not meant to serve as a survey of modern neuroscience; doing so would be greatly beyond the scope and aim of my work. Instead, here I will aim to briefly summarize several neurobiological principles, some very concrete and well-established and others abstractions of multiple underlying processes. Many of these may be familiar to the reader. These are the most

important ideas for defining and understanding the NEMO family of models.

**Atomic firing of neurons** . A neuron fires (undergoes an *action potential*) at a specific time or it does not; there is no “intermediate” or partial firing. Neurons fire as a result of sufficient electrochemical potential across their membrane as a result of neurotransmitters released from other neurons whose axons have outgoing synapses to the neuron in question.

**Excitatory and inhibitory neurons.** Neurons are either excitatory (all of their synapses *increase* membrane depolarization of the neurons they are connected to, i.e. they have all positive outgoing synaptic weights) or inhibitory (they have all negative outgoing synaptic weights. Most of the brain, and hence we believe most of neural computation and representation, takes place on the level of excitatory neurons, with inhibitory neurons playing other roles.

**Local (Hebbian) plasticity** All synapses evolve as a function of their local state *only*; that is, if a synapse from neuron  $\alpha$  to neuron  $\beta$  has state  $w$ , its state at a later time must (at least reasonably be approximated) by a function  $f(w, t_\alpha, t_\beta)$ , where  $t_\alpha$  and  $t_\beta$  are the most recent firing times of  $\alpha$  and  $\beta$ . Of course, diffuse neurotransmitters in a brain region, or previous firing times of  $\alpha$  and  $\beta$  may play a role, but it is reasonable to assume, at least as a starting point, that a function depending only on these 3 inputs would be sufficient to capture the kind of function we are interested in.

**Local and interarea inhibition** Inhibitory neurons are known to broadly come in two flavors: local and long-range (relay) inhibitory neurons. In many regions of the brain, especially in cortex (the largest and most evolutionarily distinctly developed part of the human brain, known to be involved in almost all cognitive tasks), we think that local inhibitory neurons somewhat equally and diffusely inhibit all of the neurons in a small localized region. Since they will fire more when there is more firing into the area, these neurons might play a role of actively maintaining a dynamic input threshold for firing of the area’s excitatory neurons. On the other hand, interarea (relay) inhibitory neurons have long axons and go from one area inhibiting the neurons in *another*.

The next two principles are central to the family of models studied in this thesis, but they are more tentative than the previous ones. A parallel field of study could be developed studying families of models that make assumptions *different* to the ones below. On the other hand, these assumptions give us a concrete starting point, and if it turns out, their simplest interpretations lead to the definition of models that have surprising computation power.

**Random connectivity** It is not unreasonable to model the initial or default state of an area of neurons as being randomly connected. Indeed, some sort of random connectivity is to be expected given that the brain *develops*, and synapses are sampled locally between pairs of neurons as the brain grows. The simplest random graph model is the Erdos-Renyi random graph, where each vertex is connected to every other one with a fixed probability  $p$ . However, experimental results do show that that synaptic connectivity in the animal brain deviate from the uniformity of the Erdos-Renyi graph.

**Assemblies of neurons** Currently, it is unknown how abstract concepts (words, images, memories, etc) are represented in the brain: could they be single neurons, sets of neurons, specific firing patterns or rates (of individual neurons or sets), or something else? In fact, representations could be (and likely are) different depending on the domain: single neuron representations have evidence in specific perceptual pathways such as in the visual system, where it has been found that individual neurons respond to specific locations and borders in space [7]. However, sometimes called the “grandmother neuron doctrine”, this idea has come under serious doubt especially as it applies to representations in higher-level, multi-modal or integrating cognitive processes (such as reasoning, language, etc). The main alternate hypothesis, which has been steadily gaining attention in neuroscience, is that of neural *assemblies*, or ensembles, as being the fundamental “unit” of cognition or cognitive representation. Assemblies, originally hypothesized by Hebb in the late 1940s and 50s [8], are *densely intraconnected* sets of neurons; that is, the chance that two neurons have a synaptic connection is significantly larger between two neurons of an assembly than between other neurons in the same region. There is increasing experimental evidence for assemblies in mammalian brains

[9, 10, 11]; for a survey, see [12].

## 1.4 Contribution of this Thesis

In **Chapter 2**, the thesis begins with the definition of a small family (or rather, hierarchy) of models defined by the basic neural assumptions summarized in the previous section: firing neurons, two different kinds of inhibition (local and long-range), random connectivity, and *local* synaptic plasticity. The main models in this hierarchy are one that formalizes all these principles but leaves out long-range inhibition, called **basic NEMO**. With long-range inhibition, the model will be called **NEMO**. NEMO is short for “neural model”. This initiates the theoretical study of the NEMO models, both *mathematically* (*proving* that certain properties hold, or certain algorithms function with high probability), and, especially in cases where algorithms are beyond mathematical analysis, experimentally through *simulation*. A major contribution of this thesis is the development of open-source **NEMO libraries** in Python and in C++, partly in collaboration with Google Research [13].

The basic-NEMO model defines a dynamical system, which, based off its starting conditions (which we will use to represent some inputs to the brain), evolves over time in a deterministic way. What we find is that surprisingly, interesting phenomena occur even in this most basic model. Specifically, highly interconnected, co-firing sets of neurons organically emerge in this model in the one of the simplest experiments one could formulate within the model: one where a fixed stimulus fires into a brain area. Could these be precisely the kinds of neural assemblies of the assembly-theory of neural computation? We find that the assemblies obtained via this extremely simple basic-NEMO algorithm share hypothesized and experimentally observed properties of assemblies in experimental neuroscience, namely high intra-assembly density, the ability to co-fire, and that they can be used as a basis of various operations of a computational flavor. We demonstrate, both theoretically and through simulation, that it is possible to carry out operations on these concepts, such as copying, merging, and pattern completing them. These results are covered in **Chapter 3**.

**The full NEMO model and language processing.** The phenomena that we prove or observe in the extremely simple basic-NEMO model above are still quite far from carrying out the kind of computation that underlies reasoning and language (though we can see how they may be sufficient to implement small *parts* of these broader phenomena, e.g. the merge and pattern-completion operations of basic-NEMO may underlie individual word retrieval, and be a building block in syntactic processing). The full NEMO model adds one additional tenet, namely that of *inter-area* inhibition (i.e. based on long-range, or relay, interneurons), again defined in an abstracted way (we only allow the firing of these neurons to rather bluntly inhibit *entire areas*, and not for instance arbitrarily inhibit specific excitatory neurons). This turns out to give us a very powerful set of assumptions. In **Chapter 4**, we show how to use NEMO to do our first truly complex cognitive task, by constructing an efficient parser of a small but non-trivial subset of English. To our knowledge, this constituted the first parser that is composed entirely in a model of *biologically realistic neurons, synapses and Hebbian plasticity*. However, the parsing algorithm crucially makes use of the LRI ingredient of the full NEMO model, although only in the *input* to the model. Further, the initial parser could parse only relatively simple sentence, and in particular could not handle embedding or dependency, arguably the most interesting part of language.

In **Chapter 5**, we show how to extend the parser in a non-trivial way to handle embedded or dependent structure. While coming up with a NEMO algorithm for parse embedded clauses, we stumbled upon a surprising consequence: by slightly abstracting this algorithm, (which again was “designed” to work in the constraints of the NEMO computational model, i.e. to compile down to biologically realistic hardware), one obtains a novel characterization of the context-free languages (CFL). This is suggestive because CFLs were originally defined and studied by Chomsky and other linguists as a possible model for natural language; it was later found that they are both somewhat *too* powerful for real languages (e.g. most languages do not really allow for embedding past depth-4) and sometimes not powerful enough (Swiss German and Dutch famously have context-sensitive syntactic rules); but yet, they overall remain a descent model for human language, and underlie many of the most important syntactic models in NLP (e.g. PCFGs). In a sense, we

“rediscovered” this formal language class as a result of studying parsing algorithms in a neurally plausible computational model.

Adding to the theory of how algorithms for cognitive phenomena are possible in NEMO, we show in **Chapter 6** how to implement an algorithm for solving block-stacking problems in the blocks world framework. This adds to the list of reasonably efficient NEMO algorithms for cognitive phenomena. A subroutine developed for this algorithm is a *chaining* operation, and we obtain theoretical and empirical results regarding the limits of chaining in NEMO.

**The full NEMO model and language learning.** The parsing algorithms demonstrate that NEMO is a sufficient computational model to capture algorithms for language in the “adult” brain; that is, where learning involved but rather known semantic and syntactic information about a language are used to compute the structure of incoming sentences. Of course, one of the most sophisticated abilities of the brain is learning, and a natural question is whether the axioms of NEMO are sufficient, or if not, what minimal modifications are necessary (indeed, “learning” also encompasses many very different processes; native language acquisition takes place in the child brain, and may use different principles). In **Chapter 7** we present a NEMO model of the baby brain that *learns the meaning of words, and very basic syntax*, from grounded input [14]. In addition, this is a new kind of AI that is fundamentally different from LLMs (and ANNs more broadly): ours, like the brain, learns entirely using local synaptic rules— that is, without back-propagation.

## 1.5 The Theory of Assemblies, and Comparisons to Other Models

### 1.5.1 Hebb and early theory of assemblies

There is a long and relatively old line of work studying assemblies (or ensembles) as conceptual and computational units in the brain from a theoretical perspective, much of which can be seen as leading up to our models and results, or as parallel investigations. Traditionally the first such theoretical work is normally taken to be Hebb’s 1949 book “The Organization of Behavior” [8]. The book covers many topics but of particular interest is Chapter 4, in which Hebb famously pos-

tulates the notion of Hebbian plasticity. He proposes a neurophysiological postulate, that "when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells that A's efficiency, as one of the cells firing B, is increased." In the decades that followed, experimental neuroscience would find evidence for this hypothesis and Hebbian plasticity (and close variants thereof) would become a canonical, basic kind of plasticity known to take place in the brain. Yet, only hypothesizing such a function in 1949, Hebb goes on to specify both how this might be possible (through axonal bud-like growth at junctions between myelinating cells) and how (qualitatively or intuitively) such plasticity would result in three-dimensional "lattices" of neurons with connections all of the neurons in the lattice so as for them to (tend to) fire together.

**Connections to NEMO.** Of course, without Hebb's work this thesis could not exist. In fact, one view of this thesis as attempting to answer: what are the theoretical limits of a point-neuron computational model with *only* Hebbian plasticity? The NEMO model concretizes and implements Hebb's hypothesis that assemblies can be formed via Hebbian plasticity. Other theoretical neuroscientists have taken Hebb's principal as a fundamental axiom in developing their own models, which we will visit in the following subsections, and which have quite a bit in common with NEMO.

### 1.5.2 Marr's theories of cortex and hippocampus

The next big milestone in the theoretical study of cell assemblies, and one of the most important comparisons to the models of this thesis, is Marr's line of work in the late 60s and early 70s. Even though Marr's work itself does not refer to assemblies nor would he necessarily use the term, we will see that it indeed belongs in the tradition leading up to this thesis. The main papers through which Marr developed his theory are "A Theory of Cerebellar Cortex" [15], "A Theory for Cerebral Neocortex" [16] and "Simple Memory: A Theory for Archicortex" [17], which, roughly speaking, build upon each other to construct a theoretical model of cortex and the hippocampus

(or “archicortex”) made up of biologically-realistic neurons and synapses that is able to do one operation very well: take as input various “events”, which are sparse activations of a small number of total neurons in an input space (cortex), “project” them to hippocampus which has many fewer neurons and where assemblies of those neurons represent quickly-created memories of the input events, and “recall” events by activating a small subset of the neurons of an actual event, feeding it through hippocampus, and restoring the whole event reliably. This was of interested because such short- to medium-term memory is an essential function of the brain, and the hippocampus was known to implement it; it memorized events or patterns from the environment very quickly without analyzing them further (or at least not very much), because perhaps long-term memories were formed from projections back from hippocampus with additional analysis. Hence, the hippocampus only had to be able to store a “days-worth” of events, which is estimated by Marr to be around 100,000 events per day.

What follows is a summary of Marr’s model (pooling across his sequence of papers). As mentioned, the basic model will have 2 layers: one representing cortex, where events are inputted, and one representing hippocampus, where memories of the events, i.e. compact representations of the events that are sufficient for recall from a partial event, will be formed. In the basic model, the first cortex layer is also the “output” layer in the sense that the hippocampal layer will project to the output layer in order to restore the full event from a partial event (in a slightly more expanded version there is a 3rd output layer, but that model is essentially or at least conceptually the same as the 2 layer version described here). An immediate issue is that a layer truly representing “cortex” would be too big—cortex has over  $10^8$  pyramidal (i.e. excitatory) neurons, and  $10^8$  connections into hippocampus was understood to be outside the bounds of possibility. This can easily be “solved” or abstracted away via “indicator cells” or pooling cells, each of which pools over a local area of cortex. If an indicator cell pools over about  $1/3 \text{ mm}^2$ , then estimating there to be  $400 \text{ cm}^2$  of cortex that is implicated in hippocampal-memory formation/retrieval, these leaves us with just over  $10^6$  incoming fibers (axons) into hippocampus, which is realistic. Hence we will think of the cortical/input layer as having about  $10^6$  neurons. Of course this means there is some loss of

information within each pooling region, but assuming that input patterns (or at least “distinct” ones) aren’t *too* similar, this is assumed not to be a problem. Then there are  $10^4$  or  $10^5$  cells in the second *hippocampal* layer (the exact number is a matter of debate between the Marr papers and a point of pain; obviously the important point is that this number be less than the size of the input layer for the “compression” operation to be non-trivial). Finally, there are return connections from hippocampus to the input (cortex) layer and *directly* to the pyramidal cells of cortex (i.e. not to the pooling cells); a single cell in hippocampus can make many synapses including with cortical cells in different pool zones.

Marr’s model also makes several other important assumptions about its regime. The first is regarding “activity level”, or the number of active (firing) neurons in the cortex layer per each event; it is assumed to be *low*, that is that events are *sparse* vectors in the cortical layer, and hence (if events are random) there will be a big difference between them (note that this also takes care of the indicator cell issue from before). The “desideratum” of the system is called the “retrieval cue”, which is the minimal acceptable *cue size* or simply fraction of a previously stored event that needs to fire to retrieve the full event. Marr sets this (and achieves this, for certain parameter ranges) to  $1/10$ .

The model makes use of two kinds of synapses, but both are of the same kind; they are “binary” synapses, meaning they are “on” or “off” and not real valued (though making them real valued could perhaps increase the capacity of Marr’s models); chiefly this is done for simplifying the analysis. The simplest of these is the Hebb synapse, which is in the “on” mode when the pre- and post-synaptic cells are fire at the same time. The other is the Brindley synapse, which, succinctly, has two components: a fixed component that never changes, and a modifiable component which is subject to the same “on”/“off” rule as Hebb synapses and changes over time. The cortical layer has  $N$  cells, the hippocampal layer has  $M$  cells (which are also frequently called “codon” or “codon formation” cells, where codon comes from these neurons constituting a neural representation of a compressed “code” of some event). Synapses from cortex to hippocampus are Brindley synapses, and synapses from hippocampus back to cortex are Hebb synapses. This essentially means that

no fancy or interesting computation occurs going back from hippocampal cells to cortex; instead it must occur *in* hippocampus. In our case, the interesting computation is taking a subevent and getting a whole event to fire. Logically, this means that when a subevent  $X$  of event  $E$  fires, it must end up (through some mechanism) activating the *same* hippocampal cells as  $E$  itself. This will be the goal of Marr's models.

The basic idea Marr had is this: for each event, different hippocampal cells will receive different input amounts; if we apply an  $L$ -cap, the means picking the top  $L$  neurons by input in the hippocampal layer, this can serve as the "simple" or compressed representation, i.e. the short-term memory, of the event. Marr shows how the  $L$ -cap operation can easily be implemented with inhibitory interneurons. Two kinds of inhibitory interneurons are needed to achieve this; one, called  $S$  cells, samples the "input lines" (i.e. the fibers into hippocampus) and inhibits the hippocampal cells, and the other samples the fibers from hippocampus *back* into cortex, which adjusts for the fact that weights from the cortical layer are increasing as a whole as more patterns are learned (via the modifiable part of the Brindley synapse). The analysis is not too hard to show that this mechanism can implement a constant threshold of  $L$  firing neurons in the hippocampal layer, which implements the desired representation method.

Now for *recall*, the basic idea should be that on some subevent  $X$  of  $E$  (i.e.  $E$  is the whole set of neurons that fires in the cortical layer for that event, and  $X$  is a subset of at least  $1/10$  of them), the cortical layer projects to hippocampus, and we then somehow decrease the firing threshold until  $L$  codon neurons fires; hopefully this will then (via the connections back to cortex) get back  $E$ . Of course, the issue is that you will get many "false" neurons this way. A better strategy is Marr's idea that hippocampal cells should take into account what fraction of their synapses have been modified during learning, i.e. meaning that they are part of a previously stored pattern (then we know that unmodified synapses, even if they become active during recall, are just noise). The idea is that in recall mode, a hippocampal cell only fires if a sufficient fraction of its active synapses are modified ones; the firing recall (again only in the recall regime) is an equation where the total input into the cell must be greater than a target which depends on the fraction of modified synapses. A

few very simple such equations will “work”, and Marr also discusses how such a threshold rule might be achieved using interneurons. However, an important side note is that this means the model has 2 firing thresholds for hippocampal cells: a normal one (during learning) controlled by the S and G cells which makes it such that  $L$  cells are always firing in that layer, and one for the recall stage, when there are many fewer inputs, where another set of interneurons (called D interneurons) make firing be based of the fraction of modified synapses. Marr showed that both are necessary for optimal performance of memory formation / recall in his overall framework, but exactly how having “both” is possible is left open.

That concludes a sketch of Marr’s model (or rather of the common core to the many variants). What Marr is able to show analytically is that (in slightly varying parameter regimes) is that a surprising number of events can be stored and recalled in this way, with the 1/10 retrieval cue setting; for example, with  $10^5$  neurons in hippocampus, we can store up to  $10^5$  representations reliably. Marr’s model is still the standard model for hippocampus.

**Connections to NEMO.** There is a lot in common between Marr’s models and NEMO. Both make use of a  $k$ -cap (or in Marr’s case  $T$ -cap) operation, implemented identically: via inhibitory, local interneurons. Marr’s models can essentially be thought of as a kind of biologically-motivated feed-forward neural models (barring the connections from the hippocampal layer back to the input cortical layer). This is *one* of the kinds of fibers in NEMO: those *between* NEMO brain areas. What Marr’s models and results show is that a relatively simple and realistic feed-forward set-up can be used to “memorize” a surprisingly large number of patterns (events) that can then be reliably recalled, essentially via the dynamics of projection from a higher dimensional layer to a lower dimensional layer, and Hebbian dynamics. An important starting point of this thesis and of NEMO theory is that NEMO reproduces this ability; firing between NEMO areas is a kind of random “kernel” and indeed some of our most initial results show that we obtain different  $k$ -caps from firing sufficiently different stimuli from an incoming area. These caps can be taken to represent the stimuli, and by rewarding synapses from them *back* to the stimulus area, we can

achieve similar results to Marr's. This is the case where we somehow "freeze" the initial  $k$ -caps obtained from firing in stimuli.

However, even those I call these sets of neurons "assemblies" in the previous section, they are not truly assemblies in the sense that they have no internal connectivity or stability; Marr's theory does not consider this. In NEMO, there are important *recurrent* connections in each area. We could repeat the same experiment but allow recurrent firing. However, then it will often be the case that initially overlapping but largely separate  $k$ -caps will start to overlap more and collapse into a single set of neurons. However, if inputs are sufficiently not-overlapping, and with sufficiently strong Hebbian plasticity, we will still obtain (largely) separate sets of  $k$  neurons in the "hippocampal" area, and now, these are also bona fide assemblies that have strong internal connectivity, internal pattern-completion properties, and so on (these are among the results/operations of Chapter 3). Now, NEMO can be used to do much more than just memory creation and recall; it is a much more general computational system than Marr's models, but the essential function of Marr's models is so important that they also constitute one of the main targets of study in NEMO. Another essential difference is that Marr's model (as many other theoretical models) has two "regimes": for learning, and evaluation (recall). In fact, as we saw in Marr's case, the different inhibitory circuits implementing different thresholding mechanics in the two regimes are not entirely compatible. One essential feature, or rather principle underlying NEMO is that there can be no essential difference between learning and evaluation/recall; the brain is the same in any regime, and differences in dynamics during different functions must arise from the same underlying principles.

### 1.5.3 Hopfield networks

The next milestone in the theory of assembly-like computational theories is the development of Hopfield networks and related models. In a single sentence, Hopfield networks are a *recurrent* neural model that can memorize patterns and then recall or restore them from perturbations of those patterns. Even though they are named after John Hopfield, the ideas behind Hopfield networks have their origin in thermodynamics and physics more generally from the beginning of the 20th century.

Models that are essentially the same as Hopfield networks were described by Japanese researchers Nakano and Amari by the early 70s, but were truly popularized, in particular as neurobiological models, by Hopfield in the 80s starting with the seminal paper "Neural networks and physical systems with emergent collective computational abilities".

What follows is a summary of Hopfield networks and their properties, including all of the original 1982 [18] and some notions from the general literature. As a model they are very different from Marr's models of the previous section, which could be seen as a kind of feed-forward network model, whereas Hopfield networks are inherently *recurrent* and their emergent properties are properties of recurrent networks. Later in comparison to this thesis, we will see that while Marr's model is similar to our models in the interarea (feed-forward) component, our models make heavy use of recurrence (every brain area is recurrent, and recurrence is essential to how computing with NEMO works); Hopfield networks have more in common with NEMO in this aspect.

A basic Hopfield network has  $n$  neurons, each of which has state 0 or 1 (representing resting and firing). A neuron  $i$  fires or not depending on whether its total input exceeds its (pre-set) firing threshold  $U_i$ ; the neurons of Hopfield networks are "binary threshold neurons". The connections between neurons are modeled as single weights  $w_{ij}$ , which are learned in a *training* stage (in Hopfield networks as in Marr's models the learning/recall stages are separated and slightly different dynamics apply in each). Patterns or stimuli are presented as activations of the  $n$  neurons, i.e. as binary vectors in  $E \in \{0, 1\}^n$ . To model Hebbian plasticity, synapses are increased only if the pre- and post-synaptic neurons both fire. That is for storing a single pattern (vector) in the Hopfield network, we set  $w_{i,j}$  to 1 iff  $V_i = V_j = 1$  (and otherwise it is set to 0 or to  $-1$  depending on the precise formalism used; we will use the  $-1$  version). For memorizing multiple patterns, the Hebbian learning is additive but with renormalization. That is, to memorize  $m$  patterns  $E_1, \dots, E_m$ , we set the synapse  $w_{ij} = 1/N \sum_{k=1}^m V_i^k \cdot V_j^k$ . Importantly, the  $w_{ij}$  change only during training, and are fixed during the recall phase.

In the recall (or evaluation) phase, the Hopfield network is initialized in some state (where the state is a vector of neuron activations, i.e. a vector  $V^0 \in \{0, 1\}^n$ ) and then evolves as expected;

given a state  $V^t$  the next state  $V^{t+1}$  is computed by setting  $V_i^{t+1}$  to 1 iff  $\sum_{j \neq i} w_{ji} V_j^t > U_i$  (this implicitly enforces that  $w_{ii} = 0$ ; usually Hopfield networks are defined with a reciprocal property such that  $w_{ij} = w_{ji}$ , whether this is biologically realistic or not, for the proof of convergence to work). Hopfield and later other researchers (with Bruck giving a particularly simple proof in [19]) showed that given some initial state, the network converges to some vector of activations and “stays there”; in other words, the Hopfield dynamics solve some sort of optimization problem and find a local minimum. In fact, from the definitions so far it is easy to see that the dynamics of a Hopfield network can be captured by a single “energy quantity” of the network,  $E = -\frac{1}{2} \sum_{i,j} w_{ij} V_i V_j - \sum_i U_i V_i$ . Every update of the Hopfield network either decreases  $E$  or keeps it the same, and because its range of values is finite, the Hopfield network converges to a local minimum of  $E$  (strictly speaking it could oscillate between several equal-energy states; this indeed happens in some cases, such as with non-reciprocal synapses).

The original (and most important) application of Hopfields networks is to implement *associative memory*: after presenting various patterns (vectors) during training, the network is then presented (in recall mode) a different one (that is not exactly any of the training patterns). It will then converge to the most *similar* one; this is subtly but importantly different than the recall notion of subevents in, say, Marr’s model, where a *partial* activation of a pattern leads to its full activation. In Hebb’s original paper, he numerically found that for various network sizes  $n$ , up to  $0.15n$  patterns could be successfully stored before recall error became severe (such experiments were limited by computational power at the time; Hopfield remarks that even  $n = 100$  was computationally expensive). It turns out that even this linear upper-bound was too optimistic; it was soon shown in 1987 that the “capacity” (maximum number of patterns that can be fully recovered) is upper-bounded by  $n/(2 \log_2 n)$  [20].

#### 1.5.4 “Arbitrary” computation with Hopfield networks

Recall from the previous subsection that the dynamics of Hopfield networks are captured by the energy quantity  $E = -\frac{1}{2} \sum_{i,j} w_{ij} V_i V_j - \sum_i U_i V_i$ ; the Hopfield network converges to a local

minimum of  $E$ . Well, from this it follows that if we could arbitrarily set the  $w_{ij}$  and  $U_i$  however we please, Hopfields networks, in their natural dynamics of optimization, could in fact find optima for some *other* computational problem as long as we could somehow encode the constraints of that problem in the  $w_{ij}$  and  $U_i$ . This is precisely what Hopfield and Tank observed and showed in their seminal 1985 paper ““Neural" computation of decisions in optimization problems" [21].

What follows is a summary of their main result: using Hopfield networks to solve the Traveling Salesman Problem (TSP), the famous NP-complete problem of ordering  $n$  cities, provided with their pairwise distances  $d_{ij} = d_{ji}$ , into a tour (a walk that visits every city exactly ones) that minimizes the total distance. To represent tours neurally, we will use a Hopfield network with  $n^2$  neurons; for each city  $x$ , there is a row of  $n$  Hopfield neurons where the  $i$ -th neuron in the row represents whether city  $x$  is in position  $i$  or not. We will write  $V_{xi}$  to index the neurons (again,  $V_{xi}$  is 1 if  $x$  is in position  $i$  in the tour).

In a valid, desired setting of the Hopfield network, one that actually represents a tour, there is exactly one 1 in each row and column (each city is visited once, and each position can only be used for one city). This constraint is quite easy to represent using an energy quantity (in the version of Hopfield networks where each  $V_{xi} \in \{0, 1\}$ ):

$$E_{\text{tour}} = A \sum_x \sum_i \sum_{j \neq i} V_{xi} V_{xj} + B \sum_i \sum_x \sum_{x \neq y} V_{xi} V_{yi} + C \left( \sum_x \sum_i V_{xi} - n \right)^2$$

It is easy to see that the first term is minimized (set to 0) iff there is at most one 1 in each row, the second term iff there is at most 1 in each column, and the final term iff there are exactly  $n$  active neurons; together these enforce (by setting the constants  $A, B, C$  to be appropriately large) that any local optimum will satisfy all three of these conditions, and hence represent a valid walk. The essential observation is that although  $E_{\text{tour}}$  isn't obviously in the right "energy quantity" form of a Hopfield network, by simply expanding and re-arranging the terms, it in fact *is*. Importantly, one needs only to notice that the equation consists entirely of degree-2 cross terms ( $V_{xi} V_{yj}$  for  $xi \neq yj$ ) and degree-1 terms  $V_{xi}$  (the square in the last term simplifies into degree-1 terms after expansion

because  $V_{xi} \in \{0, 1\}$ ).

Next, to enforce optimal (short) tours, we add an additional term:

$$E_{\text{short}} = D \sum_x \sum_{y \neq x} \sum_i d_{xy} V_{xi} (V_{y,i+1} + V_{y,i-1})$$

It is not hard to see that this sum equals exactly  $D$  times the total weight of the tour represented by the state of the Hopfield network! Hence, with appropriate settings of constants, optimizing  $E_{\text{tour}} + E_{\text{short}}$  finds an optimal solution to TSP. Again, note that just by expanding and re-arranging terms, we can easily rewrite  $E_{\text{tour}} + E_{\text{short}}$  in the canonical Hopfield energy quantity form  $E = -\frac{1}{2} \sum_{xi,yj} w_{xi,yj} V_{xi} V_{yj} - \sum_{xi} U_{xi} V_{xi}$ , where the  $w$  and  $U$  depend on  $A, B, C, D$  and the  $d_{xj}$ .

It turns out, naturally, that *any* cost function that can be written as a Hopfield energy function can be “solved” by encoding it as a Hopfield network whose equilibria are solutions to the optimization problem. Since TSP is NP-complete, this means any NP problem can (in some perhaps very unnatural way depending on the reduction to TSP) be represented and “solved” by Hopfield networks, by representing the cost function / constraints of the problem in the *synaptic weights and firing thresholds* of a Hopfield network. This is essentially the thesis of Hopfield and Tan’s subsequent paper in 1986, “Computing with neural circuits: A model” [22], that a “rich repertoire of problems” can be solved by Hopfield networks, with TSP being their canonical example. Importantly, the significance of the model is that it not only solves a hard computational problem, but it makes explicit *how* synaptic weights, neuron firing and firing thresholds “work together”, i.e. what their conceptual contribution is, to the network’s computational behavior.

**Connections to NEMO.** Hopfield networks have several underlying principles in common with NEMO: binary firing neurons, Hebbian plasticity, and recurrent connectivity as an essential model component. However, they are also very different: Hopfield neurons have (arbitrary) firing thresholds, whereas NEMO’s “threshold” is like that of Marr’s models: a constant fraction of neurons fires, which can be implemented implicitly with local inhibitory interneurons. NEMO allows for multiple brain areas with fibers between them (though one could study just a single NEMO brain

area), and the dynamical system describes the evolution of *all* areas simultaneously, taking into account inter-area and recurrent fibers. As we will see, that allows for much more general behaviors and problem solving.

Something very similar to the basic function of Hopfield networks, storing various patterns and then restoring them, can be performed in NEMO by projecting various stimuli into an area to create assemblies. These can then be pattern completed (though this is perhaps more similar to reproducing Marr's notion of recall from partial events). Most similar to Hopfield's notion is that after projecting multiple stimuli (learning), we can fire some mixtures: the cap in the downstream area will, over several time steps, converge to one of the previously formed assemblies! This operation is used throughout NEMO literature, and is an essential operation in the one of the most recent papers studying the use of NEMO for modeling random variables and Markov chains [23]. However, as before with Marr's models, in Hopfield networks these are separate learning and recall regimes; in NEMO everything is combined, with the same underlying principles underlying *both* how representations (memories) are formed, as well as how they are recalled from partial or perturbed inputs.

Next, there is something to be said about Hopfield's notion of general computation versus our study of NEMO for solving general problems (and Turing-completeness). Hopfield's solution to the TSP problem assumes we can "bake in" synaptic weights and thresholds to our liking; this demonstrates some potential, theoretical computational limit of Hopfield network, but it is completely unclear how synaptic weights / firing thresholds can come to depend on parameters of the computational problem (e.g. distances between cities) in such a subtle and precise way; e.g. particular neurons and particular synapses need to depend precisely on a specific pair of cities, etc. A related issue is that although "sets" of neurons represent configurations of the computational problem, these are far from "assemblies" in the Hebbian sense in that individual neurons encode very specific and fine information in these representations; there is no redundancy or resiliency as expected of assemblies. In NEMO, in particular in language, states of computational problem are represented very differently and as assemblies themselves; for instance if we were trying to

solve TSP, we would perhaps represent each city as an assemblies, each possible position as an assembly, and then strong connections between assemblies as assignments (this is just an example configuration; this thesis does not present a concrete NEMO algorithm for TSP, although it is theoretically possible!) Our Turing-completeness result, which is morally quite similar to Hopfield and Tank's TSP algorithm, is also just a theoretical "exercise" since we assume we can "bake in" a configuration of assemblies representing tape cells, 0s, and 1s. However, for concrete problems (such as in language), in addition to showing that weights can be set up in such a way as to solve a computational problem, we are also interested in *how* such weights are learned (and the dynamics of this must be same in both learning and evaluation, unlike in Hopfield networks where these regimes are separated); this is the difference between Chapters 4 and 7.

#### 1.5.5 ANNs

A frequently asked question is how does NEMO compare to, or differ from, artificial neural nets? While originally inspired by biology, the design of ANNs is not *constrained* by biology; their aim is obtaining the best performance on a particular learning task, and not modeling how the brain works. This is why ANNs perform much better on, and cover a much larger domain of, most tasks for which we have NEMO algorithms; NEMO is, at least at its current stage, not competitive with ANNs. ANNs and NEMO have some superficial similarities: both have brain areas, in some sense (layers of ANNs, though these are usually feed-forward, as recurrent ANNs are harder to train and as of the time of this thesis are not among the dominant architectures in most domains), and represent synapses as single weights. The per-neuron non-linearity of ANNs differs from the  $k$ -cap inhibitory mechanism of NEMO. The most important difference is how synaptic weights evolve over time: whereas ANNs are almost always trained using gradient methods where updates to weights in the network depend on (many) other weights, this is usually believed to be the most biologically implausible aspect about ANNs. The framework of machine learning also distinguishes two fundamentally distinct regimes: that of learning, and evaluation. In evaluation, weights are not updated at all (and in some sense, the evaluation regime is more biologically

realistic). In the brain, however, the same neurons and synapses underlie all functions of the brain. NEMO and this thesis study a particularly simple computational candidate for the brain where there is only one, *Hebbian* synaptic mechanism (representing an abstraction of long- and short-term plasticity). To a reader chiefly interested in ANNs, one could present this thesis and its underlying works as a study of what can be achieved with *only* Hebbian plasticity.

### 1.5.6 Valiant's model

Another generic computational model based on neurobiology is Valiant's model theory of neural items, surveyed in [24]. NEMO draws inspiration from this earlier model in many ways. Valiant's model is also based on random connectivity (concretely, the  $G_{n,p}$  graph), and assumes a fixed threshold for firing. In his model, concepts are represented by a sort of proto-assembly called *items*, which are just sets of neurons that fire together when their input(s) fire. Valiant's theory establishes that items can be created from other items, which is not unlike the project and merge operations of NEMO shown in Chapter 3. However, the crucial difference is that in NEMO, the assemblies that emerge from the system have all the characteristics of such *items* and many more: high internal connectivity, and various memory-like properties (firing an assemblies will continue to activate itself; pattern completion, etc). In the versions of Valiant's model that include plasticity, plasticity can be an arbitrary function (an arbitrary function at the post-synaptic site); with NEMO we restrict this to a single Hebbian rule, and studies the computational implications of this assumption.

## 1.6 Experimental study of assemblies.

One day, an ultimate confirmation of a theory like NEMO would include an experimental demonstration that there are assemblies, or sets of consistently co-firing neurons, whose activation is functionally and causally connected to a specific behavior, concept, etc. However, much stands in the way of this lofty goal. Put simply, the problem with the experimental study of assemblies is that it is difficult to record from multiple neurons simultaneously. We could try to use single-cell recordings, but this cannot really capture assemblies since they are an *emergent* property of many

individual neurons.

However, there are several techniques that can record from multiple neurons simultaneously and which have already, to some extent, been used successfully to find evidence for assemblies. The following overview is based on Yuste, Cossart and Yaksi's very recent survey on ensembles "Neuronal ensembles: Building blocks of neural circuits" [25]. The main ones are *multielectrode arrays* (MEA), which directly records from multiple neurons simultaneously, and voltage or calcium imaging, which looks at chemical (e.g. calcium) concentration changes that are resultant from neuronal firing. These are the main techniques for studying "multi-neuronal activity patterns" more generally, which includes assemblies/ensembles. Briefly, ensembles have been identified in both hippocampus and cortex. Hippocampus is the first area where significant multi-neuronal activity patterns were found, as repeated coordinating firing of subsets of excitatory (pyramidal) neurons (actually, the coordinated activity was the sequential firing of neurons, which could be activated artificially outside of the natural environment where they were recorded). Such ensembles were found to be anatomically scattered throughout hippocampus, and sparse, and there is evidence that activation of one seems to inhibit activation of others (they are in competition); interestingly, all of these properties are inline with the NEMO model. Most importantly these hippocampal ensembles can be causally linked to behavior. There is also a line of work finding similar results, i.e. ensembles with similar properties, in cortex (that is, co-active, or sequentially active, groups of neurons; these were found mostly with voltage/calcium-imaging techniques, as well as electrical recordings). As a specific example, a recent paper studied monkey motor behavior by implanting them with MEAs in frontal and parietal cortex, and recorded from them performing arm movement tasks. While the degree of correlation between the activation of *single* neurons and the animals' movement was unstable, very stable predictions could be obtained by looking at the activity of whole sets (in the authors' words, ensembles) of neurons, supporting the thesis that even something like arm movement is encoded in the brain via assemblies, with all their redundancy as a mode of encoding.

Despite all these advances, "proving" the existence and role of ensembles remains difficult.

The main challenges are that even with simultaneous recordings over some time period, care is needed to identify an ensemble instead of random correlations; sophisticated statistical models of firing are developed for this, which need to take into account neuronal firing “jitter”. In general, noise from recordings leads to the *underestimation* of neuronal ensembles. Another issue is that the time window in the relationship between neurons can vary greatly between types of neurons and behavior; co-activations or sequential activations of neurons could take place between a few hundred milliseconds up to several seconds. Due to neuromodulation, the size and speed of assemblies can also change over time; while this may be one of the computational *features* of assemblies, it also makes their identification and study more challenging.

## 1.7 Techniques in neurolinguistics

Since several chapters of this thesis study NEMO algorithms for linguistic problems, some review of the neuroscience of language is in order. There are subsections of the relevant Chapters 4 and 7 that survey some of the most relevant neurolinguistics to those problems to show how the NEMO models and algorithms broadly reflect what is known about language in the brain. Hence, in this section we will chiefly concern ourselves with a general overview of the *techniques* in neurolinguistics and, in particular, those that record directly from the brain. That is, while fMRI and EEG/MEG studies are very important in languages (and among those referred to in the neuroscience sections of Chapters 4 and 7), direct-brain recording techniques are more likely to be those through which ensembles for *linguistic* function are experimentally established.

### 1.7.1 The main methodologies

The main direct brain-recording (or rather “intracranial”) experimental methods used in studying language are (1) direct electrical stimulation, (2) single-cell electrode recordings, (3) electrocorticography (ECoG) and most recently (4) multielectrode arrays and neuropixels. Kemmerer’s classic neurolinguistic textbook [26] gives a thorough overview of these (as well as non-intracranial) techniques, and the review here largely follows his presentation.

**Direct electrical stimulation** is a major intracranial method in language, especially historically. Originally developed as a technique for studying epilepsy and seizures more generally, this technique involves using a handled electrode to directly stimulate some portion of the brain. Most work of this kind shows *negative* results; that stimulation can impair specific components of language function, which is important in studying separations between linguistic functions and in the anatomical localization. Ojemann's landmark studies in the late 80s identified a specific "mosaic" of sites whose stimulation disrupted object naming from images, and whose locations varied greatly among patients. Later, other studies showed the cortical separation of languages in multilingual people [27], dissociations (that is, separate disruptability) of language production and comprehensions [28], dissociations between nouns and verbs (disrupting ability to name/produce one but not the other [29]; this is highly related to our treatment of nouns and verbs as having separate lexical areas in Chapter 7), as examples. Direct electrical stimulation has been described as the "gold standard for brain mapping" [Mandonnet2009], but it of course has many limitations, as the effects of passing current through cortex are not well understood.

**Single cell recordings.** There is very limited work using single-cell electrodes to study language in the brain, but a representative work is Creutzfeldt's seminal 1989 study "Neuronal activity in the human lateral lobe, responses to speech" [30] which identified individual neurons in the right STG (superior temporal gyrus) that responded to specific features of linguistic auditory perception: that is, neurons that are varyingly receptive to the phonemic, syllabic, or morphological structure of words. For instance, one cell responded significantly to velar consonants (k or g) in combination with specific other consonants (r and s). A more recent and systematic survey identifies neurons that fire robustly in response to particular words, or to complex but specific sets of phonemes and in a way that is invariant to the speaker [31].

**Electrocorticography** is an extremely promising technique that implants small electrode arrays directly into cortex of epileptic patients. A representative study is a 2011 study by Flinker et al. [32] which presented implanted patients with synthesized syllables, pseudowords, real words and

proper names. The conclusion was the identification of specific electrode sites with activity in response to both phonemes and words in a short time frame, and others very consistently only to words, pointing towards a hierarchical organization of speech processing (phonemes to syllables to words). Other ECoG studies have tried to *decode* linguistic features from the ECoG data (that is, by fitting a model on top of the ECoG data), recorded during linguistic perception of production tasks, and have successfully done so in certain domains, e.g. extracting parts of the phonetic form of words; see [33] for a survey.

**Microelectrode arrays and Neuropixels** are a recent breakthrough technology in experimental neuroscience that allow for the simultaneous electrical recording from hundreds of neurons. As an extremely new technique, it is yet to be widely used to study language. However, a representative example is the 2024 study “Large-scale single-neuron speech sound encoding across the depth of human cortex” by Leonard et al [34] which recorded from 685 neurons simultaneously at nine-sites in the STG while presenting sentences. They discovered individual neurons sensitive to a range of features (consonants, vowels, pitch, onsets, etc), and finding that neurons at similar cortical depths encoded similar features.

### 1.7.2 Songbirds

Another field of study worth mentioning in relation to the theoretical study of language in the brain is the study of *songbirds*, whose song system, while very different from human language overall, bears a lot of similarity to human phonology and in particular in the domain of phonological learning (i.e. how babies learn which sounds make up their language has parallels with how baby birds learn song from adult birds). Two surveys give an overview of these connections: “Translating Birdsong: Songbirds as a model for basic and applied medical research” by Brainard and Doupe [35] and “Analogies of human speech and bird song: From vocal learning behavior to its neural basis” by Zhang et. al [**Zhang2023**].

The same basic principle underlies both song and human language learning: a feedback loop

of receiving auditory input, and repeated vocal imitation on the part of the learner, until a stable pattern of output is learned. Other similarities in these “linguistic” systems is that both human phonology and bird song have “syntactic” properties (phonological / phonotactic in the form of language, i.e. rules governing how sounds can be combined or affect each other in human language, and similar principles have been studied in bird song), and both species have “critical periods”, ages during which acquisition must happen before it is too late, suggesting that perhaps similar neural mechanisms (a period when plasticity is particularly high, or pruning occurs) is involved (note that the exact nature or extent of critical periods in humans is debated). These similarities, and the same fundamental learning paradigm, were achieved in both species through parallel evolutionary branches.

There is also high-level parallelism between the neural systems in the phonological systems of both species. Humans, songbirds, and non-human apes all have a laryngeal motor cortex pathway; i.e. parts of cortex can control the larynx and hence vocalizations. But only in humans and songbirds is there an additional *language learning pathway*, which in birds is localized in the anterior forebrain, which exchanges input to and from the laryngeal motor cortical pathway in both species.

The linguistic problems tackled in this thesis are chiefly on the syntactic and semantic linguistic levels; the question of how NEMO dynamics could be used to model phonological learning is an interesting, but currently open problem. However, when this is eventually studied, one could instead focus on modeling songbird phonology since this may not only be simpler but has more experimental neural data.

## Chapter 2: The NEMO model

This chapter introduces the NEMO family of models, which are the object of study throughout the rest of the thesis. While improved for the purpose of this thesis, these models were originally defined in with Christos Papadimitriou, Santosh Vempala, Michael J. Collins, and Wolfgang Maass in [36, 37]. The “basic-NEMO” model is the set of biologically plausible assumptions that are common to all versions of the NEMO model used in this thesis and related works. The full NEMO model is the basic-NEMO model with one additional computational ingredient (alternatively, abstraction of neurobiological principle). Finally, there are several additional variants of NEMO that are studied less often, but which are nonetheless important and are summarized at the end of this chapter.

### 2.1 The basic-NEMO model

In these models, the brain is divided into a finite number  $a$  of brain *areas* denoted  $A, B, C, \dots$  each containing  $n$  excitatory neurons. These are intended to represent a anatomically and functionally meaningful partition of the cortex, the largest part of human and primate brains that is known to be used flexibly and adaptively across almost all cognitive tasks. These areas are initialized with *random* connections; that is, each ordered pair of neurons in an area is connected by a synapse with the same probability  $p$ , independently of what happens to other pairs. That is, each area internally is a random weighted directed *Erdős–Renyi random graph*. or  $G_{n,p}$  (this is a well studied mathematical random graph model [38]). Each synapse  $(i, j)$  has *synaptic weight*  $w_{ij} > 0$  initialized to 1, but which will change dynamically.

Each ordered pair of areas, say  $(A, B)$  with  $A \neq B$  may or may not be *connected*. If they are connected, then there is a random directed *bipartite* graph connected neurons in  $A$  to neurons in

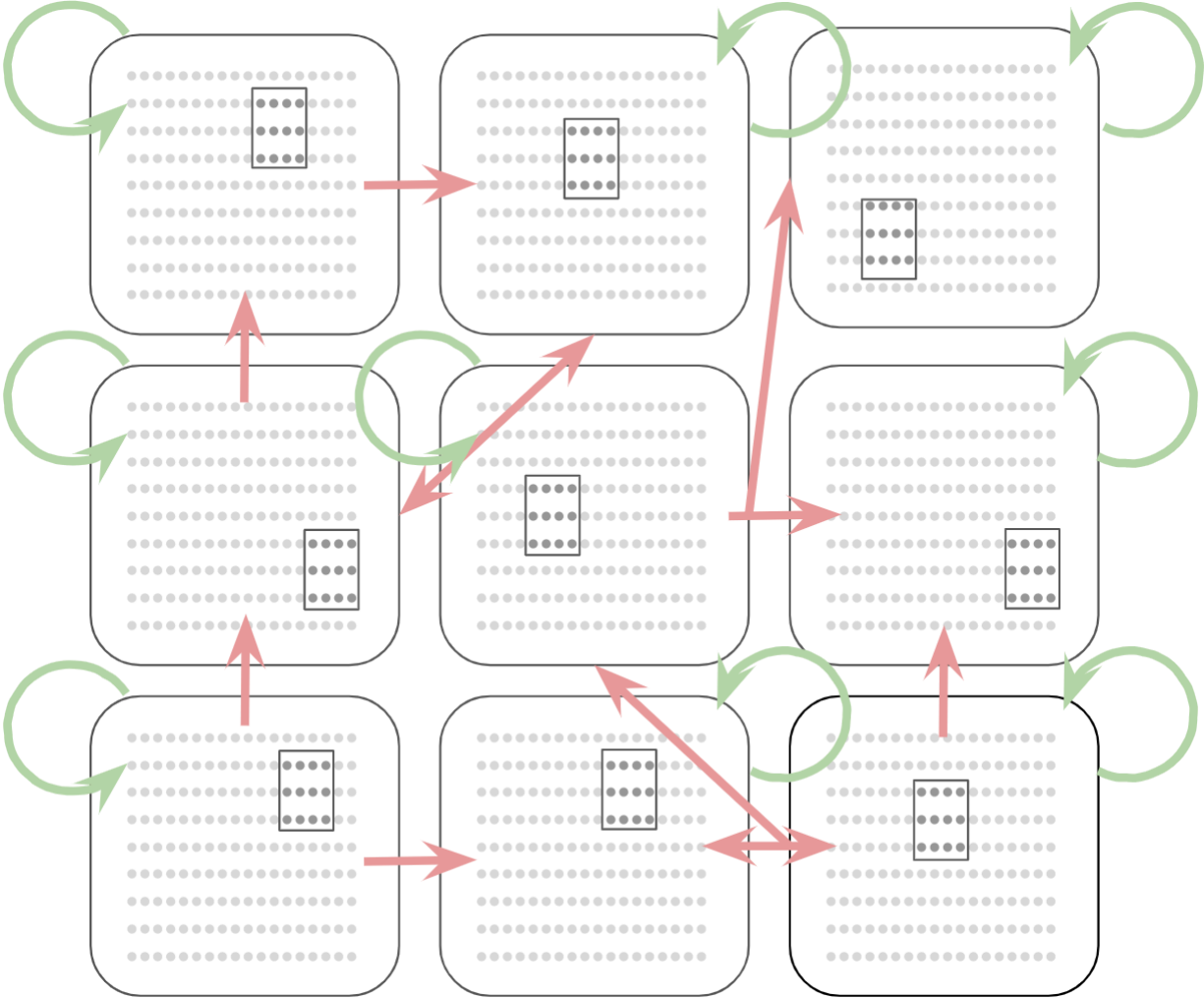


Figure 2.1: An instance of the NEMO model with 9 areas. All areas have recurrent connections (green arrows), and some ordered pairs of areas have *fibers* between them (red arrows).

*B*. That is, for every neuron in *A* and every neuron in *B* there is a probability  $p$  of a synaptic connection from the former to the latter (if sampled, it is initialized with weight 1). We will sometimes say that *A* is connected to *B*, or equivalently that there is a *fiber* from *A* to *B*.

The model also encompasses simplified forms of *local inhibition and plasticity*. Indeed, for two areas *A* and *B* that are both connected to each other, there is nothing that distinguishes the model from a single large area. The effect of local inhibitory neurons (whose output is thought to apply locally across all neurons in a single area) is that at any time step, exactly  $k$  neurons in each area can fire: the  $k$  neurons with the top synaptic inputs among all neurons in that area. Finally, we

assume one of the simplest possible abstractions of local plasticity, namely *multiplicative Hebbian plasticity*. if at a time step neuron  $i$  fires and at the next time step neuron  $j$  fires, and there is a synapse from  $i$  to  $j$ , the weight of this synapse is multiplied by  $(1 + \beta)$ , where  $\beta > 0$  is the final parameter of basic-NEMO.

That is, events happen in discrete time steps (which we think of corresponding to roughly 20 ms of time in the brain). The *state* of the model, which is in fact a dynamical system, at each time step  $t$  consists of (a) for each neuron  $i$  a bit  $f_i^t \in \{0, 1\}$  denoting whether or not  $i$  *fires* at time  $t$ , and (b) the synaptic weights  $w_{ij}^t$  of all synapses in  $E$ . Given this state at time  $t$ , the state at time  $t + 1$  is computed as follows:

1. For each neuron  $i$  compute its *synaptic input*  $SI_i^t = \sum_{(j,i) \in E, f_j^t=1} w_{ji}^t$ , that is, the sum total of all weights from pre-synaptic neurons that fired at time  $t$ .
2. For each neuron  $f_i^{t+1} = 1$  — that is,  $i$  fires at time  $t + 1$  — if  $i$  is among the  $k$  neurons *in its area* with the highest  $SI_i^t$  (breaking any ties arbitrarily).
3. For each synapse  $(i, j) \in E$ ,  
 $w_{ij}^{t+1} = w_{ij}^t (1 + f_i^t f_j^{t+1} \beta)$ ; that is, a synaptic weight increases by a factor of  $1 + \beta$  if and only if the post-synaptic neuron fires at time  $t + 1$  and the pre-synaptic neuron had fired at time  $t$ .

We call the set of  $k$  neurons in an area firing at time  $t$  the *cap* of that area. Sometimes we will refer to the cap as the area's *winners* at that time step. The basic-NEMO dynamical system can also be captured more succinctly (and perhaps intuitively) via matrix equations. Formally, with  $f_A(t)$  the indicator vector of neurons firing in  $A$ , and  $W_{A,B}$  the matrix of weights from area  $A$  to  $B$ , and  $\mathcal{A}$  the set of all areas in the basic-NEMO instance, the set of equations defining the dynamical system are

$$\forall A \in \mathcal{A}, f_A(t+1) = k\text{-cap} \left( \sum_{B \in \mathcal{A}} W_{A,B}(t) f_B(t) \right)$$

$$\forall A, B \in \mathcal{A}, W_{A,B}(t+1) = W_{A,B}(t) + f_A(t+1) \cdot f_B(t)^\top \odot (\beta \cdot W_{A,B}(t))$$

In summary, a basic-NEMO instance is defined by a choice of the following *parameters*:

- The number  $a$  of brain areas and their labels;
- The size  $n$  of brain areas;
- For each pair of brain areas, a choice of whether it is connected or not;
- The probability  $p$  of synapses (both recurrently within areas, and between connected areas)
- The size  $k$  of caps (effect of local inhibition);
- The plasticity coefficient  $\beta$ .

### 2.1.1 Parameter settings

Having the NEMO family of models have so many parameters, in particular constants like  $n, p, k$  and  $\beta$  presents the difficulty that we are simultaneously asking whether algorithms exist in these models for certain problems, and for which parameter settings. On one hand, if we can find an algorithm that works with *any* setting of the parameters, we will pronounce the model as being capable of implementing algorithms for that problem. However, the result will be more interesting (that is, be even more biologically plausible) when the parameters are in a neurobiologically reasonable range (and generally, we experiment in these ranges to begin with). This range is hard to pin down, but roughly, we want  $n$  to be between  $10^4$  and  $10^7$ ,  $k$  to be around  $\sqrt{n}$ ,  $p$  is in the range  $[0.005, 0.05]$  (that is, probability of synapses is between 0.5% and 5%; in fact, in a more complicated version of the model we might have different  $p$  between different areas or pairs of areas; see the following discussion). The plasticity parameter  $\beta$  is usually in the range  $[0.005, 0.05]$ .

To give some early intuition, for many of the algorithms that we find work (or converge) in NEMO, larger values of the plasticity coefficient  $\beta$  result in *faster* convergence, sometimes to the point of triviality (so we would like if these algorithms work even with very low  $\beta$ ), and render many of our proofs simpler. Varying other parameters, such as  $p$  or the ratio between  $k$  and  $n$  can have varying effects (increasing convergence or entirely destabilizing an algorithm).

## 2.2 NEMO

The full **NEMO** model is defined with an additional ingredient: long-range inhibition, or long-range interneurons (LRIs). The full model contains all the ingredients of basic-NEMO; that is, an instance is defined by the same choice of areas and parameters as before, with the addition of:

- A number of LRIs  $l_1, \dots, l_L$ ;
- Each pair of area  $A$  and LRI  $l_i$  may be connected, in which case each synapse from  $A$  to  $l_i$  is sampled with probability  $p$ .
- Each LRI  $l_i$  has a *target*  $T(l_i) \in \mathcal{A} \cup (\mathcal{L} \setminus \{l_i\})$ ; that is, either an area  $A$  or another LRI  $l_j \neq l_i$ ;
- An LRI firing threshold  $\tau$ .

We denote by  $f_{l_i}(t)$  whether LRI  $l_i$  fires at time step  $t$ . In addition to the previous rules of the dynamical system of basic-NEMO, whenever LRI  $l_i$  fires at time step  $t$ ,  $f_{T(l_i)}(t+1)$  is automatically set to 0; that is, it *inhibits* its target (either an entire area, or another LRI). Finally, an LRI fires at time step  $t$  whenever its pre-synaptic input at time step  $t - 1$  is greater than  $\tau$ , and it is not inhibited as the target of another LRI that fired at time step  $t - 1$ .

On one hand, LRIs turn out to be a very powerful ingredient: they enhance a model that can do basic operations on assemblies of neurons (a concept yet to be formally discussed) to a computation system that can parse the context-free languages, and much more. On the other hand, we point out that this way of defining LRIs makes their power rather blunt: they cannot inhibit individual excitatory neurons in other areas (which would allow to set up very fine-grained circuits); rather, their firing inhibits an *entire* target area.

## 2.3 Other extensions of NEMO

The basic-NEMO and NEMO models are the two most important models studied so far. However, there are many other kinds of biological detail that we could have included, or other ways we could have defined the same principles. Occasionally, we will study other variant models, a few of which are introduced here.

**Homeostasis** . The NEMO models, as described so far, would result, through plasticity, in gigantic synaptic weights after a long time of operation. An extension of the model includes a process via which weights are renormalized, at a slower time scale, so that the sum of presynaptic weights at each neuron stays relatively stable. There is a homeostasis function  $h(\cdot)$  such that at each time step, after caps are computed and the plasticity rule is applied, each weight  $w_{ij}$  is updated to  $h(w_{ij})$ . We will call versions with homeostasis *(basic-)NEMO with homeostasis*.

**Varying  $n, k, p, \beta$**  . The parameters of basic-NEMO are defined as being the same throughout the entire “brain” represented by an instance of the model. It is only a slight extension to consider a version where the model has parameters  $n_A, k_A, p_A$  and  $\beta_A$  for each area  $A$ , and  $p_{(A,B)}$  for each ordered pair of areas  $(A, B)$ , We will call these versions *area-grained (basic-)NEMO*.

**Other models of plasticity** The multiplicative Hebbian rule is one of the crudest assumptions in our model (indeed it results in the potential of exploding weights, for which homeostasis is one possible correction). Another version is an *additive* Hebbian rule, where if  $i$  fires at time  $t$  and  $j$  fires at time  $t + 1$ , we apply the rule  $w_{ij} \mapsto w_{ij} + \beta(w_{ij})$ . The function  $\beta(\cdot)$  may just be a small constant, or it may for instance be an inverse exponential (which at least in part fulfills the role of homeostasis). This last version is particularly interesting, as recent work (outside the scope of this thesis) has shown that NEMO with such an additive Hebbian rule can be used to learn arbitrary discrete conditional distributions, and more generally Markov chains [**flippingcoin**].

**Other models of connectivity** . Several experimental results [39, 40] suggest deviations of the synaptic connectivity of the animal brain from the uniformity of  $G_{n,p}$ . While we are able to achieve interesting phenomena and algorithms even with this (arguably) simplest model of connectivity, one can also define and study versions of NEMO where the initial random graph is something other than  $G_{n,p}$ .

## **Chapter 3: Computing with NEMO: Creation of Assemblies and Basic Operations**

This chapter uses material from [36, 37]. The material has been refurbished for the purposes of this thesis, but the bulk of the ideas and results presented represent the collaborative effort with the original authors: Christos Papadimitriou, Santosh Vempala, Michael J. Collins, and Wolfgang Maass.

### **3.1 The computational power of basic-NEMO**

. What is the computational power of basic-NEMO— can any interesting phenomena, or algorithms, be implemented with the very minimal set of assumptions of (uniform) random connectivity, local inhibition, and Hebbian plasticity?

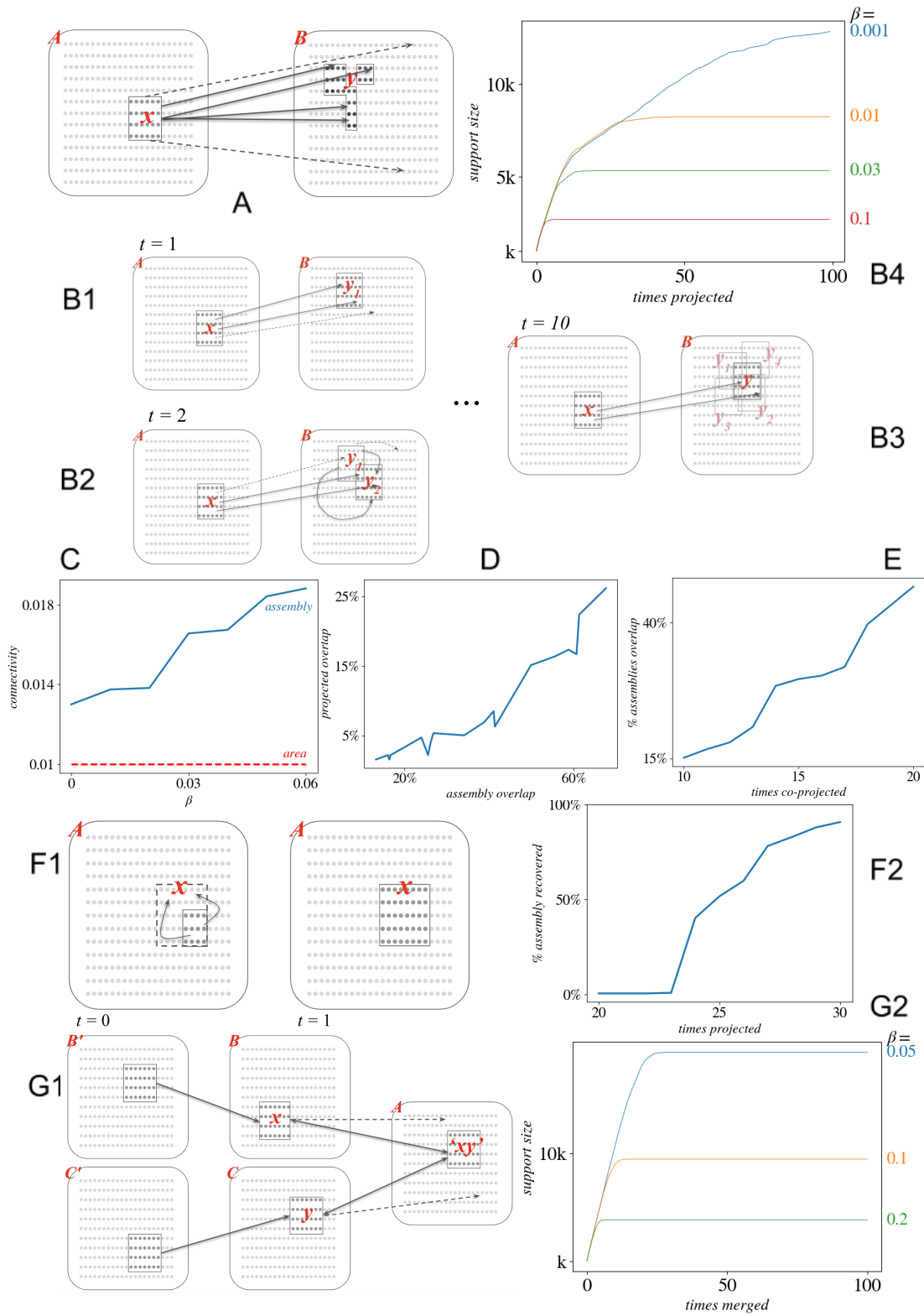


Figure 3.1: Illustrations and simulations of basic-NEMO operations; each subfigure is discussed in the corresponding subsection. In these experiments,  $n = 10^6$ ,  $k = 10^3$  and  $p = 0.01$

### 3.1.1 Assembly creation

Recall the *assembly theory* of neural computation; that subsets of neurons with high intraconnectivity and that fire together represent cognitive concepts in the brain, and that computations on these concepts are realized as manipulations of these assemblies. How do such assemblies in the association cortex come about? It has been hypothesized (see e.g. [41]) that an assembly imprinting, for example, a familiar face in a subject's medial temporal lobe (MTL) is created by the *projection* of a neuronal population, perhaps in the inferotemporal cortex (IT), encoding this face as a whole object. A similar process is predicted in the discussion of [6], where neuronal subsets fire from the olfactory bulb into the piriform cortex, resulting in assemblies in piriform cortex through repeated firing, plasticity, and local inhibition.

These biological findings and hypotheses motivate the following experiment, which is also perhaps the simplest experiment that we could define. We consider an instance of basic-NEMO with just two brain areas  $A$  and  $B$ , with a fiber from  $A$  to  $B$ . Suppose in area  $A$ , we choose a fixed subset  $x$  of  $k$  neurons to fire at all time steps  $t = 1, 2, \dots$  (within an area is used in this way—purely as input for firing into other areas— we sometimes call it a *stimulus*. Initially, at time step  $t = 1$ , no neurons in  $B$  fire. Since  $x$  in area  $A$  fires at times  $1, 2, \dots$  (and ignoring all other areas), it will effect at times  $1, 2, \dots$  the firing of an evolving set of  $k$  neurons in  $B$  — a sequence of caps —, call these sets  $y^1, y^2, \dots$ . At time 1,  $y^1$  will be simply the  $k$  neurons in  $B$  that happened to receive the largest synaptic input from  $x$  (since all weights are initialized to 1, these are also the neurons with the most edges from  $x$ ), as shown in Figure ?? A. At time 2, however,  $y^2$  will be the set of neurons in  $B$  that receive the highest synaptic input from  $x$  and  $y^1$  combined — and recall that the synaptic weights from  $x$  to  $y^1$  have increased. If this continues, we show that with high probability (where the probability space is the random connectivity of the system), the sequence  $\{y^t\}$  eventually converges to a stable assembly  $y$  in  $B$ . This is shown in Figure ?? B1-3.

We call this set the *projection* of  $x$  in  $B$ , or  $\text{proj}(x, B)$ . This set not only comes about in ways reflecting how we believe assembly formation takes place in the brain, but also has critical properties that urge this interpretation. Hebb [8] hypothesized that assemblies are *densely intraconnected*

— that is, the chance that two neurons have a synaptic connection is significantly larger when they belong to the same assembly than when they do not — and our analysis and simulations verify this hypothesis for sets obtained as projections of stimuli (random subsets in another area, as in the algorithm described previously). From the point of view of computation this is rather surprising, because the problem of finding a dense subgraph of a certain size in a sparse graph is a known difficult problem in computer science [42], and thus the very existence of an assembly may seem surprising. How can the brain deal with this difficult computational problem? The creation of an assembly through projection as outlined in the previous paragraphs provides an explanation: Since the elements of assembly  $y$  were selected to have strong synaptic input from the union of  $x$  and  $y$ , one intuitively expects the synaptic recurrent connectivity of  $y$  to be higher than random. In addition, the weights of these synapses should be higher than average because of plasticity.

Our projections also have the property that they *co-fire* (a hypothesized, essential function of neurobiological assemblies) in three increasingly strong senses:

- If we set  $x$  and  $y$  to fire, at the next time step  $y$  fires.
- If we fire just  $x$  (that is, temporarily set the winners of  $B$  to the empty set:  $f_B(t) = 0^N$ ), within a small number of time steps,  $y$  is always the firing set in  $B$  (in fact, as soon as  $y$  is the set to fire in  $B$ , if the previous condition is satisfied, then clearly  $y$  will continue to fire in perpetuity)
- If we fire just the set  $y$  (that is,  $f_A(t) = 0^N$ ), then at the next time step  $y$  fires (and hence in perpetuity).

These three notions of co-firing are slightly different; the third, in particular, is a sort of stability (or memory-like) property. We find that in any basic-NEMO parameter setting, an increasing number of firing time steps during projection is needed to achieve the stronger notions of co-firing, but all 3 are achieved relatively quickly in reasonable parameter settings. In Figure ?? B4, we simulate and plot the total support size, or the total number of neurons in the downstream area to have ever fired, for varying levels of  $\beta$ .

### 3.1.2 Association

In a recent experiment [43], electrocorticography (eCoG) recordings of human subjects revealed that a neuron in a subject's MTL consistently responding to the image of a particular familiar place — such as the Pyramids — starts to also respond to the image of a particular familiar person — say, the subject's sibling — once a combined image has been shown of this person in that place. A compelling parsimonious explanation of this and many similar results is that two assemblies imprinting two different entities adapt to the cooccurrence, or other observed affinity, of the entities they imprint by increasing their overlap, with cells from each migrating to the other while other cells leave the assemblies to maintain its size. Assemblies are large and in many ways random sets of neurons, and as a result any two of them, if in the same area, may overlap a little by chance. If the assembly size  $k$  is about less than the square root of  $n$ , as we often assume in our simulations, this random overlap, if any, should be very few cells. In contrast, overlap resulting from association is quite substantial: the results of [43] suggest an overlap between *associated* assemblies in the MTL of about 8 – 10% of the size of an assembly. The association between assemblies evokes a conception of a brain area as the arena of complex association patterns between the area's assemblies

The basic-NEMO model can implement this phenomenon in an extremely simple way. Suppose two assemblies in two different areas  $A$  and  $B$  have been independently projected into a third area  $C$  to form assemblies  $x$  and  $y$ , and subsequently, the two parents assemblies fire simultaneously. Then, each of  $x, y$  will respond by having some of its neurons migrate to the other assembly. We call this operation the *association* of  $x$  and  $y$ . In our simulations, we fire both  $A$  and  $B$  into  $C$  for  $T$  time steps, and vary  $T$  to study the resulting overlap in  $x$  and  $y$ . In Figure ?? D, the overlap of projected assemblies is plotted with respect to the overlap of input assemblies. The operation to *increase* overlap is simulated and summarized in Figure ?? E.

### 3.1.3 Pattern completion

Another important and well studied phenomenon involving assemblies is *pattern completion*: the firing of the whole assembly  $x$  in response to the firing of a small subset of its cells [44]; presumably, such completion happens with a certain a priori probability depending on the particular subset firing.

In the basic-NEMO model, we can implement pattern completion happens in a rather striking way, with small parts of the assembly being able to complete very accurately the whole assembly. Assume we have created an assembly is created in an area  $A$  by repeated firing of its parent; we vary the number of times the parent fires as a parameter  $T$ . Afterwards, we stop firing the parent, and instead fire 40% of neurons of the assembly are selected at random and fire for a small, fixed number of steps, and then observe the overlap of the resulting cap with the original assembly. This is illustrated in Figure ?? **F1**. As shown in Figure ?? **F2**, we are able to recover much if not all of the original assembly; with more reinforcement of the original assembly, the subset recovers nearly all of the original assembly.

### 3.1.4 Reciprocal project

Reciprocal projection is a version of projection with strong backward synaptic connectivity from the resulting assembly to its parent assembly or stimulus. Reciprocal projection has been hypothesized to be instrumental for implementing *variable binding* in the brain, such as designating “cats” as the subject of the sentence “cats chase mice,” see [45]. The plausibility of reciprocal projection has been experimentally verified through detailed simulations of networks of spiking neurons with STDP [45].

Reciprocal projection can be implemented in basic-NEMO with slightly more time-steps *or* only slightly stronger plasticity. As in projection, we have two areas  $A$  and  $B$ , but this time, let  $x$  be an assembly (that is, not just a stimulus but a set itself formed through projection from some parent area, and hence with co-firing stability properties). The difference is that this time, we also have a fiber from  $B$  to  $A$ . At time step 1, we fire only  $x$ , but as the system evolves, there

is also synaptic connectivity from  $B$  to  $A$  (in addition to connectivity from  $A$  to  $B$ ) which causes in the next step  $x$  to move slightly to a new assembly  $x_2$ , while  $y_1$  becomes  $y_2$ . This continuing interaction between the  $x_t$ 's and the  $y_{t\pm 1}$ 's eventually converges, albeit slower than with ordinary projection, which can be compensated by conditions of ampler synaptic connectivity and plasticity. The resulting assembly  $y$  has strong synaptic connectivity both to and from  $x$  (instead of only from  $x$  to  $y$ , as is the case with ordinary projection). We demonstrate that reciprocal projection works as described above has been shown both analytically and through simulations in our model.

### 3.1.5 Merge

Linguists had long predicted that the human brain is capable of combining, in a particularly strong sense, two separate entities to create a new entity representing this specific combination [46, 47], and that this ability is *recursive* in that the combined entity can in turn be combined with others. This is a crucial step in the creation of the hierarchies (trees of entities) that seem necessary for the syntactic processing of language, but also for hierarchical thinking more generally (e.g., deduction, discourse, planning, story-telling, etc.). Recent fMRI experiments [48] have indicated that, indeed, the completion of phrases and sentences (the completion of auditory stimuli such as “hill top” and “ship sunk”) activates parts of Broca’s area — in particular, the pars opercularis BA 44 for phrases, and the pars triangularis BA 45 for sentences. In contrast, unstructured word sequences such as “hill ship” do not seem to activate Broca’s area. Recall that Broca’s area has long been believed to be implicated in the syntactic processing of language.

A parsimonious explanation of these findings is that phrases and sentences are represented by assemblies in Broca’s area that are the results of the `merge` of assemblies representing their constituents (that is, assemblies for words such as “ship” and “sunk”); presumably the word assemblies reside in Wernicke’s area implicated in word selection in language. As these hierarchies need to be traversed both in the upward and in the downward direction (e.g., in the processes of language parsing and language generation, respectively), it is natural to assume that merge must have *two-way connections* between the new assembly and the constituent assemblies.

We present an algorithm for implement merge in basic-NEMO, which is by far the most complex of the algorithms of this chapter, and among all known algorithms of basic-NEMO. It involves the coordination of five different brain areas, with ample reciprocal connectivity between them, and requires stronger plasticity than other operations. Merge is essentially a double reciprocal projection. We first create assemblies  $x$  and  $y$  in areas  $B$  and  $C$  (and omit the labels of their two parent areas which act as stimuli). We have a fifth area  $A$ , with fibers to and from areas  $B$  and  $C$ . We then initialize the dynamical system to fire the parents of  $x$  and  $y$ , which initiates firing of  $x$  and  $y$  into  $A$ , and as in reciprocal projection, firing back from  $A$  into  $B$  and  $C$  which destabilizes  $x$  and  $y$  in the first few time steps. However, this interaction eventually converges, albeit slower (or with more demanding settings of  $\beta$ ), whereby a new assembly  $z$  is eventually formed in area  $A$ . The assemblies  $x$  and  $y$  in  $B$  and  $C$  may be modified in this process. In the resulting assemblies there is strong two-way synaptic connectivity between  $x$  and  $z$ , as well as between  $y$  and  $z$ . This is illustrated in Figure ?? **G1**, and experiment convergence results are shown in **G2**, where the total support size (number of neurons to ever fire in area  $A$ , the merge area) is plotted against time.

## Chapter 4: Parsing in NEMO

This chapter uses material from [49] The material has been refurbished for the purposes of this thesis, but the bulk of the ideas and results presented represent the collaborative effort with the original authors: Christos Papadimitriou and Michael J. Collins.

### 4.1 Overview

Language is a distinguishing human function involving the creation, articulation, comprehension, and maintenance of hierarchically structured information about the world. It is beyond doubt that language is achieved through the activity of neurons and synapses — but how? There has been extensive previous work in cognitive experiments — psycholinguistics, computational psycholinguistics, and brain imaging — that has led to many insights into how the brain processes language (see section ?? for an overview). However, no concrete narrative emerges yet from these advances about the precise way in which the activity of individual neurons can result in language. In particular, *we are not aware of an experiment in which a reasonably complex linguistic phenomenon is reproduced through simulated neurons and synapses.* This is the direction pursued here.

Developing an overarching computational understanding of the way neurons in the human brain can make language is hindered by the state of neuroscience, which (a) predominantly studies sensory and motor brain functions of animals other than humans; and (b) with respect to computational modeling, focuses on the level of neurons and neuronal circuits, and lacks the kind of high-level computational model that seems to be needed for understanding how high-level cognitive functions can emerge from neuronal activity.

However, the results of the previous chapter raises the question of whether the NEMO model is a good, or even sufficient computational framework to model language. We recall that NEMO

describes a dynamical system involving the following parts and properties, well attested in the neuroscience literature: a) brain areas with random connections between neurons; b) a simple linear model of neuronal inputs; c) inhibition within each area such that the top  $k$  most activated neurons fire; d) a simple model of Hebbian plasticity, whereby the strength of synapses increase as neurons fire. An important object emerges from these properties: the *assembly*, a large set of highly interconnected excitatory neurons, all residing in the same brain area.

Indeed, assemblies have been hypothesized by Hebb seven decades ago [50], and were identified in the brain of experimental animals two decades ago [51]. There is a growing consensus that assemblies play a central role in the way brains work [52], and were for instance recently called “the alphabet of the brain” [53]. Assemblies can, through their near-simultaneous excitation, represent an object, episode, ideas, and perhaps *words* (or rather, morphemes) and other linguistic constituents. Importantly, as shown in the previous chapters, assemblies are an emergent property of the dynamical system under conditions (a) – (d) above, both in theory and in simulations.

The dynamical system also makes it possible to create and manipulate assemblies through *operations* like projection, reciprocal projection, association, pattern completion, and merge. These operations are *realistic* in two orthogonal senses: first, they correspond to behaviors of assemblies that were either observed in experiments, or are helpful in explaining other experiments. Second, they provably correspond (they “compile down”) to the activity of individual neurons and synapses; this correspondence is proven both mathematically and through simulations. In the original paper in which we discovered and studied these operation, we hypothesized that NEMO may underlie high-level cognitive functions [37]; in particular, in the discussion section of that paper it is proposed that one particular operation, *merge*, may play a role in the generation of sentences.

Note that, regarding the soundness of NEMO, what has been established through mathematical proof is that NEMO operations work correctly *with high probability*, where the underlying probabilistic event is the creation of the random connectome. So far, the simulation experiments conducted with NEMO have demonstrated that individual commands of NEMO, or short sequences thereof, can be successfully and reliably implemented in a simulator. However, the proofs and

experiments of the previous chapters concern small manipulations of 1 to 5 brain areas and at most several assemblies; an important open problem left open by this work is: do such operations and simulations scale? For example, *can one implement in NEMO a computationally demanding cognitive function, such as the parsing of sentences, and will the resulting dynamical system be stable and reliable?* This is nature of the algorithms and experiments described in this chapter.

*In this chapter we present a Parser in the NEMO model.* In other words, we design and simulate a biologically realistic dynamical system involving stylized neurons, synapses, and brain areas. We start by encoding each word in the language as a different *assembly of neurons* in the brain area we call LEX. Next, we feed this dynamical system with a sequence of words (signals that excite the corresponding sequence of word-coding assemblies in LEX).

The important questions are: *Will the dynamical system parse sentences correctly? And how will we know?*

To answer the last question first, our dynamical system has a Readout procedure which, after the processing of the sequence, revisits all areas of the system and recovers a linked structure. We require that this structure be a parse of the input sentence. Our experiments show that our Parser can indeed parse correctly, in the above sense, reasonably nontrivial sentences, and in fact do so at a speed (i.e., number of cycles of neuron firings) commensurate with that of the language organ.

The Parser is an instance of NEMO made up of several brain areas, as well as fibers of synapses connecting these areas, and uses the operations of NEMO, enhanced in small ways explained in Section 3. It is in principle possible to use the plain NEMO itself [37], but this would complicate its operation and presentation, and entail the introduction of more brain areas in order to model fiber inhibition. This point is revisited and explained in this chapter. Another important point is that this NEMO device assumes that powerful word representations for each word of the lexicon have already been learnt in the form of assemblies with LRIs, and implements essentially a lexicalized parser, producing a dependency graph.

While the experiments in this chapter entail the parsing of rather simple sentences in English, and in particular can not handle dependent (embedded) clauses, in Chapter 5 we present an im-

proved parsing algorithm that handles exactly this, and leads to surprising consequences in the theory of formal languages. In the Extensions section of this chapter, we also argue that our Parser can potentially be extended in various other diverse directions (other than dependency and embedding), such as error detection and recovery, polysemy and ambiguity, and languages beyond English. We also build a toy Russian parser, as well as a universal device that takes as its input a description of the language in the form of word representations, syntactic actions and connected brain areas.

#### 4.1.1 Goals

This research seeks to explore the two questions already highlighted above:

1. Can a reasonably complex linguistic phenomenon, such as the parsing of sentences, be implemented through simulated neurons and synapses?
2. Can a computationally demanding cognitive function be implemented by the NEMO, and will the resulting dynamical system be stable and reliable? What does this tell us about the computational power of the NEMO model

The second objective is perhaps the more important and rigorous one. It is difficult to claim that our implementation of the Parser necessarily resembles the way in which the brain actually implements language, and in its current state, it cannot predict experimental data. We first and foremost see the results of this chapter as an existence proof of a nontrivial linguistic device built entirely out of simulated neuron dynamics, which tells us something about the computational power of the rather simple set of neurally-inspired assumptions that define NEMO. Of course, the Parser and its algorithms *are* too a hypothesis in its infant stages; as both the Parser is scaled and improved, and a deeper understanding relating NEMO to experimental predictions is developed, we believe that concrete, testable hypothesis will emerge from this line of work (and some hypotheses implied by the model even at its current stage are discussed at the end of this chapter).

#### 4.1.2 Background

##### **Computational psycholinguistics.**

There is a rich line of work in computational psycholinguistics on cognitively plausible models of human language processing. Such work focuses chiefly on (a) understanding whether high-level parsing methods can be used to predict psycholinguistic data (such as reading-time or eye-tracking data) and neural data (eg. fMRI and ECoG data from linguistic experiments); and (b) developing parsing methods that have specific, hallmark, experimentally-established cognitive properties of human syntactic processing (most importantly *incrementality* of parsing, *limited memory* constraints, and *connectedness* of the syntactic structures maintained by the parser). See [54] for a summary of the psycholinguistic desiderata of (b), as well as a discussion of evaluation standards for (a). Exemplars of this line of work are Jurafsky’s use of probabilistic parsing to predict reading difficulty [55], the surprisal-based models of Hale, Levy, Demberg and Keller [56, 57, 58], and the strictly incremental predictive models of Demberg and Keller [59, 60]. Much work attempts to achieve the properties in (b) above while maintaining (a), that is, neural and psycholinguistic predictiveness, e.g. the PLTAG parser of [61], or the parser of [62], which is constructed in ACT-R, a high-level meta-model of cognitive processing.

A related line of work takes modern neural (ANN) parsing methods, and examines whether the internal states of these models at parse time can be used to model psycholinguistic and neurological data observed for the same sentences — see [63] which uses an neural action-based parser, [64] and [65] which examine various RNN architectures and transformers, and [66] which compares a wide range of ANN methods. A related direction is that of [67] who build a *reservoir* network (a recurrent network that builds a representation of a sentence using fixed, random sparse matrices for weights) and build a classifier on top that predicts grammatical roles and that has also has some psycholinguistic predictivity (this has some similarities to aspects in NEMO: namely sparse, random connections).

The present paper differs from these works in key ways. In all previous work, parsers are

written in a high-level programming language, whereas we focus on whether a simple parser can be implemented by millions of individual neurons and synapses through the simulation of a realistic mathematical model of neuronal processing. In this framework, it is nontrivial to implement a single elementary step of syntactic processing, such as recording the fact that a particular input word is the sentence's subject, whereas in previous work such actions are built into the framework's primitives. In a survey paper of cognitive models of language that use ANNs, Frank summarizes that *"In spite of the superficial similarities between artificial and biological neural networks ... these cognitive models are not usually claimed to simulate processing at the level of biological neurons. Rather ... neural network models form a description at Marr's algorithmic level"* [68]. Our work can be seen as largely orthogonal to the related work described above, as we attempt to bridge granular neuronal mechanics with the study of complex cognitive processes such as syntax. See Section 4.6 for discussions of potential future work that connects the two areas.

### **Language in the brain.**

The NEMO model makes use of abstracted *brain areas* (defined in Section 3); therefore the design of our parser starts with the identification of a set of such areas. Here we discuss how our choices relate to what is known about language in the brain — a field in which much is yet unknown, or debated.

It has been known for 150 years that Wernicke's area in the superior temporal gyrus (STG) and Broca's area in the frontal lobe are involved in language; it is now generally — but not universally, see the discussion below — accepted that Broca's area is implicated in the processing of syntactic structures, while Wernicke's area is involved in word use. Language processing appears to start with access to a *lexicon*, a look-up table of word representations thought to reside in the left medial temporal lobe (MTL) opposite Wernicke's area. Major anatomical and cytological differences are known between humans and chimps at and near those areas of the left hemisphere, with evolutionary novel powerful fibers of synapses connecting these areas in humans [69]. Based on this, we believe the inclusion of a lexicon brain area (LEX), containing assemblies representing words, is

largely uncontroversial, as are its strong synaptic connections into other brain areas used by the parser.

The book [70] provides an excellent overview of a major direction in the theory of the language organ. After word look-up, activity in the STG is thought to signify the identification of syntactic roles; for example, it is known that the same noun is represented at different points in STG when it is the subject vs. the object of a sentence [71], suggesting that there are specialized areas representing subject, object, presumably verb, and perhaps other syntactic categories. However, there is active discussion in the literature on whether brain areas are dedicated to specific linguistic functions such as syntactic and semantic processing, see for example [72, 73, 74]. In our parser, we do make use of areas for different syntactic roles, but in doing so, we are not taking sides in the debate over the syntactic specialization of brain areas; we are not claiming that syntactic analysis is the exclusive function of these areas — even the LEX area containing representations of words could be part of a larger memory area in the medial temporal lobe partaking in several aspects of language.

At the highest level, the parser is generating a hierarchical dependency-based structure of a sentence that is processed incrementally. In the brain, creation of phrases or sentences seems to activate Broca’s area — what in [75] is called “merge in the brain.” Long sequences of rhythmic sentences each consisting of 4 monosyllabic words of the same syntactic structure as in “bad cats eat fish,” dictated at 4 Hz, result in brain signals from the subject’s brain with Fourier peaks at 1, 2, and 4 Hz, suggesting that the brain is indeed forming hierarchical structures [76]. Our parser represents a plausible hypothesis for the mechanism behind this process, in that it implements it within a realistic model of neural computation.

## **4.2 The NEMO model: Review and Additional Assumptions**

In this section we briefly review the NEMO model and discuss some modifications used in implementing the Parser; importantly, these modifications can be implemented fully in the NEMO model of Chapter 2 with some small additional overhead.

Recall that in NEMO, events happen in discrete time steps (think of each step as 20 ms). The *state* of the dynamical system at each time step  $t$  consists of (a) for each neuron  $i$  a bit  $f_i^t \in \{0, 1\}$  denoting whether or not  $i$  *fires* at time  $t$ , and (b) the synaptic weights  $w_{ij}^t$  of all synapses in  $E$ . Given this state at time  $t$ , the state at time  $t + 1$  is computed as follows:

1. For each neuron  $i$  compute its *synaptic input*  $SI_i^t = \sum_{(j,i) \in E, f_j^t=1} w_{ji}^t$ , that is, the sum total of all weights from pre-synaptic neurons that fired at time  $t$ .
2. For each neuron  $f_i^{t+1} = 1$  — that is,  $i$  fires at time  $t + 1$  — if  $i$  is among the  $k$  neurons *in its area* with the highest  $SI_i^t$  (breaking any ties arbitrarily).
3. For each synapse  $(i, j) \in E$ ,  
 $w_{ij}^{t+1} = w_{ij}^t (1 + f_i^t f_j^{t+1} \beta)$ ; that is, a synaptic weight increases by a factor of  $1 + \beta$  if and only if the post-synaptic neuron fires at time  $t + 1$  and the pre-synaptic neuron had fired at time  $t$ .

We call the set of  $k$  neurons in an area firing at time  $t$  the *cap* of that area.

These are the equations of the dynamical system of basic-NEMO. NEMO also has LRIs— neurons whose firing can inhibit entire areas or other LRIs. In contrast to the assembly creation and manipulation operations of Chapter 3, LRIs will be a crucial ingredient to constructing a Parser, although they will be used in a limited way and only as part of the *input* to the device. We will use the shorthand terminology of saying an area is *inhibited* (that is, the neurons in it prevented from firing because of some LRI(s)), or *disinhibited* (if it is currently in effect because no LRIs fire into it, or all such LRIs are inhibited).

In this chapter, *we will also assume that fibers can be inhibited* (be prevented from carrying synaptic input to other areas); that is, an LRI can have as its target a *fiber*. This additional assumption can be easily implemented in NEMO: given a NEMO instance  $\mathcal{B}$  with fiber inhibition, there exists an “equivalent” instance  $\mathcal{B}'$  which has an additional area  $A_f$  for each fiber  $f$  of  $\mathcal{B}$ ; for every LRI of  $\mathcal{B}$  whose target is a fiber  $f$ , its target instead set to be the area  $A_f$ . There are some small additional but trivial details to this manoeuvre (in particular the system will need to fire for

several more time steps to relay input through the new area), and doing so results in an increase in an increase in the number of brain areas.

Now, there may be multiple populations that inhibit an area or a fiber. We denote this command by  $\text{inhibit}(A, i)$ , which inhibits  $A$  through the excitation of a population named  $i$ . Similarly, disinhibition *inhibits* a population of inhibitory neurons (such as  $i$  above), which currently inhibit  $A$ , an operation denoted  $\text{disinhibit}(A, i)$ . If, for instance, we  $\text{inhibit}(A, i)$  and also  $\text{inhibit}(A, j)$ , and then  $\text{disinhibit}(A, j)$ ,  $A$  is still inhibited (because of  $i$ ). Finally, we assume that a *fiber*  $(A, B)$  can be similarly inhibited or disinhibited, denoted  $\text{inhibit}((A, B), i)$  and  $\text{disinhibit}((A, B), i)$ .

We can now define the full *state* of this dynamical system at time  $t$ . The state contains the firing state of each neuron, edge weights  $w_{ij}$ , and inhibition information. If an area  $A$  is inhibited at time  $t$ , and disinhibited at time  $t'$ , we assume that for all  $i \in A, j \in (t + 1) \dots t', f_i^j = f_i^t$ . That is, the firing state is maintained during the entire period of inhibition.

We recall that a critical emergent property of the system is that of *assemblies*. An *assembly* is a special set of  $k$  neurons, all in the same area, that are *densely interconnected* — that is, these  $k$  neurons have far more synapses between them than random, and these synapses have very high weights — and are known to represent in the brain objects, words, ideas, etc.

Suppose that at time 0, when nothing else fires, we execute  $\text{fire}(x)$  for a fixed subset of  $k$  neurons  $x$  in area  $A$  (often these  $k$  neurons will correspond to an assembly), and suppose that there is an adjacent area  $B$  (connected to  $A$  through a fiber) where no neurons currently fire. Since assembly  $x$  in area  $A$  fires at times 0, 1, 2,  $\dots$  (and ignoring all other areas), it will effect at times 1, 2,  $\dots$  the firing of an evolving set of  $k$  neurons in  $B$  — a sequence of caps —, call these sets  $y^1, y^2, \dots$ . At time 1,  $y^1$  will be simply the  $k$  neurons in  $B$  that happened to receive the largest synaptic input from  $x$ . At time 2,  $y^2$  will be the set of neurons in  $B$  that receive the highest synaptic input from  $x$  and  $y^1$  combined — and recall that the synaptic weights from  $x$  to  $y^1$  have increased. If this continues, with with high probability (where the probability space is the random connectivity of the system), the sequence  $\{y^t\}$  eventually converges to a stable assembly  $y$  in  $B$ , called *the projection of  $x$  in  $B$* . There are more high-level operations in *AC* (reciprocal projection,

merge, association, etc.), comprising a computational framework capable of carrying out arbitrary space-bounded computations. Here we shall only focus on projection, albeit *enhanced* as described below.

Suppose an assembly  $x$  in area  $A$  is projected as above to area  $B$  to form a new assembly  $y$ , a process during which neurons in  $B$  fire back into  $A$ . It is quite intuitive that, after projection,  $y$  is densely interconnected and it has dense synaptic connections from the neurons in  $x$ , because of the way in which it was constructed. Consequently, if  $x$  ever fires again,  $y$  will follow suit.

We next define a piece of syntactic sugar, that is, a shorthand for referring to a sequence of projection operations, which we call *strong projection* (see Algorithm 1). Consider all disinhibited areas of the dynamical system, and all disinhibited fibers containing them. This defines an undirected graph (which in our usage will always be a tree). Call a disinhibited area *active* if it contains an assembly — the one most recently activated in it. Now, suppose that all these assemblies fire simultaneously, into every other disinhibited adjacent area through every disinhibited fiber, and these areas fire in turn, possibly creating new assemblies and firing further down the tree, until the process stabilizes (that is, the same neurons keep firing from one step to the next). We denote this system-wide operation *strong project* or *project\**. Note that *project\** simply abbreviates a sequence of projections (which can be done in the NEMO model); we recall that the notion of fiber inhibition, which we introduce to simplify the parser implementation, could be removed at the expense of more NEMO brain areas and perhaps time steps.

Our experiments with the Parser show that indeed the operation *project\** works as described — that is, the process always stabilizes.

In the pseudocode above, the “active assembly  $x$  in  $A$ ” means either the set of  $k$  neurons most recently fired in  $A$  (which, by the dynamics of the system, happens to be an assembly), or a set of  $k$  neurons that is activated (set to fire at  $t = 1$ ) externally: we use this for activating the assembly corresponding to a fixed word in Section 4.3.

**Algorithm 1: AC code for project\***

```
1 foreach area  $A$  do
2   if there is active assembly  $x$  in  $A$  then
3     foreach  $i \in x$  do
4        $f_i^1 = 1$  ;
5     end
6   end
7 end
8 for  $i = 1, \dots, 20$  do
9   all  $A$ , all  $i \in A$ , initialize  $SI_i^t = 0$  ;
10  foreach uninhibited areas  $A, B$  do
11    if fiber ( $A, B$ ) inhibited then
12      skip ;
13    end
14     $x = \{i \in A : f_i^t = 1\}$  ;
15    foreach  $j \in B$  do
16       $SI_j^t += \sum_{i \in x, (i,j) \in E} w_{ij}$  ;
17    end
18  end
19  foreach uninhibited area  $A$  do
20    foreach  $i \in A$  do
21      if  $SI_i^t$  in top- $k_{j \in A}(SI_j^t)$  then
22         $f_i^{t+1} = 1$  ;
23      else
24         $f_i^{t+1} = 0$  ;
25      end
26    end
27  end
28  foreach uninhibited areas  $A, B$  do
29    if fiber ( $A, B$ ) inhibited then
30      skip ;
31    end
32    foreach  $(i, j) \in (A \times B) \cap E$  do
33      if  $f_i^t = 1$  and  $f_j^{t+1} = 1$  then
34         $w_{ij} = w_{ij} \times (1 + \beta)$ 
35      end
36    end
37  end
38 end
```

## 4.3 The Parser

### 4.3.1 Parser Architecture

The Parser is a program in NEMO whose data structure is an undirected graph  $G = (\mathcal{A}, \mathcal{F})$ .  $\mathcal{A}$  is a set of brain areas and  $\mathcal{F}$  is a set of fibers, that is, unordered pairs of areas. One important area is LEX, for *lexicon*, containing word representations. LEX, which in the brain is believed to reside in the left MTL, is connected through fibers with all other areas. The remaining areas of  $\mathcal{A}$  perhaps correspond to subareas of Wernicke’s area in the left STG, to which words are projected from left MTL for syntactic role assignment. In our experiments and narrative below these areas include VERB, SUBJ, OBJ, DET, ADJ, ADV, PREP, and PREPP. Besides LEX, several of these areas are also connected with each other via fibers (see Figure 4.2). Each of these areas was postulated because it seemed necessary for the Parser to work correctly on certain kinds of simple sentences. As it turns out, they correspond, roughly yet unmistakably, to *dependency labels*.  $\mathcal{A}$  can be extended with more areas, such as CONJ and CLAUSE; see Section 4.6 for a discussion of some of these extensions.

With the exception of LEX, all of these areas are standard NEMO brain areas, containing  $n$  randomly connected neurons of which at most  $k$  fire at any point in time. In contrast, LEX contains a fixed assembly  $x_w$  for every word in the language. The  $x_w$  assemblies are special, in that the firing of  $x_w$  entails, besides the ordinary effects of firing assemblies have on the system, the execution of a short program  $\alpha_w$  specific to word  $w$ , called the *action* of  $w$ . Intuitively, the sum total of all actions in assemblies of LEX constitute something akin to the device’s *grammar*. The action of each word is implemented by an LRIs for each rule of the action.

The action  $\alpha_w$  of a word  $w$  is two sets<sup>1</sup> of INHIBIT and DISINHIBIT commands, for specific areas or fibers. The first set is executed before a *project\** operation of the system, and the second afterwards.<sup>2</sup>

---

<sup>1</sup>We do not call them sequences, because we think of the commands in each as executed in parallel.

<sup>2</sup>To simplify notation, we *enumerate* the inhibitory populations  $i$  separately for each area and fiber, i.e. each area and fiber can be inhibited by inhibitory populations  $\{0, 1, \dots\}$ ; in most cases there is only one, rarely two.

$$\alpha_{\text{saw}} = \left\{ \begin{array}{l} \textit{Pre-commands} = \\ \text{disinhibit}((\text{LEX}, \text{VERB}), 0) \\ \text{disinhibit}((\text{VERB}, \text{SUBJ}), 0) \end{array} \begin{array}{l} \vdots \\ \vdots \\ \vdots \end{array} \left. \begin{array}{l} \textit{Post-commands} = \\ \text{inhibit}(\text{SUBJ}, 0) \\ \text{disinhibit}(\text{OBJ}, 0) \\ \text{inhibit}((\text{LEX}, \text{VERB}), 0) \end{array} \right\}$$

Figure 4.1: An example of an *action*, in this case for the transitive verb *saw*

The action for the word *chase* is shown in Figure 4.1 (More examples of actions and their commands for other parts of speech are given in figure 4.2). In fact, every standard transitive verb has the same action. The pre-commands disinhibit the fibers from LEX and SUBJ to VERB, allowing an assembly to be formed in VERB that is the merge of the word assembly  $x_{\text{chase}}$  in LEX and the assembly representing the subject noun phrase in SUBJ, which because subjects precede verbs in (our subset of) English, must have hitherto been constructed. Since *chase* is a transitive verb, the post-commands include the disinhibition of OBJ in anticipation of an obligatory object. In terms of neurons and synapses, we think that the  $k$  firing neurons comprising the assembly  $x_w$  contain certain neuron subpopulations whose firing excites the appropriate inhibitory and disinhibitory neural populations (the ones in the post commands through a delay operator). These subpopulations may be shared between all transitive verbs.

### 4.3.2 Operation

As shown in Algorithm 2, the Parser processes each word in a sentence sequentially. For each word, we activate its assembly in LEX, applying the pre-commands of its lexical item. Then, we *project\**, projecting between disinhibited areas along disinhibited fibers. Afterwards, any post-commands are applied.

In the pseudocode above,  $\square \in \mathcal{A} \cup \mathcal{F}$  can represent either an area or a fiber, depending on the command.

**Algorithm 2:** Parser, main loop

```

input : a sentence  $s$ 
output: representation of dependency parse of  $s$ , rooted in VERB
1 disinhibit(LEX, 0) ;
2 disinhibit(SUBJ, 0) ;
3 disinhibit(VERB, 0) ;
4 foreach word  $w$  in  $s$  do
5   | activate assembly  $x_w$  in LEX ;
6   | foreach pre-rule (Dis)inhibit( $\square, i$ ) in  $\alpha_w \rightarrow$  Pre-Commands do
7     | (Dis)inhibit( $\square, i$ ) ;
8   | end
9 end
10 project* ;
11 foreach post-rule (Dis)inhibit( $\square, i$ ) in  $\alpha_w \rightarrow$  Post-Commands do
12   | (Dis)inhibit( $\square, i$ )
13 end

```

## 4.3.3 Readout

After the completion of Algorithm 2, we will argue that a dependency tree of the sentence is represented implicitly in the synaptic connectivities  $w_{ij}$  between and within brain areas of  $G$ . To verify this, we have a *readout* algorithm which, given the state of  $G$  (in fact, only the  $w_{ij}$  and the  $k$  neurons last fired in VERB are needed), outputs a list of dependencies. Our experiments verify that when applied to the state of  $G$  after parsing a sentence with Algorithm 1, we get back the full set of dependencies.

For notational convenience, we define an operation *try-project*( $x, B$ ), in which an assembly  $x$  in some area  $A$  projects into area  $B$ , but only if this results in a stable assembly in  $B$ . This is tantamount to projecting only if  $x$  was projected into  $B$  during the Parser's execution (as part of a *project\** in some step). Lastly, define *getWord*() to be the function which, at any given time when an assembly  $x_w$  is active in LEX, returns the corresponding word  $w$ . In the following pseudocode, *try-project*( $x, B$ ) returns an assembly  $y$  in  $B$  if it succeeds, and NONE otherwise.

We present the readout algorithm mostly as a means of proving the Parser works, since it demonstrates a mapping between the state of the Parser graph  $G$  (namely, its synaptic weights)

**Algorithm 3:** Read out of parse tree in  $G$ 

```

input :  $G$  after parsing, with active root assembly  $v$  in VERB
output: the parse tree stored implicitly in  $G$ 

1 initialize stack  $s$  as  $\{(v, Verb)\}$ ;
2 initialize dependencies  $\mathcal{D} = \{\}$ ;
3 while  $s$  not empty do
4    $(x, A) = s.pop()$ ;
5    $project(x, LEX)$ ;
6    $w_A = getWord()$ ;
7   foreach area  $B \neq A$  s.t.  $(A, B) \in \mathcal{F}$  do
8      $y = try-project(x, B)$ ;
9     if  $y$  not None then
10       $project(y, LEX)$ ;
11       $w_B = getWord()$ ;
12      add dependency  $w_A \xrightarrow{B} w_B$  to  $\mathcal{D}$ ;
13       $s.insert((y, B))$ ;
14    end
15  end
16 end
17 return  $\mathcal{D}$ ;

```

and a list of dependencies. However, we remark that it is not biologically implausible. *try-project* is not a new primitive, and can be implemented in terms of neurons, synapses, and firings. Most simply,  $x$  can fire into  $B$ , and if the resulting  $k$ -cap in  $B$  is not stable (changes with a repeated firing, say), it is not an assembly. Alternatively (and in simulation), one can project from  $B$  into LEX, and if the cap in  $B$  is not a stable assembly formed during parsing, the  $k$ -cap in LEX will *also* not be an assembly, i.e. not correspond to any valid  $x_w$  (this is related to “nonsense assemblies” discussed in Section 4.6). Further, the stack of Algorithm 3 is only used for expository clarity; as the assemblies projected are always the ones projected to in the previous round, with some thought one could implement the algorithm with *project\**.

As an example, figure 4.2 shows the trace of the Parser throughout the sentence “*the man saw a woman*”. The action  $\alpha_i$  of each word is given beneath a figure showing the state of the Parser after the Pre-commands and *project\** have occurred for that word. Green areas and fiber arrows are disinhibited; red are inhibited. All assemblies active right after *project\** are shown (with the

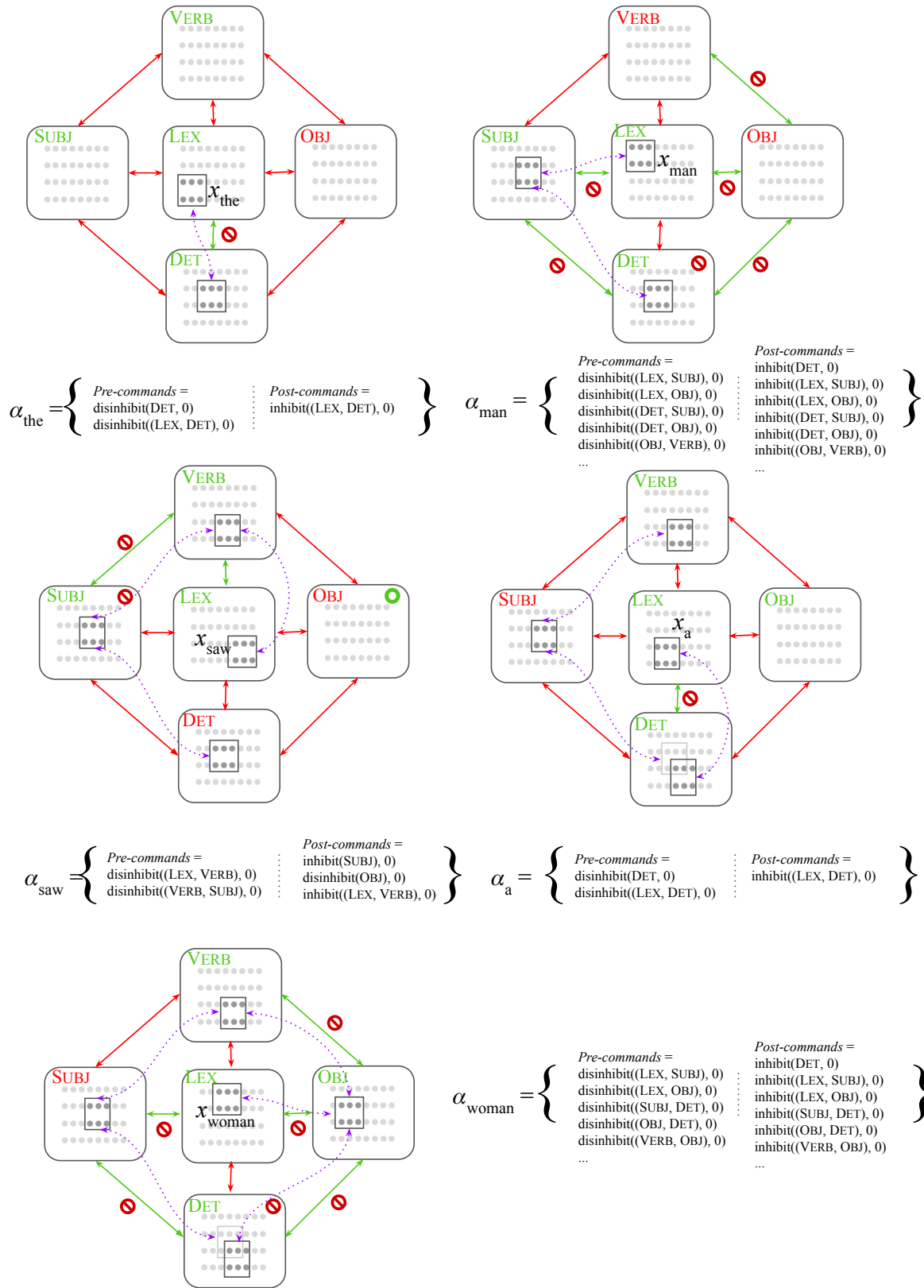


Figure 4.2: Execution of the Parser on an example sentence

one in LEX labeled  $x_w$ ). Purple dotted arrows indicate assemblies that have been connected; one can fire to activate the other. Purple arrows to LEX are shown only in the stage in which they are created. Green (○) and red (⊗) circles indicate areas or fibers that will be disinhibited/inhibited by the Post-commands.

## 4.4 Experiments

We provide an implementation of the Parser in Python. Importantly, the entire algorithm, at its lowest level, is running by simulating *individual neurons and synapses*, and any higher-level concept or structure (such as the parse tree itself) is an abstraction that is encoded entirely in the individual firings and neuron-to-neuron synaptic weights of a giant graph and dynamical system representing the brain.

We provide a set of 200 varied sentences that are parsed correctly by the Parser. For each sentence we have a target correct dependency structure; we verify that the Readout operation produces this dependency structure for each sentence.

The sentences were designed by hand in order to demonstrate a variety of syntactic phenomena that our Parser (and NEMO) can handle; this includes prepositional phrases, transitivity and intransitivity (including verbs that can be both transitive or intransitive), optional adverb placement pre- or post-verb, and several more. See Section 4.5 for these example sentences and the syntactic pattern they represent. We remind the reader that to our knowledge, this is the first realistic parser implemented on the level of individual neurons and synapses.

The 200 sentences parsed were sampled from 10 syntactic patterns the Parser was designed to handle, from a vocabulary of 100 words. Since the underlying dynamical system can in principle develop an instability with low probability, failure is still possible, but it did not happen in our experiments. There are syntactic structures that the Parser does not currently handle, but which we believe are possible in our model in two senses: first, the grammar (word actions) can be expanded to handle these structures, and more importantly, the dynamical system will still correctly parse them with high probability. A prime example of such a structure is sentence embedding (“the man

said that..."). See Section 4.6 for more discussion of such extensions.

The speed of parsing in our simulations is commensurate with known bounds on neural language processing. In each step, the participating assemblies must fire enough times for the project\* operation to stabilize; we find through simulation that for reasonable settings of the model's hyperparameters, convergence occurs within 10-20 firing epochs, even when multiple brain areas are involved. Assuming 50-60 neuron spikes (NEMO time steps) per second, and allowing for the (parallel) execution of the inhibit/disinhibit operations in the word's action, which will take a small number of additional cycles, we arrive at a frequency range of 0.2-0.5 seconds/word.

#### **4.5 Details of the experiment**

**English** We generated 10 examples each from the 20 templates shown below, a total of 200 sentences. Our Parser simulator correctly parsed all 200 examples, in the sense that a correct dependency graph was generated by the readout. Each of the 20 templates is a part-of-speech sequence. For each template we show below an example sentence, and in the code files we provide the correct dependencies for each to compare to the Parser's output. In the templates below, V = V-TRANS.

1. N V-INTRANS (people died)
2. N V N (dogs chase cats)
3. D N V-INTRANS (the boy cried)
4. D N V N or N V D N (the kids love toys)
5. D N V D N (the man saw the woman)
6. ADJ N V N or N V ADJ N (cats hate loud noises)
7. D ADJ N D ADJ N (the rich man bought a fancy car)
8. PRO V PRO (I love you)

9. {D} N V-INTRANS ADVERB (fish swim quickly)
10. {D} N ADVERB V-INTRANS (the cat gently meowed)
11. {D} N V-INTRANS ADVERB (green ideas sleep furiously)
12. {D} N ADVERB V {D} N (the cat voraciously ate the food)
13. {D} N V-INTRANS PP (the boy went to school)
14. {D} N V-INTRANS PP PP (he went to school with the backpack)
15. {D} N V {D} N PP (cats love the taste of tuna)
16. {D} N PP V N (the couple in the house saw the thief)
17. {D} N COPULA {D} N (geese are birds)
18. {D} N COPULA ADJ (geese are loud)
19. *complex sentences with copula* (big houses are expensive)
20. *chained adjectives, extended model* (the big bad problem is scary)

**Russian** To demonstrate parsing of a syntactically very different language, we consider Russian sentences with subject, object, and indirect-object, like “женщина дала мужчине сумку” (*woman-nom give-past man-dat bag-acc*, “the woman gave the man a bag”). All  $4! = 24$  permutations of this sentence are valid, e.g. “сумку женщина мужчине дала”. For each of them, the Parser produces the same dependencies  $\mathcal{D} = \{ \text{дала} \xrightarrow{\text{НОМ}} \text{женщина}, \text{дала} \xrightarrow{\text{ДАТ}} \text{мужчине}, \text{дала} \xrightarrow{\text{АСС}} \text{сумку} \}$ .

## 4.6 Extensions and Discussion

**The UParser and a parser for Russian.** The underlying meta-algorithm of the Parser, which we call the UParser, takes as input a set of areas  $\mathcal{A}$ , a set of fibers  $\mathcal{F}$ , a set of words  $W$  and a set of actions  $a_w$  for each  $w \in W$  (whose commands manipulate only areas in  $\mathcal{A}$ ), and parses with NEMO

using Algorithms 2 of Section 3. The UParser is language-agnostic, and can be seen as modeling a universal neural basis for language: a specific language instantiates this basis by specifying  $\mathcal{A}$  (roughly, its syntactic constituents),  $\mathcal{F}$ ,  $W$  and the  $\alpha_i$ . This is in no way constrained to English; for instance, our model and algorithm are equally well equipped to handle a highly inflectional language with relatively free word order, where syntactic function is indicated morphologically. We demonstrate this with a parser for a toy subset of Russian, a language with free word order in simple sentences. Particularly, our parser works on a set of sentences that is closed under permutation; permutations of the same sentence are parsed correctly, and produce the same dependency tree (as verified with Algorithm 3). See section 4.5 for more details on the Russian Parser.

**The big bad problem and recursion.** The Parser of Section 3 is rather skeletal and needs to be extended to handle recursive constructions such as compounding and embedded clauses. While how to handle embedding is the theme of Chapter 5, compounding is a much simpler problem. Consider the sentence “The big bad problem is scary”. Following the Parser of Section 3, after the word “big”, an assembly  $a$  representing “big” has been created in ADJ which the system anticipates projecting into SUBJ in a subsequent step (when a noun is encountered) to form an assembly in SUBJ representing the subject NP. However, the next word is not a noun but “bad”, part of a chain of adjectives. Now, if we project from LEX into ADJ to form an assembly  $b$  representing “bad”, we lose all reference to  $a$ , and there is no way to recover “big”! There are several simple and plausible solutions to the chaining problem, which can also be used to parse adverb chains, compound nouns, and other compound structures.

One solution is using two areas  $\text{COMP}_1$  and  $\text{COMP}_2$  in  $\mathcal{A}$ . When we encounter the second adjective (or more generally a second word of the same part of speech which otherwise would fire into the same area, overwriting earlier assemblies), we project from LEX into  $\text{COMP}_1$  instead of ADJ, but simultaneously project from ADJ into  $\text{COMP}_1$  to link the first adjective with the second. For a third adjective in the chain, we project LEX into  $\text{COMP}_2$  but also  $\text{COMP}_1$  into  $\text{COMP}_2$  *unidirectionally*; generally, for adjective  $i$  in addition to LEX we project unidirectionally from  $\text{COMP}_{\text{parity}(i)}$

(which contains the previous adjective assembly) into  $\text{COMP}_{\text{parity}(i-1)}$ . We demonstrate parsing chains of varying lengths with these two areas in our simulations. However, one limitation of this solution is that it requires *unidirectional* fibers; if the projection from  $\text{COMP}_1$  to  $\text{COMP}_2$  above is reciprocal, it won't be possible to link the assembly in  $\text{COMP}_2$  to *another* assembly in  $\text{COMP}_1$ .

Another approach, which may be more realistic and obviates the need for unidirectional fibers (though it cannot handle arbitrary-length chains) is to add more than 2 areas,  $\text{COMP}_1, \dots, \text{COMP}_m$ , for some small but reasonable  $m$ , perhaps 7. Parsing proceeds as in the two area solution but by chaining  $\text{COMP}_i$  to  $\text{COMP}_{i+1}$ . The number of areas  $m$  could model well-studied cognitive limits in processing such complex structures, see [77, 78]. To model longer chains, in high-demand situations or with practice, the brain could recruit additional brain areas.

Such maneuvers can handle right recursion of the form  $S \rightarrow A^*!A^+$ . Center embedded recursion is more complicated. To construct a parse tree for a sentence with an embedded sentence or relative clause, for an execution of the Parser's inner loop, the subjects and/or the verbs of the two sentences may need to coexist in the corresponding areas. This is not a problem, as brain areas can handle tens of thousands of assemblies, but the linking structure must be preserved. It must be possible to save the state of the embedding sentence while the embedded one is serviced. One possibility is to introduce `CURRENTVERB` areas, which can be chained like the compounding areas above, and will act as a stack of activation records for this more demanding form of recursion. The idea is that, by recovering the verb of the embedding sentence, we can also recover, through linking and after the end of the embedded sentence, the full state of the embedding sentence at the time the embedded sentence had begun, and continue parsing. This needs to be implemented and experimented with.

**Disambiguation.** In English, *rock* can be an intransitive verb, a transitive verb, or a noun; so far, we have ignored the difficult problems of polysemy and ambiguity. To handle polysemy, every word  $w$  may need to have many action sets  $\alpha_w^1, \alpha_w^2, \dots$ , and the parser must disambiguate between these.

We believe that the Parser can be extended to handle such ambiguity. The choice of the action could be computed by a *classifier*, taking as input a few words' worth of look-ahead and look-behind in the current sentence (or perhaps just their parts of speech), and selecting one of the action sets; the classifier can be trained on the corpus of previously seen sentences. This also needs implementation and experimentation.

**Error detection.** Grammaticality judgment is the intuitive and intrinsic ability of native speakers of any language to judge well-formedness; this includes the ability to detect syntactic errors. Neurolinguists are increasingly interested in the neural basis of this phenomenon; for instance, recent experiments have detected consistent signals when a sentence fragment is suddenly continued in a way that is illegal syntactically [79]. Built into the Parser is the ability to detect some malformed sentences.

There are at least two simple mechanisms for this. One is when a fragment is continued by a word that immediately makes it ungrammatical, such as following “the dogs lived” with “cats”. The Parser, having processed the intransitive verb “lived”, has not disinhibited area OBJ, and all other noun-related areas are inhibited. Upon encountering “cats”, *project\** will not fire any assemblies; we call this an *empty-project* error.

Other kinds of syntactic violations can be detected by *nonsense assembly* errors. Such an error occurs during readout when an area is projected into LEX, but the resulting assembly in LEX does not correspond to  $x_w$  for any  $w \in W$ ; in other words, when the function *getWord()* of the readout algorithm of Section 3 fails, which indicates that the state of  $G$  must have resulted from an impossible sentence. We provide a list of illegal sentences for which our Parser simulation detects empty-project or nonsense-assembly errors, indicating different kinds of syntactic violations.

One kind of syntactic error our Parser does not detect currently is *number agreement*: Our simulator would not complain on input “Cats chases dogs;” this is not hard to fix by having separate areas for the projection of singular and plural forms. Other types of agreement, such as in gender, can be treated similarly.

**The role of Broca’s area.** Language processing, especially the formation of phrases and sentences, has been observed to activate Broca’s area in the left frontal lobe; so far, the parser only models processes believed to be in the left STG and MTL. We hypothesize that Broca’s area may be involved in the building of a concise syntax tree summarizing the sentence, consisting of only the subject, the verb, and the object if there is one (but with access to the rest of the parsed sentence through the other areas). This would involve new areas S (for sentence) and VP (for verb phrase), with fibers from VERB and OBJ to VP, and from VP and SUBJ to S. Building the basic three-leaf syntax tree can be carried out passively and in the background while the rest of syntax processing is happening in our current model; this is explored in more detail in Chapter 5.

**Closed classes and an alternative architecture.** Words such as *the*, *of*, *my* may reside in areas dedicated to closed parts of speech in Wernicke’s area, instead of being a part of LEX as we assume, for simplicity, in our model and simulations. In fact, before exploring the architecture described here, we considered an alternative in this direction, which has been suggested in the neuroscience literature [80].

Suppose that LEX in the left MTL only contains the phonological and morphological information necessary for recognizing words, and all grammatical information, such as the actions  $\alpha_w$ , reside in Wernicke’s area, perhaps again in subareas not unlike the ones we postulate here. For example, SUBJ could contain every noun (in Russian, in its nominative form), as assemblies which are in permanent projection back and forth with the corresponding assemblies in LEX; likewise for OBJ (accusative in Russian). Verbs are all permanently projected in VERB, and so on. In this model, syntactic actions are specific to areas, not words. We plan to further explore what new problems such a parser would present — and which problems it would solve —, as well as what advantages and disadvantages, in performance, complexity, and biological plausibility, it may have.

**Predictions of our Model.** As explained in the introductory section of this chapter, the Parser is first and foremost a proof of concept, an “existence theorem” establishing that complex cognitive,

and in particular linguistic, functions can be implemented in a biologically realistic way through NEMO. However, to this end, we chose an architecture that is compatible with what is known and believed about language processing in large parts of the literature. Several predictions can be made that can be verified through EEG and fMRI experiments. One such prediction is that the processing of sentences of similar syntactic structure (for example, “dogs chase cats” and “sheep like dogs”) would result in similar brain activity signatures, while sentences with a different syntactic structure (“give them food”) would generate different signatures.

When our understanding of the NEMO model improves to a point that it can be used (heuristically) to simulate experimental productions, and/or experimental techniques improve to record from large numbers of individual neurons in humans, the Parser model can be concretely tested, as it predicts long-term assemblies corresponding to lexical items in an identifiable *LEX* area. It should also be possible to observe the formation, on-the-fly, of assemblies corresponding to syntactic structures and parts of speech formed during parsing; one could also identify the corresponding areas (such as VERB, SUBJ, etc.). When it comes to identifying areas, the precise anatomical location of these may vary per individual, but be consistent in any individual. Our ultimate prediction is that the long-sought *neural basis of language* consists of — or rather, can be usefully abstracted as — a collection of brain areas and neural fibers at the left MTL and STG and elsewhere in the brain, powered by the *project\** operation, and adapted during the critical period to an individual’s maternal tongue and other circumstances.

**Open questions** A major set of questions left open in this chapter concern extending the scope of the Parser; this includes the extensions mentioned above (embedding and recursion in particular, which is addressed in Chapter 5).

Another focus will be integrating this work with the existing directions in computational psycholinguistics. This includes enhancing the parser to exhibit the hallmark psycholinguistic desiderata described in Section ???. Our parser in fact has *incrementality* in the same sense as this literature, but it would be interesting to achieve *connectedness* of intermediate structures. Another future di-

rection would be to consider how a parser implemented in NEMO could be used to predict or model experimental data (such as processing time).

To conclude, we highlight one open problem for future work, the contextual action problem. We are given sentence  $s$  with labeled dependencies  $\mathcal{D}$ , and a Parser with an area for each label that occurs in  $\mathcal{D}$ , as well as LEX. Do there exist *contextual actions*  $\alpha_w^*$  for each word  $w \in s$  such that the parsing algorithm, combined with readout, yields  $\mathcal{D}$ ? Can we construct an oracle  $O(s, w)$  that returns the contextual actions? If not, then what set of labeled dependency trees can be recognized under this formalism? How are such actions represented neurally? Can we plausibly implement contextual actions in NEMO, based on a word and its immediate neighbors in  $s$ ?

#### 4.6.1 Conclusion

Few students of the brain think seriously about language, because (1) language is by far the most advanced achievement of any brain, and (2) despite torrential progress in experimental neuroscience, an overarching understanding of how the brain works is not emerging, and the study of language will require that. This is most unfortunate, because language has evolved rapidly over a few thousand generations, presumably to adapt to the capabilities of the human brain, and it therefore presents a great opportunity for neuroscience. This paper is a small first step towards establishing a framework for studying, computationally, linguistic phenomena from the neuroscience perspective, and we hope that it will be followed by bolder experiments and far-reaching advancements in our understanding of how our brain makes language and the mind.

## Chapter 5: Parsing embedded clauses

This chapter uses material from [81]. The material has been refurbished for the purposes of this thesis, but the bulk of the ideas and results presented represent the collaborative effort with the original authors: Adiba Ejaz, Mirah Shi, Mihalis Yannakakis and Christos Papadimitriou.

### 5.1 Introduction

Several research directions were left open by the Parser of the previous chapter. Preeminent among them were these two: (a) how can the parser be extended so that dependent clauses, and especially center-embedded ones, are parsed correctly? and (b) how could the *constituency* representation of the sentence (the tree with "Sentence" and "Verb Phrase" as its internal nodes and the "Subject," "Verb," and "Object" as leaves) be produced — there is experimental evidence that the main constituents of a sentence are indeed created in Broca's area during sentence processing (e.g. [ZF]). *Can these problems be solved in the NEMO model?*

Parsing center-embedded sentences presents a serious conceptual difficulty: the parsing of the embedding sentence must be interrupted, and then be continued after the embedded sentence is parsed. This is the nature and essence of recursion. A mechanism for recovering the state of the parser at the moment of the interruption seems thus necessary. It would be tempting to posit that recursion in the brain happens as it does in software, through the creation of a stack of activation records, but as we shall discuss, this is unlikely to be biologically plausible.

In this paper we pursue these two unresolved research goals and make significant progress on both (a) and (b) above. We start our description with constituents, which is the simpler narrative. The Parser in Chapter 4 consists of the brain area LEX where the lexicon resides (believed to be a part of the medial temporal lobe, MTL), as well as several other areas labeled SUBJ, VERB, OBJ,

DET etc. corresponding to syntactic roles, believed to be subareas of Wernicke's area; this is where the dependency parse is created. We show that this architecture can be augmented by new areas and fibers in such a way that the basic constituency tree of the sentence (the three-leaf tree that has inner nodes Sentence and Verb Phrase) can be also built "on the side" while the sentence is parsed. This entails two new brain areas denoted S for sentence and VP for verb phrase. The brain areas in our model are intended to correspond to two well known subareas of Broca's area, BA 43 and BA 44, respectively, where such activity is thought to take place [**Friederici**].

Coming now to the problem of dependent clauses, to handle clauses that are not embedded (say "if dogs are angry, they chase cats"), the Parser needs only one extra brain area beyond the areas needed by the original [49], namely an area labeled DS for "dependent segment."

To see the problem with embedded sentences, consider the variant "dogs, when they are angry, chase cats." The Parser would recognize "dogs" as the subject of the sentence and project the assembly for "dogs" to the SUBJ area and change the state as appropriate (inhibit a fiber). Upon encountering the first comma, the parsing of the sentence is curtailed, and the parsing of a new clause will be initiated. This is not a problem, since brain areas can contain many assemblies at the same time, and the two verbs, subjects, etc. will not interfere with each other. However, upon the second comma (for simplicity we assume that these clues are always available— in speech, they may, for instance, be indicated using prosodic or pausal cues), the Parser needs to continue the parsing of the outer clause (that is, the main sentence), and for this *it needs to restore the state at the moment the parsing was interrupted by the embedding sentence*. We could have the state saved in a pushdown store and retrieve it at this point, but we can see no biologically plausible implementation of either Parser state records or a pushdown store. As we shall point out, the Parser so far is a finite automaton, and thus it is not surprising that it has trouble handling embedded clauses.

The question is, *which biologically plausible departure from finite state machines, and from the model in Chapter 4, can handle embedding?* We propose the following: The Parser stores the part of the utterance already parsed in some *working memory* (as in [**neuro**], for example). When

the embedded clause has been parsed, the Parser returns to the beginning of the outer sentence in the working memory and reprocesses it (up to the comma) to restore the state. We only need to execute the action sets of the words in the first part of the sentence, an activity that we call *touching*, which is, in terms of elapsed time, an order of magnitude faster than parsing the first time. Once the first comma is seen, the embedded clause is skipped (without state changes) until the second comma, and parsing of the outer sentence is resumed from the recovered state. Any embedded clause can be handled with the touching maneuver — non-embedded dependent clauses are much simpler.

We show by processing several examples that several forms of dependent and embedded clauses can be correctly parsed (that is, a correct dependency graph of the whole sentence can be retrieved from the device after processing). If the embedded clause has its own embedded clause, the trick can be repeated.

Very surprisingly to us, this extension of the Parser, arrived at strictly through considerations of biological plausibility and ease of implementation, yields a rather unexpected theorem in formal language theory: if one defines an extension of nondeterministic finite-state automata to capture the operation of the enhanced Parser, with the extra capabilities of (a) marking the current input symbol and (b) reverting from the current input symbol to the previously marked symbol that is closest to the current one, then the class of languages accepted by these devices — call them *fallback automata* — coincides with the context-free languages!

### 5.1.1 The original Parser

The recall that the basic architecture of the Parser in of the previous chapter consists of several brain areas connected by fibers. LEX special, and contains representations of all words. Upon input of a new word (in brain reality, read or heard) the corresponding representation is excited, and upon firing it executes the *action set* of the word, commands which capture the grammatical role of the word. These commands open and close (disinhibit and inhibit) certain fibers. Then the project\* operation is executed: all active assemblies fire. By this scheme, it was shown in [49]

that many categories of English sentences (and Russian as well) can be parsed correctly (the correctness can be verified because the running of the parser on an input sentence leaves a retrievable graph structure within and between brain areas, which constitute a correct dependency graph of the sentence). This completes the description of the Parser’s architecture. We provide a figure with an example sentence parsed, and refer the interested reader to [49].

### 5.1.2 Neuroscience

We believe that the improved Parser represents a reasonable hypothesis for parsing in the brain. The neurobiological underpinnings of NEMO, with which our model is built, and the original Parser of [49], which ours extends, are presented in Chapter 4. Briefly, NEMO is based on established tenets of neuron biology, including that neurons fire when they receive sufficient excitatory input from other neurons, the atomic nature of neuron firing, and a simplified narrative of synaptic Hebbian plasticity (see for instance [NeuralScience], Chapters 7, 8, and 67). Assemblies, in turn, are an increasingly popular hypothesis for the main unit of higher-level cognition in modern neuroscience— first hypothesized decades ago by Hebb, they have been identified experimentally [51] displacing previously dominant theories of information encoding in the brain, see e.g. [52]. With what regards the higher-level Parser architecture, language processing appears to start with access to a *lexicon*, a look-up table of word representations thought to reside in the left medial temporal lobe (MTL), motivating the inclusion of an area LEX. After word look-up, activity in the STG is thought to signify the identification of syntactic roles. Overall, the Parser generates a hierarchical dependancy-based structure that from a sentence that is processed incrementally, which we believe models something like the creation of hierarchical structures in Broca’s areas in experiments such as [76].

## 5.2 Constituency

Constituency parsing revolves around the idea that words may lump into a single assembly. The noun subject of a sentence — along with its dependent adjective(s) and determinant(s) if any

— form the “Subject,” while the verb and object form the “Verb Phrase.” At the coarsest level, the “Subject” and “Verb Phrase” then form the “Sentence.” We modify the underlying framework of the Parser to assign this syntactic structure to a sentence.

We add two new brain areas that hold the “Verb Phrase” and “Sentence,” VP and S respectively, as well as fibers between VERB and VP, OBJ and VP, SUBJ and S, and VP and S. These fibers remain disinhibited throughout parsing so that the constituency tree is built concurrently as we parse the sentence. That is, when the verb is processed, a corresponding assembly is formed in VP, and upon encountering the object, the assemblies in VERB, OBJ, and VP fire together to form in VP what is now the merge of assemblies representing the verb and object. Parallel to this process, assemblies fire along the SUBJ to S and VP to S fibers so that the final assembly in S represents the joining of the “Subject” and “Verb Phrase.”

**Experiments.** We extend the implementation of the Parser in Python to incorporate these new abilities. Additionally, we tailor the readout algorithm (which in [49] recovers the dependency tree) to output the desired tree rooted in S. To verify that the Parser produces the correct constituency tree, we provide a test set of 40 sentences constructed from 20 syntactic patterns that include variations in word orderings; additions of determinants, adjectives, adverbs, and prepositional phrases; and both transitive and intransitive verbs. The Parser generates the correct constituency trees on all of our given test cases. Importantly, the constituency Parser can handle any sentence structure that the original Parser can handle.

As in the original parser, we execute 20 firing epochs of project\* to allow the dynamical system to stabilise. The multiple concurrent projections into S and VP cause a slowdown by a factor of 2.5 relative to the dependency parser, resulting in a frequency of 0.5-1.3 seconds/word.

### 5.3 Embedded Sentences

To handle embedded sentences, the Parser requires a new area DS (for dependent segment) to handle dependent and embedded clauses. Additionally, we need modifications that recover the

state *before* an embedded clause (which we recover when we finish parsing an embedded clause, i.e., upon a “right comma”).

In particular, we assume that there is a *working memory area* which holds the words which have already been processed, in sequence. We assume for simplicity that the parser can always recognize the beginning of a dependent sentence — that this is always possible through simple clues such as a comma (in text), prosodic cues (in speech), and/or a *complementizing* pronoun or preposition, such as “who”, “if”, or “that”. We also assume that there is a unique sentence or clause at each depth, though with minor modifications we can handle the more general case.

To handle center-embedded clauses, the parser utilizes a limited working memory: it must remember the sentence up to the point when an embedded clause begins in order to efficiently reprocess these words after parsing the embedded clause. More concretely, when a center embedding is detected, the Parser “cleans the slate”; that is, the Parser state (the fiber and area states) is reinitialized in order to parse the new embedded clause, which is parsed normally until its end is detected. At this point the Parser has to restore its last state when it was parsing the outer clause. To do so, it reinitializes the state again and reprocesses the sentence from the beginning of the outer clause up to the interruption. However, this re-processing is special: the parser only “touches” the words, by which we mean that for each word we apply its action sets (inhibiting or disinhibiting areas/fibers), fire the entire system exactly once to reactivate existing assemblies that were formed in the initial parse of the fragment (when we initially parsed this part of the sentence, we used *project\**, which fires the entire system 20+ times in order for assemblies to form and converge). In this way, touching is by an order of magnitude faster than parsing on initial input of the word — both in our simulation and in the hypothesized language organ — since it is only recovering the *preexisting* structure. Note that when the first comma is scanned, the parsing of the outer sentence is resumed from the second comma, skipping the (already parsed) embedded clause.

The remaining difficulty is in linking the outer and inner clauses. The last word of the outer clause before the inner sentence functions as a “signature” for the outer clause. This signature may reside in SUBJ or OBJ, for instance. After parsing the fragment of the outer clause pre-interruption,

we project from the relevant signature area to DS. Then, on the verb of the inner clause, we project from both LEX and DS into VERB simultaneously. This way, we can later recover the root verb of the inner clause via the signature assembly of the outer sentence (through projection into DS and subsequently VERB). to DS, and then to VERB.

**Experiments.** We extend the implementation of the dependency Parser in Python to incorporate touching, linkage, and recursion within the main Parser loop. In principle, the developed Parser can handle arbitrarily many levels of embedding. We test on 20 sentences sampled from 5 different embedding structures, with depths 0, 1, and 2 (informed by the lack of three or greater depth sentences in ordinary language). Our test cases feature center-embedding, edge-embedding, mixed embedding, and relative clauses modifying the subject or the object. We assume there is a unique clause at each depth. The Parser generates the expected dependency tree on all of our given test cases.

Again, we execute 20 firing epochs of `project*`. The modified parser preserves the speed of the original, with a negligible increase in time for linkage projections and touching. Despite several challenges created by the added complexity of sentence embedding, the linkages between projected assemblies correspond to the correct dependency graph in all sentences.

Figure 5.1 shows a snapshot of the Parser while processing the center-embedded sentence “*dogs, when they run, chase cats.*” Green arrows represent fibers that have been disinhibited between each stage; red arrows represent inhibited fibers. Purple dotted arrows indicate assemblies that have been connected through `project*`. Pink dotted arrows indicate assemblies touched through activating their corresponding assemblies in LEX. (a) Outer sentence, up until beginning of inner sentence, parsed; (b) Assemblies in the signature and DS areas linked; (c) Fiber and area states reset in anticipation of a new sentence; (d) Inner sentence parsed and linked to outer sentence through the DS area; (e) Outer sentence revisited: each word prior to dependent sentence is touched, restoring Parser state of stage (a); (f) Remainder of outer sentence parsed.

**Algorithm 4:** Enhanced Parser, main loop**input** : a sentence  $s$ , depth  $d$ **output:** representation of dependency parse of  $s$ , rooted in VERB

```
1 Function parse ( $s, d \leftarrow 0$ ) :
2   foreach word  $w$  in  $s$  do
3     if  $(d + 1)$ -depth clause begins after  $w$  then
4       |   disinhibit(DS) ;
5       |   disinhibit(DS, AREA( $w$ )) ;
6       |   project* ;
7       |   inhibit(DS, AREA( $w$ )) ;
8       |   inhibit(DS) ;
9     end
10    else if  $w$  begins  $(d+1)$  depth clause then
11      |    $d \leftarrow d + 1$  ;
12      |   clear the slate ;
13    end
14    else if  $w$  ends embedded sentence then
15      |    $d \leftarrow d - 1$  ;
16      |   foreach word  $y$  before  $w$  do
17        |   activate  $y$  in LEX ;
18        |   fire DISINHIBITED AREAS ;
19      |   end
20    end
21    if  $d > 0$ , AREA( $w$ ) = VERB then
22      |   disinhibit(DS) ;
23      |   disinhibit(DS, VERB) ;
24    end
25    execute  $w$  actions and project* ;
26    inhibit(DS) ;
27    inhibit(DS, VERB) ;
28  end
```

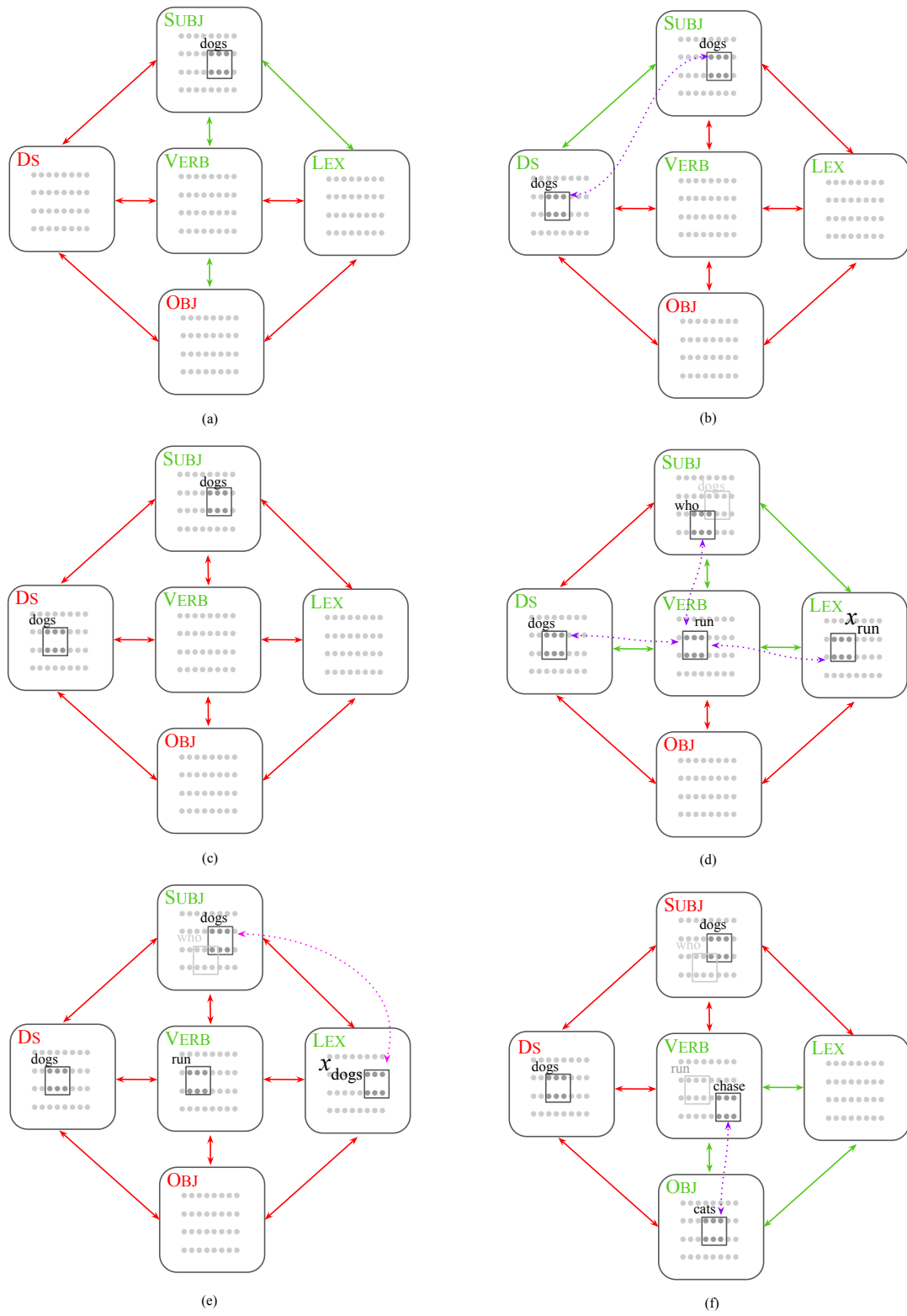


Figure 5.1: Example of the extended Parser on an embedded phrase

## 5.4 A Little Formal Language Theory

The Parser, without the embedded sentence module, is a *finite state device*. The reason is that the Parser's state is an acyclic subgraph of the graph of brain area and fibers (excluding LEX). It is no wonder then that sentence embedding (intuitively, a feature involving recursion and therefore moving us out of the realm of regular languages) requires an extension, and among many other options that seemed to us less biologically plausible we chose to revisit the outer sentence and restore the state of the Parser. It is now natural to ask: Can the operation of the Parser when handling embedded sentences be seen as a more powerful genre of automaton? This is the motivation for the results in this section.

**Definition:** A *fallback automaton (FBA)* is a tuple  $A = (\Sigma, K, I, F, \Delta)$ , where (as in a nondeterministic finite-state automaton)  $\Sigma$  is a nonempty finite set of symbols,  $K$  is a set of states,  $I$  is the set of initial states,  $F$  is the set of accepting states, and  $\Delta$  is the transition relation.

Define the *type set*  $T = \{f, s\} \cup K$ . Whereas in nondeterministic FSAs  $\Delta \subseteq (\Sigma \times K) \times K$ , the transition relation of FBAs is more complex.

$$\Delta \subseteq ((\Sigma \times T \times K) \times (K \times \{s, \checkmark, \leftarrow\}))$$

The automaton is *nondeterministic*, reflecting the nondeterministic nature of parsing in general, due to ambiguity and polysemy. The transition relation can be understood thus: symbols on the tape are marked by a type, either  $f$  (for *fresh*),  $s$  (for *seen*), or by a state  $q \in K$ . Initially, all symbols are fresh and hence marked  $f$ . When a symbol is marked  $s$  or by a state of  $K$ , it is not being scanned for the first time — the automaton may scan a symbol multiple times. The first time a symbol is scanned, its type is changed from  $f$  to  $s$ , or, if the rule outputs  $\checkmark$ , it is set to the current state  $q$ . Subsequently, after a symbol marked by a state is scanned, the type reverts to  $s$ .

To formalize the operation of the automaton, at each step, there is a *tape*  $x \in (\Sigma \times T)^*$ ; we denote its  $i$ th symbol as  $x_i = (\sigma_i, t_i)$ , where  $\sigma_i \in \Sigma$  and  $t_i \in T$ . The automaton is in a state  $q \in K$  and the  $i$ th symbol  $x_i$  is scanned. The overall configuration is thus  $(x, q, i)$ . In the initial

configuration, the type of every symbol is  $f$ ,  $q \in I$  and  $i = 1$ .

If the next configuration after  $(x, q, i)$  is  $(y, r, j)$  then:

- $y$  is identical to  $x$  except that the type of the  $i$ th symbol may have changed:  $f$  must become  $s$  or the machine's current state  $q$ ,  $s$  stays  $s$ , and  $q \in K$  always become  $s$ .
- unless this is a fallback step,  $j = i + 1$ .
- *Fallback.* When scanning a fresh symbol, the automaton may return to a position  $j < i$ , where  $j$  is the largest position  $j$  that was marked — that is,  $t_j \in K$ . Notice that in the next step,  $t_j = s$ — the symbol is unmarked.
- When scanning a symbol with type  $q \in K$ , i.e., a symbol fallen back to, the transition can map to only  $s$  or  $\leftarrow$ — that is, it can fallback again, but it cannot mark the symbol again for fallback.
- In all cases, the corresponding pair  $((x_i, q)(r, \sigma))$  must be in  $\Delta$ .  $\sigma = s$  means that the  $i$ -th symbol's type is changed to  $s$ ,  $\sigma = \checkmark$  means it is changed to the state  $q$ , and  $\sigma = \leftarrow$  means the step is a fallback.

This concludes the definition of the FBA. We say that a string  $x$  in  $\Sigma^*$  is accepted by FBA  $A$  if there is a sequence of legal steps from a configuration with state in  $I$  and tape  $x$  with all symbols fresh to a configuration in which the state is in  $F$ .

Note that when the FBA falls back to a previous tape location  $j < i$ , it then passes again over the seen symbols (marked  $s$ ) between  $x_j$  and  $x_i$ , and may do meaningful computation upon this revisiting. Furthermore, an FBA can fallback repeatedly, immediately after a fallback move. Our Parser more closely corresponds to an FBA without this abilities. Hence it is interesting to define a weaker model without this ability:

A *weak-FBA* is an FBA with the additional requirements that 1) for any symbol marked  $s$ , transitions cannot change the state (that is, for all  $\alpha \in \Sigma$ ,  $q \in K$  and  $(\alpha, s, q) \times (q', s) \in \Delta$ , we

require  $q' = q$ , in effect, skipping over all  $s$  symbols) and 2) for any symbol marked with  $q \in K$ , the transition must output  $(q', s)$  (that is, there are no repeated fallbacks).

It may seem that FBAs can do more than weak-FBAs. Consider the following language over  $\Sigma = \{0, 1, \alpha, \beta\}$ ,  $L = \{\alpha^n x \beta^n : x \in \{0, 1\}^*\}$ . By marking every  $\alpha$  and falling back on every  $\beta$ , an FBA can read through  $x$  at least  $n$  times, a linear dependence. However, a weak-FBA can read each symbol in the tape exactly 1 or 2 times.

Perhaps surprisingly, it turns out that the ability to do additional computation on revisited symbols offers no additional power. More importantly, it turns out that both models recognize a fundamental class of formal language theory. Denote the class of languages accepted by FBAs as **FBA**, that by weak-FBAs as **weak-FBA**, and the class of context-free languages as **CFL**. We can prove the following:

**Theorem: weak-FBA = FBA = CFL**

**Proof sketch:** To show that **weak-FBA**  $\supseteq$  **CFL**, we recall the classic theorem of Chomsky and Schützenberger [**chomsky\_shutz**] stating that any context-free language  $L$  can be written as  $L = R \cap h(D_k)$ , where  $R$  is a regular language,  $D_k$  denotes the Dyck language of balanced parentheses of  $k$  kinds, and  $h$  is a homomorphism, mapping any symbol in the alphabet of  $D_k$  to a string in another alphabet. Let us take a context-free language in this form. Note that FBAs are ideal for accepting  $h(D_k)$ . The machine uses non-determinism to guess which symbol is represented by the next sequence of characters. When it guesses that it will see the image of a left parenthesis, say '{', it checks each symbol of  $h(\{)$  (and rejects if the sequence of symbols is not  $h(\{)$ ), and marks the final character (with the state  $q_{\{}$ ). For a right parenthesis, after checking for the sequence  $h(\})$ , it falls back and checks that the symbol fallen back to is marked  $q_{\{}$ ). Intersection with the regular language  $R$  is done by simultaneously maintaining the state of the automaton accepting  $R$  in a separate component of the FBA's state (in fact, one can show that the languages accepted by FBAs are closed under intersection with regular languages).

To show that **FBA**  $\subseteq$  **CFL**, we emulate the execution of a FBA with a *push-down automaton* (that is, a non-deterministic finite state automaton with the additional computational power of

one stack). By the classic result proved independently by [ChomskyCFG, SchutzenbergerCFG, Evey], the languages recognized by push-down automata are exactly CFL. The emulation uses the following trick: the stack is composed of *vectors* of states of length  $|K|$ . These vectors keep track of the execution of the FBA on every possible state on the sequence of symbols between consecutive pairs of marked symbols, and between the most recent marked symbol and the head. Whenever the FBA falls back and is in state  $q$  where  $q$  corresponds to the  $i$ -th coordinate in the stack vectors, the emulation pops the top vector on the stack and jumps to the state in the  $i$ -th coordinate, as this would be the resulting state *had the machine re-read the seen symbols starting in state  $q$* . The full proof is presented in the last section of this chapter.

## 5.5 Discussion

The Parser of Chapter 4 can be seen as a concrete hypothesis about the nature, structure, and operation of the language organ. In this chapter, we have extended this hypothesis: first, rough constituency parsing (the creation of the two highest layers of the syntactic — or constituency — tree of the sentence) can be carried out simultaneously with the main dependency parsing. Second, dependent sentences can also be parsed. For center-embedded sentences, a significant extension of the Parser is required: A working memory area stores the whole utterance, and the parser returns to the beginning of the utterance to recover the state of the Parser after processing the outer sentence, and then skips the embedded sentence and continues parsing the outer one.

Even though this maneuver was motivated by biological realism and programming necessity, we showed that it transforms the device from one that handles only regular languages to one capable of accepting *all context free languages* — and *just* these. We find this quite surprising, and possibly significant for the history of linguistic theory: Seven decades ago, Noam Chomsky sought to formalize human language and in the mid 1950s introduced CFLs expressly for this purpose. In the following two decades, this choice was criticised as too generous (not all features of CFLs are needed) and also as too restrictive (some aspects of natural language are not covered by CFLs). Arguably, this criticism was accepted by Chomsky's school of thought: Grammar remained im-

portant, of course, but context-free rules besides  $S \rightarrow NPVP$  (right-hand side unordered) were not used often. Much of NLP centered around the dependency formulation of syntax. Two-thirds of a century later, computer scientists speculating about syntax in the brain came up with a computational trick in order to handle center recursion. And this maneuver, when formalized properly, leads to a device that can recognize all CFLs.

Besides speculating on the meaning of this theoretical result, our work suggests a major open problem: If we assume that syntax in the brain is handled in a way similar to the one suggested by the Parser<sup>1</sup>, and all humans are born with a system of brain areas and fibers in their left hemisphere capable of such operation, *how do babies learn to use this device?* How are words learned and projected, presumably from the hippocampus, where they are associated with world objects and episodes, to the LEX in the medial temporal lobe? And how is each of them attached to the correct system of interneurons that are capable of changing the inhibited/disinhibited status of fibers and possibly of brain areas?

## 5.6 Proof of Theorem 1

Here we give the full proof of our main theoretical result:

**Theorem: weak-FBA=FBA=CFL.**

**Note:** We say a symbol on the tape is “marked” whenever its type is some  $q \in K$ .

**Lemma:** For any strong-FBA, there exists a strong-FBA recognizing the same language that is deterministic whenever the input symbol is marked  $s$  (that is, for every state and symbol pair  $q, \alpha$ , there is at most one rule  $((\alpha, s, q), (q', s)) \in \Delta$ ).

**Proof:** this is shown using essentially the same reduction from non-deterministic to deterministic finite state automata (FSA), since when at a seen symbol, the FBA cannot mark or fallback and is hence in a FSA-like regime. Concretely, if  $K$  is the original state set, the state set of the new FBA is  $2^K$ .  $\Delta$  contains the same rules when the input is in state  $f$  or is marked (we represent states of the

---

<sup>1</sup>A far-fetched assumption, of course, but one that is somewhat justified by the fact that there is no competing theory that we are aware of.

original FBA with the singleton of that state in  $2^K$ )— whenever the input is in state  $s$ , it transitions to the  $\epsilon$ -closure of the subset represented by the state (i.e.,  $(\alpha, s, S) \rightarrow (S', s)$  iff  $S'$  is the set of all states that can be reached from a state of  $S$  on symbol  $\alpha$ , before or after any epsilon transitions). Additionally, whenever the FBA reads symbol marked with anything other than  $s$ , or reaches the end of the tape, if the current state-set  $S$  is a non-singleton, it non-deterministically transitions to the singleton of any  $q \in S$  (thereby returning to a “regular” state of FBA, and moving all the non-determinism away from  $s$  symbols). The new FBA is deterministic on  $s$ -inputs and recognizes the same language: any transition through a sequence of  $s$ -states corresponds to a specific non-deterministic transition to a single state from the final state-set at the first non- $s$  symbol.  $\square$

The main technical result is showing that a pushdown automaton (PDA) can simulate a **strong-FBA**, i.e., that **strong-FBA**  $\subseteq$  **CFL**.

**Proof of theorem:** By the lemma, without loss of generality we can assume that the strong-FBA is deterministic when the input has type  $s$ . Let  $K = \{q_1, \dots, q_t\}$  be the state set of the strong-FBA. The PDA will have state set  $K$ , and stack alphabet  $\Sigma \times K \times K^{|K|}$ , that is, tuples of a symbol, a state, and *state vectors*. We define the “ $s$ -transition on  $\alpha$ ” of a state  $q$  to mean the (deterministic) FBA transition rule with left-side  $(\alpha, s, q)$ .

The PDA simulates the execution of the strong-FBA. When in state  $q'$  and on fresh tape symbol  $\alpha$ , the PDA will:

- 1) Non-deterministically select a rule with left side  $(\alpha, f, q')$  on the left-hand side. Let  $(q, \sigma)$  be the right-hand side. That is,  $\sigma \in \{s, \checkmark, \leftarrow\}$ .
- 2) Update the state to  $q$ .
- 3) If the stack is non-empty, pop the top element  $(\beta, p, (r_1, \dots, r_t))$  from the stack. For each vector coordinate  $i \in [t]$ , apply the  $s$ -transition on  $\alpha$  to each  $r_i$ — push the updated tuple  $(\beta, p, (r'_1, \dots, r'_t))$  back to the stack.
- 4) if  $\sigma = \checkmark$ , push  $(\alpha, q, (q_1, \dots, q_t))$  to the stack.
- 5) if  $\sigma = \leftarrow$ , the corresponding PDA transition does nothing else, but enters the following loop:
  - 5.1) Pop the top element  $(\beta, p, (r_1, \dots, r_t))$  of the stack and sample a rule of the FBA that tran-

sitions on the marked symbol, that is, a rule of the form  $(\beta, p, q) \rightarrow (q', \sigma)$ . Note that  $\sigma \in \{s, \leftarrow\}$ . If the stack is non-empty, pop the next element,  $(\gamma, w, (u_1, \dots, u_t))$  and “apply”  $(r_1, \dots, r_t)$  to the state vector— that is, for each  $i \in [t]$ , if  $u_i = q_j$ , replace  $u_i$  with  $u'_i := r_j$ . Push the resulting pair  $(\gamma, w, (u'_1, \dots, u'_t))$  back onto the stack.

5.2) if  $\sigma = \leftarrow$  (which we call a “fallback-again” rule of the FBA), the PDA updates the state to  $q'$ , and returns to the beginning of 5.1). If  $\sigma = s$  (which we call a non-“fallback-again” rule), the PDA updates the state to  $r_i$  (that is, the  $i$ -th coordinate in the stack-vector from 5.1), where  $i$  is the index of  $q'$ , i.e.  $q' = q_i$ . Because the FBA can have “fallback-again” rules, the PDA can repeat 5.1 multiple times, and stops when it selects a non-“fallback-again” rule (or rejects).

For any execution of the PDA, it will have  $|x| + F + M$  transitions, or steps, where  $x$  is the input,  $F$  is the number of times the PDA *repeated* step 5.1 (i.e. executed a “fallback-again” rule of 5.2), and  $M$  is the number of times it ends the loop of 5), i.e. selects a non-“fallback-again” rule. Each step  $i$  will correspond to a “strong”-step of an equivalent FBA execution, which is a transition where the input symbol has type not equal to  $s$  (in other words, a step where the FBA either processes a fresh symbol, or falls back to a marked symbol).

Claim: There is a one-to-one correspondence between executions of the FBA on  $x$  with executions of the PDA on  $x$ , such that, for each such pair, the PDA simulation maintains the following invariant at each step  $i$  with respect to the FBA:

a) the PDA state *after* the  $i$ -th step is equal to the FBA state immediately before the  $i + 1$ -th strong step (or the final state, if  $i = |x|$ ), and the position of the PDA head is equal to the position of the last  $f$ -symbol seen by the FBA.

b) the PDA stack has  $l$  elements, where  $l$  is the number of marked symbols after the  $i$ -th strong-step of the FBA, and

c) the  $k$ -th stack element consists of the  $k$ -th marked symbol and type in the FBA execution, and a vector that contains for each state  $q_j$ , the execution of  $s$ -transitions starting from the symbol immediately after the  $k$ -th marked symbol, up to and including either the  $(k + 1)$ -th marked symbol, or the head, whichever comes first.

Note that if the claim is true, by part a) of the invariant, the final state of the PDA is an accept state iff the equivalent FBA execution accepts  $x$ , so the theorem is proved.

We prove the claim recursively. It is trivially true at the beginning. Now, assume that after  $i - 1$  steps of execution, the FBA corresponds to an execution of  $i - 1$  strong-steps of the PDA, and that the invariant is true.

At step  $i$ , we consider all possible strong-steps of the FBA.

First, for any rule of the type  $((\alpha, f, q'), (q, s)) \in \Delta$ : by a) of the invariant the FBA and PDA must both be scanning a new symbol  $\alpha$  and are in state  $q'$ . The PDA simulation can sample this rule by step 1), which results in updating the state  $q'$  to  $q$ , satisfying a). Since no marked symbols were added or removed by this kind of FBA transition, b) is trivially satisfied. Finally, 2) updates the top state vector on the stack with the  $s$ -transitions on  $\alpha$ — since it previously represented the  $s$ -transitions from the most recent marked symbol to the previous tape symbol, it now represents an execution up to and including the current symbol, that is, c) is maintained.

For any rule of the type  $((\alpha, f, q'), (q, m)) \in \Delta$ : similarly, the PDA simulation can select this rule in 1), updating the state to  $q'$ , giving a). The number of marked symbols in the FBA execution changes if this rule were applied, so invariants b) and c) must be checked. By step 3), we push  $(\alpha, q, (q_1, \dots, q_t))$  to the stack, immediately satisfying b). Note that the penultimate element of the stack is now “frozen”, showing an execution of  $s$ -transitions from the previous marked-symbol up to and including  $\alpha$ , the new marked symbol. The vector of the top element of the stack,  $(q_1, \dots, q_t)$ , trivially represents an execution of every state from the new marked symbol to the head (since it is empty). Hence c) is satisfied.

For any rule of the type  $((\alpha, f, q'), (q, \leftarrow)) \in \Delta$ : again both the PDA and FBA are at a fresh tape symbol, but after this step, the FBA head will be at the previously marked symbol. By step 1) the PDA can sample this rule, updating the state to  $q$ , which satisfies a). Note that b) and c) are trivially satisfied.

Next for any strong rule of the type  $((\beta, p, q'), (q, \leftarrow))$ , the FBA head must be at a marked symbol, and immediately before the next strong step, it will be at the previous marked symbol in

state  $q$ . Indeed, the FBA can sample this rule in 5.1, updates state to  $q$  satisfying a), pops the top stack vector satisfying b), and as for c), the PDA “applies” the state vector of the popped tuple,  $(r_1, \dots, r_t)$ , to the *next* state vector on the stack,  $(\gamma, w, (u_1, \dots, u_t))$ . If  $u_i$  is the  $s$ -transition of  $q_i$  from the symbol after  $\gamma$  up to and including  $\beta$  (this is guaranteed by the invariant) and  $u_i = q_j$ , then  $r_j$  is exactly the  $s$ -transition of  $q_i$  from the symbol after  $\beta$  up to and including the head, ensuring c).

Finally if the rule is of the type  $((\beta, p, q'), (q, s))$ , the FBA head must be at a marked symbol, and immediately before the next strong step it *passes through every  $s$ -symbol between the marked symbol and the fresh symbol*. The corresponding PDA transition satisfies b) and c) as in the previous case (a marked symbol is removed and the other vectors in the stack are updated), but this time, we also update the current state based on the popped vector; since, by b), it contained the  $s$ -transition of each state from the marked symbol to the head, it correctly yields the state of the FBA before the next fresh symbol.

## Chapter 6: Planning in the Blocks World

This chapter uses material from [82]. The material has been lightly refurbished for the purposes of this thesis, but the bulk of the results and content presented represents the collaborative effort with the original authors: Francesco D’Amore, Pierluigi Crescenzi, Emanuele Natale, and Christos Papadimitriou.

### 6.1 Introduction

Towards understanding the computational expressiveness of the NEMO model, and hence its potential as the sort of bridging “logic” sought by Axel, the previous chapters have pursued the demonstration (both empirical and experimental) of reasonably complex cognitive phenomena formulated and implemented within NEMO. This includes the creation and manipulation of assemblies of Chapter 3, and the results of Chapters 4 and 5, a Parser implemented in NEMO was demonstrated to analyze syntactically reasonably complex sentences of English (and which be generalized to more complex features as well as other natural languages).

*The contribution of this chapter is to demonstrate that a program in NEMO is capable of implementing reasonably sophisticated stylized planning strategies – in particular, heuristics for solving tasks in the blocks world [83, 84]. A blocks-world configuration is defined by a set of stacks, where a stack is a sequence of unique blocks, each sitting on top of the previous one. A stack of size one is just a block sitting on the table (see e.g. Fig. 6.1-A). A configuration can be manipulated by moving a block from the top of a stack (or from the table) to the top of another stack (or to the table). A task in the blocks world is the following: Given a starting configuration  $C_{init}$  and a goal configuration  $C_{goal}$ , find a sequence of actions which transforms  $C_{init}$  into  $C_{goal}$ . It was shown in [85] that solving a task in the blocks world with the smallest number of actions is*

NP-Complete, and it was observed that the following provides a simple 2-approximation strategy: *Move to the table all blocks that are not in their final positions, and then move these blocks one by one to their final positions.*

Here we implement this strategy in the NEMO. From the exposition of this implementation and demonstration — which happens to employ representations and structures of a different style from those needed for language tasks [86, 87] — we believe that it will become clear that more complicated heuristics for solving related tasks can be similarly implemented in the NEMO.

In fact, the kind of representations needed for planning, involving long “chains” of assemblies linked through strong synaptic connections, reveals a limitation of NEMO (or rather, a deeper understanding of its capacity) which was not apparent before: we find empirically that there are limits — depending on the parameters of the execution model, such as the number of excitatory neurons per brain area, synaptic density, synaptic plasticity, and assembly size — on the length of such chains that can be implemented reliably. As chaining is also used in the Turing machine simulation demonstrating the completeness of the NEMO [86], such limitations are significant because they bound from above the space complexity — and therefore the parallel time complexity — of NEMO computations. We briefly discuss and quantify this issue in the experimental validation section.

## **6.2 Background and related work**

Terry Winograd introduced the blocks world half a century ago as the context for his language understanding system SHRDLU [88], but since then blocks-world planning has been widely investigated, primarily because such tasks appear to capture several of the difficulties posed to planning systems [83, 85]. There has been extensive work in AI on blocks world problems, including recently on leveraging ANNs for solving them, and learning to solve them from examples (e.g., the Neural Logic Machines of [89], or Neural Turing Machines, which are used for related problem-solving tasks [90]).

Bridging the gap between low-level models of neural activity in the brain and high-level sym-

bolic systems modelling cognitive processes is a fundamental open problem in artificial intelligence and neuroscience at large [91, 92]. Several computational cognitive-science papers address the problem of solving (or learning to solve) block-worlds tasks in higher-level computational models of cognition, such as ACT-R or SOAR (see for instance [93, 94, 95]). In contrast to the present paper, however, these works utilize high-level languages and data structures for the programming of these systems, without providing a link, as we do, to the behavior of stylized neurons and synapses, in an effort to remain as faithful as possible to the ways animal brains would solve these tasks. Less related to our problem is the literature on block stacking (see, for example, [96, 97]). These papers the focus on the ability of humans and chimpanzees to place a block on top of an existing tower without toppling it. Finally, it is worth mentioning some previous works on solving planning tasks through spiking neural networks, such as [98, 99], in which the attention is more focused on learning world models.

### 6.3 The Blocks World NEMO Program

A blocks world (BW) configuration is a set of *stacks*, where each stack is a sequence of *blocks*, from top to bottom. Each block is assumed to be a unique integer between 1 and  $s$ . Two BW configurations, the initial and the target configuration, constitute the input to the NEMO program (see Figure 6.1-A). We shall at first concentrate on *configurations with a single stack* — already a meaningful problem — and we shall eventually graduate to multiple stacks (see subsection 3.5 below). We next describe four NEMO programs: (a) a program that takes the input — a sequence of integers representing a stack — and creates a list-like structure, in a set of brain areas and fibers, for representing the stack; (b) a program that removes the top block of a stack thus represented; (c) a program that adds a new block to the represented stack; and (d) a program for computing the *intersection* of two stacks represented this way, that is, the longest common suffix of the two sequences, read from bottom to top.

All four programs work on a common set of brain areas connected with bi-directional fibers: the area BLOCKS contains a fixed assembly for every possible block (these assemblies are special,

in that each can be activated explicitly as the presentation of the corresponding number in the input). There are four other areas used in our NEMO programs: HEAD, NODE<sub>0</sub>, NODE<sub>1</sub>, and NODE<sub>2</sub>. HEAD is connected to the NODE<sub>0</sub> area via fibers, while each NODE area is connected to BLOCKS, and to each other in the shape of a triangle: NODE<sub>0</sub> is connected with NODE<sub>1</sub>, which is connected with NODE<sub>2</sub>, which is connected with NODE<sub>0</sub> (see Figure 6.1-B). All of these areas are standard NEMO brain areas, containing  $n$  randomly connected neurons of which at most  $k$  fire at any time.

### 6.3.1 The Parser

#### Algorithm 5: PARSER ( $S$ )

```

input: a stack  $S$  of blocks  $b_1, b_2, \dots, b_s$ .
1 disinhibitArea ({BLOCKS, HEAD, NODE0});
   disinhibitFiber ({(HEAD, NODE0), (NODE0, BLOCKS)});
2 activateBlock ( $b_1$ ); strongProject();
3 inhibitArea ({HEAD});
   inhibitFiber ({(HEAD, NODE0), (NODE0, BLOCKS)});
4 foreach  $i$  with  $2 \leq i \leq s$  do
5    $p = (i - 2) \bmod 3$ ;  $c = (i - 1) \bmod 3$ ;
6   disinhibitArea ({NODE $c$ });
   disinhibitFiber ({(NODE $p$ , NODE $c$ ), (NODE $c$ , BLOCKS)});
7   activateBlock ( $b_i$ ); strongProject();
8   inhibitArea ({NODE $p$ });
   inhibitFiber ({(NODE $p$ , NODE $c$ ), (NODE $c$ , BLOCKS)});
9 end
10 inhibitArea ({BLOCKS, NODE $(s-1) \bmod 3$ });

```

The parser (see Algorithm 5) processes each block in a stack sequentially, starting from the top. When it analyses the first block (see lines 1-3), the three areas BLOCKS, HEAD, and NODE<sub>0</sub>, and the fibers between HEAD and NODE<sub>0</sub> and between NODE<sub>0</sub> and BLOCKS are disinhibited. The block assembly is then activated and a strong projection is performed, thus creating a connection between the assembly in BLOCKS corresponding to the block and an assembly in NODE<sub>0</sub>, and between this latter assembly and an assembly in HEAD (see the red dashed lines in Figure 6.1-C1). Successively, the HEAD area and the fibers between HEAD and NODE<sub>0</sub> and between NODE<sub>0</sub>

and BLOCKS are inhibited. For each other block in the stack (see lines 5-8), the NODE area next to the one (i.e.,  $NODE_{i \bmod 3}$ ) currently disinhibited (i.e.,  $NODE_{i+1 \bmod 3}$ ) is disinhibited, and the fibers between this NODE area and the BLOCKS area and between the two NODE areas are disinhibited. The next block assembly is then activated and a strong projection is performed, creating a connection between the assembly in BLOCKS and an assembly in the NODE area just disinhibited, and between this latter assembly and the assembly previously activated in the previous NODE area (see the red dashed lines in the figures 6.1-C2,C3,C4). After this and before the next block, this latter NODE area and the fibers between it and the NODE area after it, and those between the NODE area after it and the BLOCKS area, are inhibited.

The final data structure is a *chain* of assemblies starting from an assembly in HEAD and passing through assemblies in the NODE areas (see Figure 6.1-C6). Note that this chain can contain more than one assembly in the same NODE area: for instance, in Figure 6.1-C6, the chain contains two assemblies in  $NODE_0$  and  $NODE_1$ . Each assembly in the chain is also connected to the assembly in BLOCKS corresponding to a block in the stack. For instance, the sequence of such assemblies in Figure 6.1-C6 corresponds to the sequence of blocks 4, 5, 3, 1, 2, which is exactly the sequence of blocks in the stack from top to bottom (see the left part of Figure 6.1-A). Note that Algorithm 5 uses a constant number of brain areas (that is, five), independently of the number of blocks in the stack.

### 6.3.2 Removing the Top Block

In order to implement the algorithm which transforms an input stack of blocks into a target stack of blocks, we start by describing a NEMO program to remove a block from the top of a stack. This program uses the same areas and fibers of the parser described in the previous section (see Figure 6.1-B), with the addition of fibers between HEAD with  $NODE_1$ , and HEAD with  $NODE_2$ . Intuitively, these fibers are needed to allow changing the head of the chain representing the current stack, without having to shift all the assemblies one position to the left.

The NEMO program, which “removes” the block from the top of the stack, uses the connections

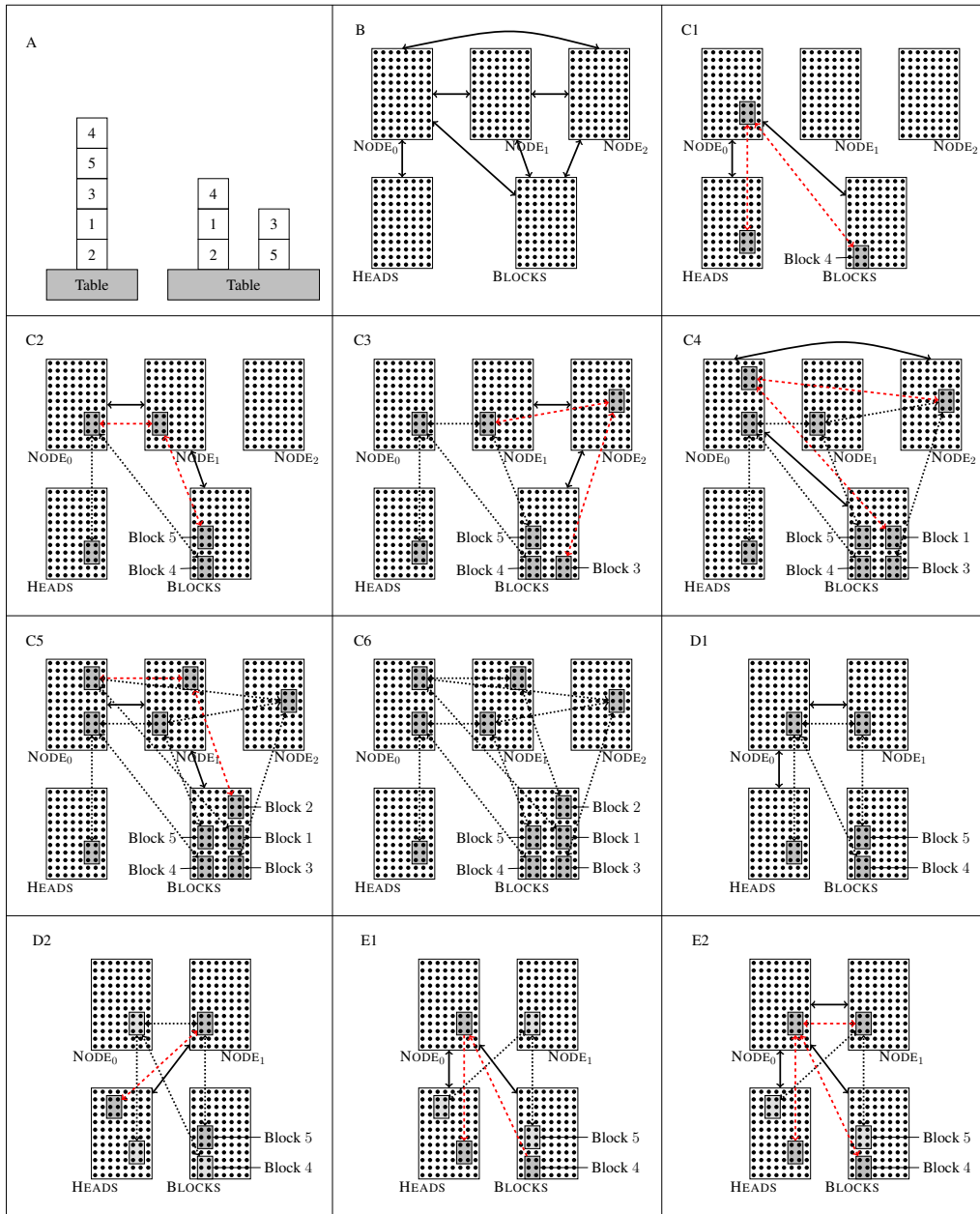


Figure 6.1: **A.** Two BW configurations. In the rest of the figure, we consider the BW configuration shown on the left. **B.** The five areas used by the parser, along with the connections through fibers. **C1-6.** The behavior of the parser. The black solid lines denote the fibers of Figure B which are disinhibited. The red dashed lines denote the newly created connections between assemblies in different areas, while the black dotted lines denote the connections previously created. **D1-2.** The behavior of the NEMO program which removes the block from the top of a stack, with input the data structure resulting from the parser execution (only the areas involved in the remove operation are shown). The black solid lines denote the fibers which are disinhibited. The red dashed lines denote the newly created connections between assemblies in different areas, while the black dotted lines denote the already existing connections. **E1-2.** The behavior of the NEMO program which put the block 4 on top of the stack, above the block 5. The black solid lines denote the fibers which are disinhibited. The red dashed lines denote the newly created connections (unidirectional and bidirectional) between assemblies in different areas, while the black dotted lines denote the already existing connections.

created by the parser in order to activate the assembly in the  $NODE_1$ , which is connected to the block just below the top block (that is block 5 in Figure 6.1-D1,D2). This is done by projecting from the HEAD into  $NODE_0$ , and projecting from  $NODE_0$  into the  $NODE_1$  (see Figure 6.1-D1). Through strong projection, the program successively creates a new connection from the active assembly in the  $NODE_1$  area to a new assembly in the HEAD area (see the red dashed line in Figure 6.1-D2).

Note that the connections between the light gray assemblies in Figure 6.1-D2 are still active, but they will not be used in the future since the last active assembly in the HEAD area is now connected to the assembly in the  $NODE_1$  area. These connections, indeed, might later disappear because of a process of homeostasis, which can be modeled in NEMO through a sort of “renormalization” (as described in [86]). In a certain sense, the system will slowly “forget” which block was on the top of the stack, before a removal operation.

The removal of the top block can be repeated as many times as the number of blocks in the stack. The only difference is that the activation of the assembly in NODE corresponding to the block below the top one is done by projecting HEAD into the NODE area corresponding to the top block, and then projecting from this NODE area to the one following it (in modular arithmetic).

In order to maintain an updated representation of the blocks world configuration, we use four additional brain areas to store the chain of blocks which have been removed and that, hence, are currently on the table. This chain can be implemented in NEMO exactly the same way we did when parsing a stack of blocks. Then, when we want to read the current data structure stored in this NEMO system, we examine the stack of blocks represented in HEAD and the NODE areas, as well as the chain of blocks on the table in the additional areas.

### 6.3.3 Putting a Block on Top of the Stack

The second operation we need in order to implement a minimal planning algorithm for the blocks world problem is *putting* a block on top of the stack. The NEMO program, for this operation first projects the block from in BLOCKS into the NODE area preceding (in modular arithmetic) the

NODE area currently connected to HEAD, and then projects the newly created assembly into HEAD (see Figure 6.1-E1). Successively, the program executes a strong projection between the four areas in order to correctly connect them (see Figure 6.1-E2). Once again, an active connection between the HEAD area and a NODE area will still exist after the execution of the NEMO program, but this connection will not be used in the future.

#### 6.3.4 Computing the Intersection of Two Stacks

The pop and put operations described in the previous two sections are sufficient to implement a simple planning algorithm, which consists in moving all the blocks on the table (by using pop), and by then moving the blocks on the table on top of the stack (by using put) according to the target stack.

In order to improve this algorithm and execute the two-approximation algorithm described in the introduction, we need a NEMO program which implements a third operation, that is, finding the *intersection* of two stacks. This operation looks for the common sub-stack of the two stacks (starting from the bottom) and return the highest block in this sub-stack. Then only the blocks above this block have to be moved on the table and reassembled in the right order.

In a nutshell, this can be achieved in NEMO by first reaching the bottom of the two stacks which have to be compared, and then proceeding upwards until we find two different blocks, or the end of one of the two stacks.

#### 6.3.5 Multiple Stacks

So far in this exposition we have concerned ourselves with configurations consisting of one stack. In our experiments (see the next section) we have implemented up to five stacks by employing a different set of four areas for each stack. This is a bit unsatisfactory, because it implies that the maximum number of stacks that can be handled by the brain is encoded in the brain architecture. There is a rather simple — in principle — way to achieve the same effect by re-using the same four areas; we have an initial implementation of this idea, which we intend to test in the

future.

With multiple stacks one has to solve the *matching problem*: identifying pairs of stacks in the input and output that must be transformed one to the other. Naively, this can be done by comparing all pairs of stacks, but this entails effort that is quadratic in the number of stacks. This latter strategy is the one currently employed in our experiments. In the future, we intend to test a more principled way, based on *hashing* the stacks into their bottom element, and attending to any collisions.

## 6.4 Experiments

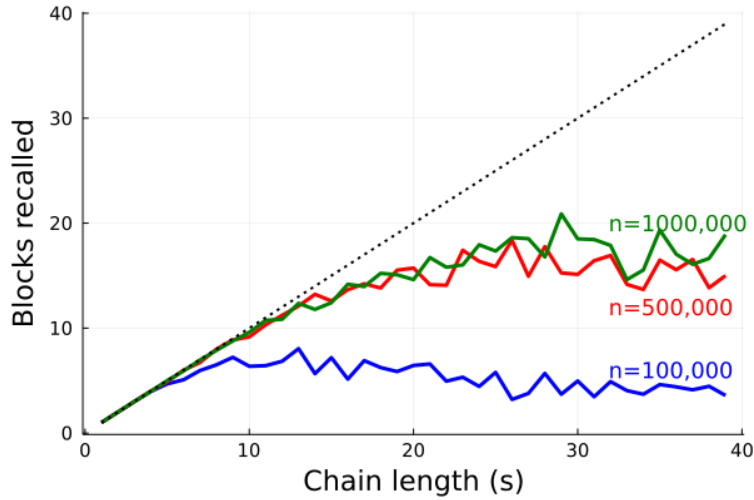
A software system for programming in NEMO, as well as implementations of the algorithms described in this paper, have been written in Julia [100]. We make use of the Java generator for BW configurations available at [101], based on [84]. We ran experiments on over 100 blocks-world configurations, with up to five stacks and 10, 20, and 30 blocks. The algorithm worked correctly in every instance. We have used various settings of the parameters  $n, k, p, \beta$  – a particularly good set of parameters is  $n = 10^6, k = 50, p = 0.1, \beta = 0.1$ . Interestingly, the algorithms do not work in all parameter settings, because of limits on the chaining operation (see the next discussion). The Julia source code can be found at [102].

In general, the amount of rounds of strong project (parallel spikings of neurons) needed to carry out the BW tasks seems to be around 35 spikes per block processed (parse, popped, or pushed), which, assuming roughly 50 Hz spikes for excitatory neurons in the brain, is around 1.4 seconds per operation.

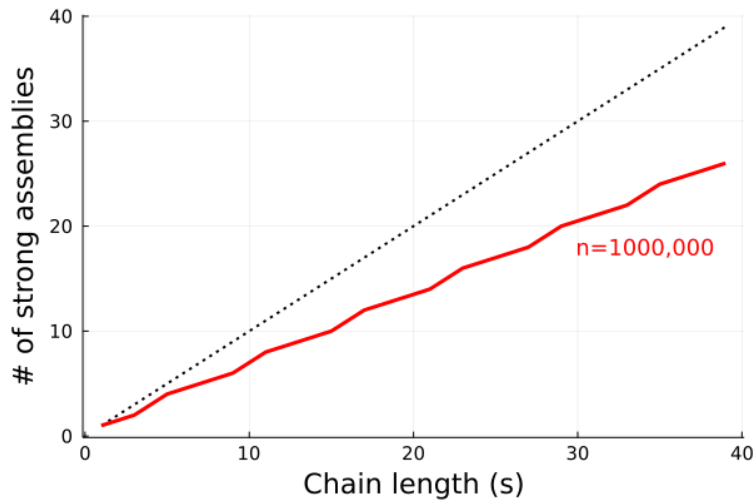
## 6.5 Chaining, and its limits in NEMO.

An unexpected finding of our simulations is that they are stable only under very specific parameter settings. The bottleneck of the planning algorithms is in parsing the chain of blocks, that is, memorizing the sequence of blocks so they can be read out reliably. In isolation we call this operation “chaining”.

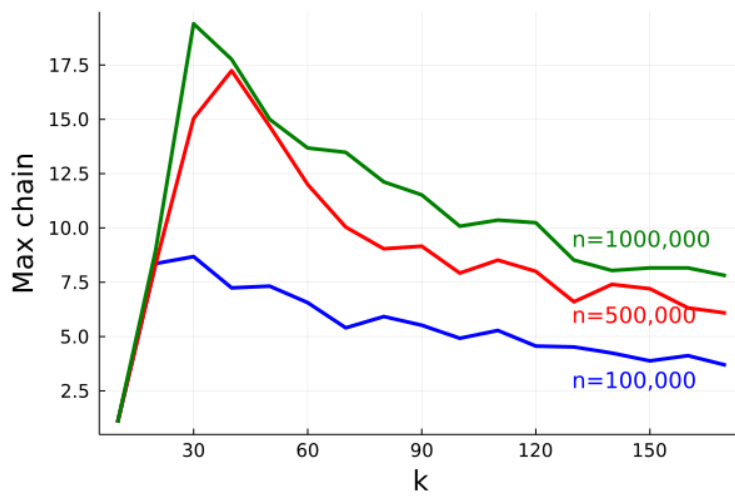
The results in this section, which describe some properties and limits of chaining, can be viewed



(a)



(b)



(c)

Figure 6.2: Experiments on the “chaining” operation, the bottleneck of the NEMO planning algorithm. (a) shows number of blocks correctly chained for various chain length; (b) shows number of “strong” assemblies formed in chaining; (c) shows maximal chain length that is correctly parsed for varying  $k$ . (b) and (c) show averages over 50 trials per parameter setting (exact numbers, including sample standard deviation, are provided in the full version at [103]). In these charts,  $p = \beta = 0.1$  was used, in (a) and (b)  $k = 50$ .

as theoretical properties of the AC. First, we find it is only possible to chain a rather limited number of blocks. For instance, even though with  $n = 10^6$  and  $k = 50$  there is, at least in theory, space for  $10^6/50 = 200000$  non-overlapping assemblies, even with strong  $p$  and  $\beta$ , we can only reliably chain up to 20 blocks. This is illustrated in Figure 6.2a, which shows how many of  $s$  blocks were successfully read out after chaining. Generally, for higher values of  $n$  (and a higher  $n : k$  ratio), longer portions of the chain tend to be correctly stored, but the operation is highly noisy: in some trials it will fail and then succeed for a *longer* chain. Indeed, unlike the assembly operations described in [86] (Project, Merge, and so on) which are either stable with overwhelming probability under appropriate parameters, or do not succeed if the parameters are not appropriately strong, chaining appears to push the computational power of NEMO to its limits, and often succeeds or fails between repeated trials with the same parameters.

One can also look at a related property: after chaining, how many of the assemblies in the  $\text{NODE}_i$  areas during readout are “strong” in the sense that they pass  $\text{ISASSEMBLY}()$  with a high threshold (i.e. firing those  $k$  neurons recursively results in the same set of  $k$  winners)? Interestingly, this proportion, which is significantly less than the maximum of  $s$ , does not change significantly when we vary  $n, p, \beta$ —there appears to be a natural proportion of strong assemblies formed during chaining (Figure 6.2b).

Finally, in Figure 6.2c we varied  $k$  and found the maximally long chain that succeeded completely. These experiments again showed that for higher  $n : k$  ratio, longer chains are possible, and that for each setting of  $n$  there is a narrow window of optimal  $k$  that allows for the longest chains—above of this range, as we increase  $k$  the maximum chain does not change, i.e. it appears to settle to some natural lower bound. A more thorough analysis of chaining is an important direction in NEMO theory, since such maneuvers could be subroutines in various cognitive processes (for instance, [87] suggest using it for processing chains of identical parts of speech, such as multiple adjectives in a noun phrase).

## 6.6 Conclusions and Future Directions

The aim of this work is not so much to produce a performing system, but to demonstrate experimentally that reasonably large and complex programs in the assembly calculus can execute correctly and reliably, and in particular can implement in a natural manner planning strategies for solving instances of the blocks world problem. In fact, the implementation of these strategies is based on the realization of a *list-like data structure* which makes use of a *constant* number of brain regions. Confirming theoretical insights, we have experimentally found that the structure's reliability depends on the ratio between the number of neurons and the size of the assemblies in each region — even though the dependency was a bit more constraining than we had expected. The reasons and extent of this shortcoming must be the object of further investigation.

We have also shown how simple manipulations of the data structure (such as the top, pop, and append operations) can be realized by making use of a constant number of brain regions. These manipulations allowed us to implement planning strategies based on two basic kinds of moves, that is, moving the block from the top of a stack to the table, and putting a block from the table to the top of a stack. All our programs work for an *arbitrary* number of blocks and a *bounded* number of stacks — while current work involves implementing a version with an arbitrary number of stacks.

After syntactic analysis in language and blocks world planning, what comes next as a compelling stylized cognitive function, which could be implemented in the NEMO? There is work currently in submission dealing with *learning* through assemblies of neurons. Two further realms of cognition come to mind, and they happen to be closely related: *Reasoning*, as well as *planning and problem solving* in less specialized domains than BW. It would be interesting to figure out the most natural way for assemblies and their operations to carry out deductive tasks, and, even more ambitiously, to carry out planning in the context of logical and constraint-based formalisms of planning, see for example [104].

## Chapter 7: Planning in the Blocks World

This chapter uses material from [14]. The material has been refurbished for the purposes of this thesis, but the bulk of the results and content presented represents the collaborative effort with the original co-author, Christos Papadimitriou.

### 7.1 Introduction

We briefly summarize the results of the previous few chapters, which describe novel algorithms for problems corresponding to cognitive phenomena. In Chapter 3, a computational system called the Assembly Calculus was proposed, based on a simplified mathematical model of spiking neurons and synapses, which reflects the basic elements and tenets of Neuroscience: brain areas, excitatory neurons, local inhibition, plasticity (see the next section for a detailed description of the enhanced version of this model used here). Within this framework, neuromorphic computational systems simulating certain large-scale cognitive phenomena were implemented in the previous chapters: a system for planning in the blocks world in Chapter 6; a system for learning to classify representations through few-shot training [105]; and, most relevant to this chapter, a system for *parsing sentences* in English and other languages (Chapters 4 and 5).

We believe that pursuing this research program of constructing more and more ambitious neuromorphic artifacts simulating cognitive phenomena is important, for at least two reasons. First, each step on this path entails concrete progress in the bridging problem, as more and more advanced domains of cognition are explored through artifacts consisting of reasonably realistic and brain-like, if stylized, systems of neurons and synapses. Second, further progress in this direction may be of interest to Artificial Intelligence: Despite amazing advances over the past ten years, arguably AI still lags behind human brains in several important dimensions: grounded language,

continual learning, originality and inventiveness, emotional and social intelligence, and energy usage. Creating intelligent artifacts that are more brain-like, and rely on modes of learning other than backpropagation, may eventually point to new possible avenues of progress for AI.

The biologically plausible parser of Chapters 4 and 5 includes a lexicon containing neural representations of words. It is assumed that each neural representation of a word is wired so that, when excited by an outside stimulus, it sets in motion specific neural activities inhibiting and/or disinhibiting remote brain areas that are associated with the word's syntactic role (verb, subject, etc.). This works fine for the purposes of the parser, except that it leaves open perhaps the most important questions: How are these word representations created? How are these neural activities set up in the infant brain and how are they associated with the representation of each word, thus implementing the word's part of speech? And how are those other brain areas labeled with the appropriate syntactic roles? In other words, *how is language acquired in the human brain?*

This is the question we set out to answer in this paper.

We seek to create a *neuromorphic language organ*: a tabula rasa of neural components — roughly, a collection of brain areas with randomly connected neurons, with certain additional neural populations, all consistent with basic Neuroscience and plausibly set in place during the infant's development — which, upon the input of modest amounts of grounded language, in any natural language, will acquire the ability to comprehend and generate syntactically and semantically correct sentences in the same language — definitions of all these terms forthcoming.

One important remark is in order: By designing such a system, we are not articulating a scientific theory about the precise way in which language is implemented in the human brain — a theory to be tested by experiments on human subjects. The artifact we create is a *proof of concept*, an existence theorem stating that something akin to a language organ can be put together with basic neuroscientific materials which can be plausibly delivered by a biological developmental apparatus. We believe that this has not been done before. But, having said that, we have taken care that aspects of the system we present here are consistent with the consensus in neurolinguistics about the nature of the language organ, *wherever such consensus exists*; we point out instances of such

convergence throughout the paper.

## 7.2 On Language in the Brain

When it comes to language in the brain, much less is known with certainty than about neuron cellular dynamics; see [106, 70, 107] for recent books on the subject. Still, it is impossible to survey the entire field. Here, we summarize the state of our knowledge of the language organ most pertinent to this work.

The language organ carries out two main functions: speech production and speech comprehension. There is a broad consensus that, in the systems responsible for both functions, there exist abstract representations for each word in the language within a centralized lexical area; this area can be thought of as a hub-like interface between the phonological subsystem and the semantic representations of each word. Though not uncontroversial, there is also growing evidence that these representations are *shared* between production and comprehension systems — they are believed to reside in the mid and mid-posterior MTG [108]. On the other hand, the *semantics* of nouns and verbs are represented in a distributed way across many brain areas, many at the periphery of the motor, visual, and other sensory cortex, and the aforementioned word representations are richly connected to these areas [109, 110, 111].

Nouns and verbs differ in some of the context areas with which they are most strongly connected. Parts of the motor cortex are much more strongly involved in the processing of verbs than that of nouns (namely the PLTC and the pSTS subarea), whereas a different part of the motor cortex is more active in the processing of nouns involving action (i.e., tools and limbs) than verbs; see [112] and [113] for surveys. Furthermore, areas of the motor cortex that are activated in response to perceiving someone *else* perform an action, known as *mirror cells*, are activated much more for verbs than for nouns; for a review on motor and mirror area recruitment for verbs vis-a-vis nouns, see [114, 115]. In addition to involving different context areas, it is also known that there is a separation between the systems for the perception and generation of nouns and that of verbs, and there is growing evidence that this may, at least partly, be because noun and verb lexical representations

reside in different subparts of the mid and mid-posterior MTG, that is, the lexical area [116, 117, 26]. Our language organ model will reflect these principles by having separate lexical areas for nouns and verbs, and featuring contextual that are connected *exclusively* to each of the noun and verb areas — in addition to many shared context areas.

Neurolinguists also strongly suspect that there is an abstract phonological representation for each word (in an area called Spt) that is connected to the word’s centralized lexical representation, and that the same representation is used in both perception and production [118, 119]. This representation can be thought of as containing implicit representations of sequences of phonemes and interfacing to the sensorimotor subsystems for the perception and production of these words. In our work, we abstract away phonological processing and acquisition, and will have a special phonological input/output area that is shared by production and perception.

To summarize, we have the following simplified picture of language in the brain: each word has a root representation in a lexical hub area, likely within different sub-areas for nouns and verbs, which is connected to a phonological representation of the word — representations which are used both for recognizing and for articulating words. The lexical hubs are richly connected to many sensory and semantic areas across the brain through which the many complex shades of meaning and nuances of a word are represented; crucially, nouns and verbs have strong connections to different context areas.

### **7.3 Psycholinguistic theories**

The most important comparisons to our work are the existing psycho- and neurolinguistic models of language processing. Among the most influential and established theories are the Lemma Model for production [120], the Dual Stream Model of language perception [121], and the Hub-and-Spoke model for the semantic representation of words [122]. While there is much debate regarding in which ways these models can be combined, they all have in common the basic consensuses, or near-consensuses, outlined in the previous section. One important contribution of our work is that it constitutes a *concrete, neuronal implementation* of the underlying common core of

these three mainstream models of language processing. That is, whereas at the highest level the Lemma Model posits the existence of word lemmas connected to phonological representations and a hub-and-spoke like semantic network, and the Dual Stream Model predicts a lexical interface between the integration of phonological input and the semantic and syntactic features of a word, our work *fully implements*, in terms of realistic stylized neurons, the basic underlying mechanisms of these models. Importantly, our model explains how the *lexical representations* common to these models can be acquired from grounded input.

**A toy language.** We will shortly define a language organ in NEMO that will learn from sentences of a toy language with  $l$  nouns and  $l$  intransitive verbs, where  $l$  is a small parameter that we vary in our experiments. We denote the combined lexicon as  $L$ . In this language, all sentences are of length two: “cats jump” and “dogs eat.” Importantly — and this is the hard part of our experiment — the language can have either SV (subject-verb) principal word order (as in English, Chinese and Swahili) or VS (as in Irish, Classical Arabic, and Tagalog), and our model should succeed in either scenario.

## 7.4 The Language Organ

Our language organ, denoted  $O$ , consists of two separate *lexical areas* for nouns and verbs,  $LEX_N$  and  $LEX_V$ , and an area PHON containing the phonological representations of words. It also has several *context* areas: VISUAL and MOTOR are the two basic ones, but there are several others which we denote  $CONTEXT_i$  for  $i \in [C]$ . ( $C$ , the number of additional context areas, is a parameter of the model; here  $C = 10$ ). PHON is connected through fibers with  $LEX_N$  and  $LEX_V$ , whereas VISUAL is connected with  $LEX_N$ , and MOTOR with  $LEX_V$ . All other context areas  $CONTEXT_i$  are connected to both  $LEX_N$  and  $LEX_V$ ; all these connections are two-way (see Figure 7.1). For each word  $W$ , we additionally pre-select a random subset of  $[C]$ , representing which extra context areas are implicated for the word  $W$  (for instance an olfactory area for  $W = flower$  an emotional affect area for  $W = hug$ , and so on). In our experiment, this set has only one element, denoted  $i[W]$ .

Hearing each word  $W$  by the learner is modeled as the activation of a unique corresponding assembly  $\text{PHON}[W]$  for that word in  $\text{PHON}$  for the duration of the perception of a word, that is, for  $\tau$  time steps, where  $\tau$  is another parameter of the model. We further assume that our input is *grounded*:: whenever a noun  $W \in L$  is heard it is also seen — that is, an assembly corresponding to the static visual *perception* of the object (cat, dog, mom, etc) is active in  $\text{VISUAL}$ , denoted  $\text{VISUAL}[W]$ . Similarly, an assembly corresponding to the intransitive action (jump, run, eat, etc.) in  $\text{MOTOR}$ , denoted  $\text{MOTOR}[W]$  for a verb  $W \in L$ . *These areas represent the union of the differing somatosensory cortical areas feeding into nouns and verbs covered in Section 7.2*). We also activate an assembly  $\text{CONTEXT}_{i_w}[W]$  in the extra context area corresponding to  $W$ . Importantly, the assemblies in the contextual areas ( $\text{VISUAL}$ ,  $\text{MOTOR}$  and the  $\text{CONTEXT}_i$ ) are activated throughout the perception of the entire sentence (that is,  $\tau \times |\text{sentence}|$  steps), *not* just when the corresponding word is perceived. This corresponds to the fact that the learner perceives the sentence as a whole, associated with the world-state perceived that moment through shared attention with the tutor.

Effectively, the above means that in our experiment, whereas  $\text{LEX}_N$  and  $\text{LEX}_V$  are pristine *tabulae rasae*, areas with random connectivity devoid of special structure,  $\text{PHON}$  is pre-initialized with assemblies for each word in the lexicon;  $\text{VISUAL}$  has assemblies for each noun, as does  $\text{MOTOR}$  for each verb. This reflects that we seek to model the acquisition of highly grounded, core lexical items, and are abstracting away phonological acquisition — which is of course a highly interesting direction in its own right. These lexical items are acquired before more abstract nouns and verbs (such as *peace* and *explain*) that may require a variant of this representation scheme. We are confident that appropriate extensions of our basic model will handle abstract language — see Section 7.6 for a discussion of this and other extensions.

To summarize, a sentence  $s = W_1W_2$  of our language in the SV setting (the VS setting is analogous) is input into  $\mathcal{O}$  as follows: the corresponding assemblies in all the context areas, that is  $\text{VISUAL}[W_1]$ ,  $\text{MOTOR}[W_2]$ ,  $\text{CONTEXT}_{i[W_1]}$  and  $\text{CONTEXT}_{i[W_2]}$  fire for  $t - 1 \in [2 \times \tau]$ , while  $\text{PHON}[W_1]$  fires for  $t - 1 \in [\tau]$ , and then  $\text{PHON}[W_2]$  fires for  $t - \tau - 1 \in [\tau]$ . We will denote these

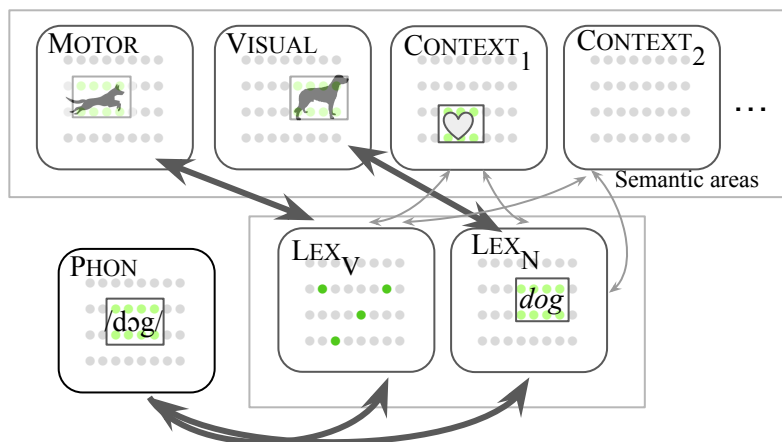


Figure 7.1: The architecture of the language organ  $\mathcal{O}$  in the NEMO model of neuronal computation. This example show the state of a trained  $\mathcal{O}$  after hearing the word *dog* in a grounded setting when the listener also sees a dog jumping (this could be part of a sentence like “the dog jumps”). The corresponding assemblies are active in VISUAL (the image of a dog) and MOTOR (the action of jumping); assemblies can also be active in  $\text{CONTEXT}_i$  areas, representing additional semantic contextual stimuli such as an emotional affect.

steps of the dynamical system by the shorthand  $\text{Feed}(s)$ .

## 7.5 The learning experiment

We first select the parameters  $n, k, p, \beta$  (which may vary across different areas);  $l$  (the lexicon size),  $\tau$  (how many times each word fires), and  $C$  (the number of extra context areas). To *train*  $\mathcal{O}$ , we generate random sentences  $s_1, s_2, \dots$  in our toy language, executing  $\text{Feed}(s_i)$  for each  $s_i$ .

Our experiments reveal that, for varying settings of the parameters (such as  $n = 10^6, k = 10^3, \beta = 0.1, l = 5, \tau = 2$ ), after some number of training sentences the model accomplishes something interesting and nontrivial, and *necessary for language acquisition*: it forms assemblies for nouns in  $\text{LEX}_N$  but not in  $\text{LEX}_V$ , assemblies for verbs in  $\text{LEX}_V$  but not in  $\text{LEX}_N$ <sup>1</sup>, and in addition, the assemblies in these areas are reliably connected to each word’s corresponding assemblies in PHON, MOTOR, and VISUAL, and also reasonably well connected to the other context areas. In other words, and in a concrete sense, the model has learned which words are nouns and verbs, and

<sup>1</sup>To see why this is highly nontrivial, the reader is reminded that this is done in the absence of knowledge of whether, in the language being learned, subject precedes verb or the other way around.

has formed correct semantic representations of each word.

We say that an experiment *succeeded* after  $m$  training sentences if we have that for each word  $W \in L$ , the resulting *synaptic weights* of  $\mathcal{O}$  satisfy properties  $P$  and  $Q$ . Property  $P$  captures a kind of *production* ability — that is, ability to go from semantic representations to phonological form, much like the mapping from lemma to lexeme in psycholinguistics; properties  $Q$  guarantee that a stable representation for each word is formed in the word’s correct area —  $\text{LEX}_N$  or  $\text{LEX}_V$  — and not in the other area.

We start by defining the  $P$  property: A noun (respectively, verb)  $W$  satisfies property  $P$  if firing  $\text{VISUAL}[W]$  (resp.  $\text{MOTOR}[W]$ ) and  $\text{CONTEXT}[i[W]]$  activates via  $\text{LEX}_N$  (resp. via  $\text{LEX}_V$ ) almost all of the representation  $\text{PHON}[W]$ ; in our tests, we define “almost” as least 75% of the cells in that assembly. We say the experiment satisfies  $P$  if every word satisfies the  $P$ .

For the  $Q$  properties, suppose  $W$  is a noun and that  $\text{PHON}[W]$  fires once. Let  $\nu$  be the resulting  $k$ -cap in  $\text{LEX}_N$ , and  $\mu$  the resulting  $k$ -cap in  $\text{LEX}_V$ . The properties  $Q_i$  are defined as follows.

1.  $Q_1$ : the synaptic input into  $\nu$  is *greater* than that into  $\mu$  by a factor of two.
2.  $Q_2$ : if we fire  $\nu$ , it activates  $\text{PHON}[W]$  and  $\text{VISUAL}[W]$ ; whereas if we fire  $\mu$ , it does not activate any of the predefined assemblies in  $\text{PHON}$  or  $\text{MOTOR}$ .
3.  $Q_3$ : if we fire  $\nu$ , it activates  $\nu$  within  $\text{LEX}_N$  itself; whereas if we fire  $\mu$ , the next  $k$ -cap in  $\text{LEX}_V$  has small overlap with  $\mu$  (less than 50%).

If  $W$  is a verb, the  $Q_i$  are defined as above but swapping noun with verb, and  $\text{MOTOR}$  with  $\text{VISUAL}$ . Intuitively, the  $Q_i$  capture that *a stable hub representation of each word has been formed in the correct part-of-speech lexical area for that word*. The experiment satisfies  $Q$  is every word satisfies the  $Q_i$ .

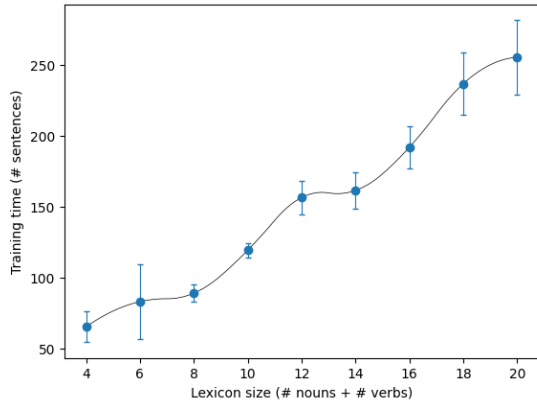
**Results.** We run our NEMO -based language organ with a variety of parameters with random sentences until success, that is, until  $P$  and  $Q$  are satisfied, and report the resulting training time. Despite representing a dynamical system of *millions of neurons and synapses*, the system

converges and yields stable representations (satisfying  $P$  and  $Q$ ) for reasonable settings of the parameters.

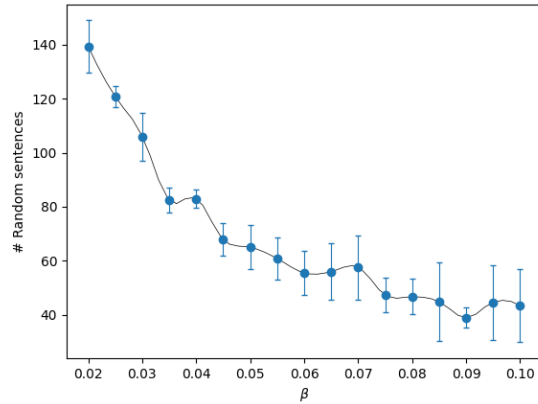
The results are summarized in Figure 7.2, where we see that the number of training sentences grows roughly linearly with the lexicon, or number of words acquired. While the number of training sentences may appear somewhat large, there are a few points to keep in mind. Our model describes the acquisition of one’s “first words”, the most contextually rich and consistent, for which 10-20 overhead sentences per word does not seem unrealistic. Furthermore, to our knowledge ours is the first simulation of a non-trivial part of language acquisition performed entirely in a bioplausible model of neurons and synapses. Nevertheless, reducing the number of training sentences is a crucial goal of this line of research: we propose a heuristic for this in the following subsection, and discuss ideas for future research in Section 7.6. We also experiment running the model with varying  $\beta$  (the plasticity parameter) revealing roughly inverse-exponential acceleration of the rate of convergence to stable representations with increasing  $\beta$ . In experiments with or without extra context areas, the training time remains roughly the same. See Figure 7.2 for details.

### 7.5.1 Individual word tutoring

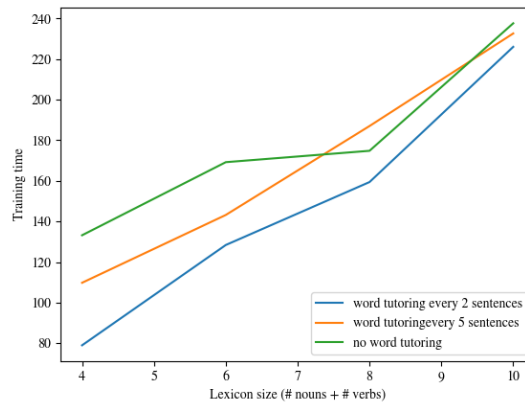
Our model is able to learn word semantics from *full* sentences, without ever being presented isolated words. While it is known that children can acquire language in this way, in our experiments the number of sentences required is rather large, and scales linearly with the size of the lexicon. An important problem for our theory is to understand how to reduce this size, especially to model later stages of acquisition, since humans acquire language from small amounts of data. We believe this is done two ways: At the early stages with *individual word tutoring*, and at later stages through *functional words* (see the next section). To test individual word tutoring, after every fixed number of sentences we randomly select a single word  $W \in L$  and fire PHON[ $W$ ] and its contextual areas for some  $\tau$  time-steps. We find that this greatly decreases the total training *time*. In particular, at early stages of acquisition, individual word tutoring reduced the training time by over 40%.



(a) Increasing lexical size



(b) Varying beta



(c) Mixing in individual word tutoring

Figure 7.2: Results of our experiments. In (a) the learning experiment of section 7.5 is performed for varying sizes of the lexicon, revealing a linear trend ( $n = 10^5$ ,  $p = 0.05$ ,  $\beta = 0.06$ ,  $k_{\text{LEX}_N} = k_{\text{LEX}_V} = 50$ ,  $k_{\text{CONTEXT}_i} = 20$ , other areas  $k = 100$ ,  $C = 20$  and  $\tau = 2$ ). In (b) the learning experiment is repeated for varying  $\beta$  and  $C = 0$ , always for a lexicon of size 4. In (c) we run learning experiments as in (a) — green — along with two variants, one in which a round of individual word tutoring is performed after every 2 random sentences (blue), and another every 5 random sentences (orange): individual word tutoring decreases training time significantly, particularly when a smaller set of words is taught at a given time. (a)-(c) were performed with both a NV and NV word orders with similar results; NV results are shown here and both are available in the supplementary data. Each experiment is repeated 5 times; means and standard deviations are reported.

## 7.6 Future Work

**Multilinguality** We believe that our model can be extended to handle *multilinguality* by adding an additional area LANG, connected into LEX<sub>N</sub> and LEX<sub>V</sub>. Like the contextual areas, LANG would have several assemblies, one for every language the multilingual child is exposed to, with strong input into LEX<sub>N</sub> and LEX<sub>V</sub>. For learning to succeed in the sense of Section ??, separate assemblies for each concept in each language must form in the lexical areas; we expect that this will require more training time — reflecting the fact that multilingual children may begin to speak later than monolingual children [123].

**Functional words and faster learning.** Functional words are words in closed lexical classes that have limited semantic content but have important syntactic roles (such as English prepositions, determinants, etc.); more broadly, functional categories include morphemes and inflectional paradigms of this type (e.g. the possessive marker “’s”, the adverbializer “-ly” and so on). Functional categories are somewhat of a paradox: cross-linguistically, children begin to accurately *produce* them much later than lexical words (verbs and nouns), but in recent decades, an explosion in language acquisition research has come to establish that young children are extremely sensitive to them, likely forming representations of them well before they can produce them, and utilizing them in many ways: to aid understanding, for learning lexical items (a word that follows “the” is likely to be a noun), and for bootstrapping syntax [124].

An important open problem is handling functional words, and, possibly, using them to accelerate word acquisition (reducing the learning times of 7.5, particularly important for modeling words with less contextual consistency). As a starting point, suppose we extend our language to have a mandatory article “a” before every noun (with no semantic content), that is, in the NV version of our language, every sentence has the form “a NOUN VERB”. *O*. Extending the model to acquire “a” (perhaps as a representing in an area for functional words FUNC) is an important goal; then, it can be used to quickly identify any following word as a noun (i.e., forming an initial representation in LEX<sub>N</sub>).

**Abstract words and contextual ambiguity.** Currently, our model of grounded context is rather simplistic: we assume only *object nouns* and *action verbs*, we have two areas that are specific to each kind of input, and several other unspecified contextual areas that fire randomly when we hear the word. Eventually, we would like to be able to handle abstract words like “disagreement” and “aspire”. Extending our model, in particular its representation of semantics, to handle such words is one of our main future directions.

**Generation and Syntax.** *Perhaps the most important direction left open by our work is syntax.* As a first step, we want the model to learn whether the toy language has NV or VN order. Concretely, this would entail the following experiment: after exposure to some number of random sentences (as in the current model), we can *generate* sentences by activating the assemblies in contextual areas corresponding to every word in the sentence, and, letting the dynamical system run, it will fire the assemblies in PHON in the correct order of the language (NV or VN). This itself is but a small piece of syntax; transitive verbs and object would be the next step, which we believe can be carried out by modest, and hardly qualitative, extensions of our setup and methods.

## 7.7 Conclusion

We have defined and implemented a dynamical system, composed of millions of simulated neurons and synapses in a realistic but tractable mathematical model of the brain, and in line with what is known about language in the brain at a high level, that is capable of learning representations of words from grounded language input. We believe this is a first and crucial step in neurally plausible modeling of the language organ and of language acquisition. We have outlined a number of future directions of research, within the reach of our approach, that are necessary for a complete theory of language in the brain.

## References

- [1] R. Axel, “Neuron Q&A”, *Neuron*, vol. 99, pp. 1110–1112, 2018.
- [2] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. J. Hudspeth, *Principles of Neural Science*. McGraw-Hill, 2013, vol. 5th Edition.
- [3] S. J. Cook *et al.*, “Whole-animal connectomes of both *caenorhabditis elegans* sexes”, *Nature*, vol. 571, pp. 63–71, 2019.
- [4] E. Jonas and K. P. Kording, “Could a neuroscientist understand a microprocessor?”, *PLOS Computational Biology*, vol. 13, no. 1, pp. 1–24, Jan. 2017.
- [5] S. Dasgupta, C. F. Stevens, and S. Navlakha, “A neural algorithm for a fundamental computing problem”, *Science*, vol. 358, pp. 793–796, 2017.
- [6] K. M. Franks, M. J. Russo, D. L. Sosulski, A. A. Mulligan, S. A. Siegelbaum, and R. Axel, “Recurrent circuitry dynamically shapes the activation of piriform cortex”, *Neuron*, vol. 72, no. 1, 2011.
- [7] M.-B. Moser, D. Rowland, and E. Moser, “Place cells, grid cells, and memory”, *Cold Spring Harbor perspectives in medicine*, vol. 5, a021808, Feb. 2015.
- [8] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Wiley, New York, 1949.
- [9] K. D. Harris, J. Csicsvari, H. Hirase, G. Dragoi, and G. Buzsáki, “Organization of cell assemblies in the hippocampus”, *Nature*, vol. 424, no. 6948, 2003.
- [10] K. D. Harris, “Neural signatures of cell assembly organization”, *Nature Reviews Neuroscience*, vol. 6, no. 5, 2005.
- [11] E. Pastalkova, V. Itskov, A. Amarasingham, and G. Buzsáki, “Internally generated cell assembly sequences in the rat hippocampus”, *Science*, vol. 321, no. 5894, 2008.
- [12] G Buzsaki, “Neural syntax: Cell assemblies, synapsembles, and readers”, *Neuron*, vol. 68, no. 3, 2010.
- [13] *Assemblies simulation*, <http://github.com/dmitropolsky/assemblies>, 2019.

- [14] D. Mitropolsky and C. Papadimitriou, *The architecture of a biologically plausible language organ*, 2023. arXiv: 2306.15364 [cs.NE].
- [15] D. C. Marr, “A theory of cerebellar cortex”, *The Journal of Physiology*, vol. 202, 1969.
- [16] D. C. Marr, “A theory for cerebral neocortex”, *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 176, pp. 161–234, 1970.
- [17] D. C. Marr, “Simple memory: A theory for archicortex.”, *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, vol. 262 841, pp. 23–81, 1971.
- [18] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities”, *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, 1982.
- [19] J. Bruck, “On the convergence properties of the hopfield model”, *Proc. IEEE*, vol. 78, pp. 1579–1585, 1990.
- [20] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, “The capacity of the hopfield associative memory”, *IEEE Trans. Inf. Theory*, vol. 33, pp. 461–482, 1987.
- [21] J. J. Hopfield and D. W. Tank, ““neural” computation of decisions in optimization problems”, *Biological Cybernetics*, vol. 52, pp. 141–152, 1985.
- [22] J. J. Hopfield and D. W. Tank, “Computing with neural circuits: A model.”, *Science*, vol. 233 4764, pp. 625–33, 1986.
- [23] M. Dabagia, D. Mitropolsky, C. Papadimitriou, and S. Vempala, *Coin-flipping in the brain: Statistical learning with neuronal assemblies*, 2024. arXiv: 2406.07715 [cs.NE].
- [24] L. Valiant, “A quantitative theory of neural computation”, *Biological cybernetics*, vol. 95, pp. 205–11, Oct. 2006.
- [25] R. Yuste, R. Cossart, and E. Yaksi, “Neuronal ensembles: Building blocks of neural circuits”, *Neuron*, vol. 112, Jan. 2024.
- [26] D. Kemmerer, D. Rudrauf, K. Manzel, and D. Tranel, “Behavioral patterns and lesion sites associated with impaired processing of lexical and conceptual knowledge of actions”, *Cortex; a journal devoted to the study of the nervous system and behavior*, vol. 48, pp. 826–48, Nov. 2012.
- [27] T. Lucas, G. McKhann, and G. Ojemann, “Functional separation of languages in the bilingual brain: A comparison of electrical stimulation language mapping in 25 bilingual patients and 117 monolingual control patients”, *Journal of neurosurgery*, vol. 101, pp. 449–57, Oct. 2004.

- [28] D. Boatman, B. Gordon, J. Hart, O. Selnes, D. Miglioretti, and F. Lenz, “Transcortical sensory aphasia: revisited and revised”, *Brain*, vol. 123, no. 8, pp. 1634–1642, Aug. 2000. eprint: <https://academic.oup.com/brain/article-pdf/123/8/1634/861114/1231634.pdf>.
- [29] D. Corina, E. Gibson, R. Martin, A. Poliakov, J. Brinkley, and G. Ojemann, “Dissociation of action and object naming: Evidence from cortical stimulation mapping”, *Human brain mapping*, vol. 24, pp. 1–10, Jan. 2005.
- [30] G. A. OJEMANN, O. CREUTZFELDT, E. LETTICH, and M. M. HAGLUND, “NEURONAL ACTIVITY IN HUMAN LATERAL TEMPORAL CORTEX RELATED TO SHORT-TERM VERBAL MEMORY, NAMING AND READING”, *Brain*, vol. 111, no. 6, pp. 1383–1403, Dec. 1988. eprint: <https://academic.oup.com/brain/article-pdf/111/6/1383/841066/111-6-1383.pdf>.
- [31] A. Chan *et al.*, “Speech-specific tuning of neurons in human superior temporal gyrus”, *Cerebral Cortex*, vol. 24, May 2014.
- [32] A. Flinker, E. Chang, N. Barbaro, M. Berger, and R. Knight, “Sub-centimeter language organization in the human temporal lobe”, *Brain and Language*, vol. 117, no. 3, pp. 103–109, 2011, First Neurobiology of Language Conference: NLC 2009.
- [33] S. Martin, J. d. R. Millan, R. Knight, and B. Pasley, “The use of intracranial recordings to decode human language: Challenges and opportunities”, *Brain and Language*, vol. 193, Jul. 2016.
- [34] M. Leonard *et al.*, “Large-scale single-neuron speech sound encoding across the depth of human cortex”, *Nature*, vol. 626, pp. 1–10, Dec. 2023.
- [35] M. Brainard and A. Doupe, “Translating birdsong: Songbirds as a model for basic and applied medical research”, *Annual review of neuroscience*, vol. 36, Jun. 2013.
- [36] C. H. Papadimitriou, S. S. Vempala, D. Mitropolsky, M. Collins, W. Maass, and L. F. Abbott, “A calculus for brain computation”, *2019 Conference on Cognitive Computational Neuroscience*, 2019.
- [37] C. H. Papadimitriou, S. S. Vempala, D. Mitropolsky, M. Collins, and W. Maass, “Brain computation by assemblies of neurons”, *Proceedings of the National Academy of Sciences (PNAS)*, vol. 117, no. 25, pp. 14 464–14 472, 2020. eprint: <https://www.pnas.org/content/117/25/14464.full.pdf>.
- [38] P. Erdős and A. Renyi, “On the evolution of random graphs”, *Publ. Math. Inst. Hungary. Acad. Sci.*, vol. 5, pp. 17–61, 1960.

- [39] S. Song, P. J. Sjöström, M. Reigl, S. Nelson, and D. B. Chklovskii, “Highly nonrandom features of synaptic connectivity in local cortical circuits”, *PLoS Biology*, vol. 3, no. 3, 2005.
- [40] S. Guzman, A. J. Schlögl, M. Frotscher, and P. Jonas, “Synaptic mechanisms of pattern completion in the hippocampal CA3 network”, *Science*, vol. 353, no. 6304, 2016.
- [41] R. Q. Quiroga, “Neuronal codes for visual perception and memory”, *Neuropsychologia*, vol. 83, 2016.
- [42] U. Feige, D. Peleg, and G. Kortsarz, “The dense k-subgraph problem”, *Algorithmica*, vol. 29, no. 3, 2001.
- [43] M. J. Ison, R. Q. Quiroga, and I. Fried, “Rapid encoding of new memories by individual neurons in the human brain”, *Neuron*, vol. 87, no. 1, 2015.
- [44] J.-e. K. Miller, I. Ayzenshtat, L. Carrillo-Reid, and R. Yuste, “Visual stimuli recruit intrinsically generated cortical ensembles”, *Proceedings of the National Academy of Sciences*, vol. 111, no. 38, E4053–E4061, 2014.
- [45] R. Legenstein, C. H. Papadimitriou, S. Vempala, and W. Maass, “Assembly pointers for variable binding in networks of spiking neurons”, *arXiv preprint arXiv:1611.03698*, 2016.
- [46] R. C. Berwick and N. Chomsky, *Why only us: Language and evolution*. MIT Press, 2016.
- [47] M. D. Hauser, N. Chomsky, and W. T. Fitch, “The faculty of language: What is it, who has it, and how did it evolve?”, *science*, vol. 298, no. 5598, 2002.
- [48] E. Zaccarella, L. Meyer, M. Makuuchi, and A. D. Friederici, “Building by syntax: The neural basis of minimal linguistic structures”, *Cerebral Cortex*, vol. 27, no. 1, 2017.
- [49] D. Mitropolsky, M. J. Collins, and C. H. Papadimitriou, “A Biologically Plausible Parser”, in *Transactions of the Association for Computational Linguistics*, arXiv: 2108.02189, Aug. 2021.
- [50] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Wiley, New York, 1949.
- [51] K. D. Harris, “Neural signatures of cell assembly organization”, *Nature Reviews Neuroscience*, vol. 6, no. 5, p. 399, 2005.
- [52] H. Eichenbaum, “Barlow versus hebb: When is it time to abandon the notion of feature detectors and adopt the cell assembly as the unit of cognition?”, *Neuroscience letters*, vol. 680, pp. 88–93, 2018.

- [53] G. Buzsaki, *The brain from inside out*. Oxford University Press, USA, 2019.
- [54] F. Keller, “Cognitively plausible models of human language processing”, in *Proceedings of the ACL 2010 Conference Short Papers*, Uppsala, Sweden: Association for Computational Linguistics, Jul. 2010, pp. 60–67.
- [55] D. Jurafsky, “A probabilistic model of lexical and syntactic access and disambiguation”, *Cognitive Science*, vol. 20, no. 2, pp. 137–194, 1996. eprint: [https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog2002\\_1](https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog2002_1).
- [56] J. Hale, “A probabilistic Earley parser as a psycholinguistic model”, in *Second Meeting of the North American Chapter of the Association for Computational Linguistics*, 2001.
- [57] R. Levy, “Expectation-based syntactic comprehension”, *Cognition*, vol. 106, no. 3, pp. 1126–1177, 2008.
- [58] V. Demberg and F. Keller, “Data from eye-tracking corpora as evidence for theories of syntactic processing complexity”, *Cognition*, vol. 109, no. 2, pp. 193–210, 2008.
- [59] V. Demberg and F. Keller, “A psycholinguistically motivated version of TAG”, in *Proceedings of the Ninth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+9)*, Tübingen, Germany: Association for Computational Linguistics, Jun. 2008, pp. 25–32.
- [60] V. Demberg and F. Keller, “A computational model of prediction in human parsing: Unifying locality and surprisal effects”, Jan. 2009, pp. 1888–1893.
- [61] V. Demberg, F. Keller, and A. Koller, “Incremental, predictive parsing with psycholinguistically motivated Tree-Adjoining Grammar”, *Computational Linguistics*, vol. 39, no. 4, pp. 1025–1066, 2013.
- [62] R. L. Lewis and S. Vasishth, “An activation-based model of sentence processing as skilled memory retrieval”, *Cognitive Science*, vol. 29, no. 3, pp. 375–419, 2005. eprint: [https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog0000\\_25](https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog0000_25).
- [63] J. Hale, C. Dyer, A. Kuncoro, and J. Brennan, “Finding syntax in human encephalography with beam search”, in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 2727–2736.
- [64] D. Merx and S. L. Frank, *Comparing transformers and rnns on predicting human sentence processing data*, 2020. arXiv: 2005.09471 [cs.CL].

- [65] S. L. Frank and J. Hoeks, *The interaction between structure and meaning in sentence comprehension: Recurrent neural networks and reading times*, 2019.
- [66] M. Schrimpf *et al.*, “The neural architecture of language: Integrative reverse-engineering converges on a model for predictive processing”, *bioRxiv*, 2020. eprint: <https://www.biorxiv.org/content/early/2020/10/09/2020.06.26.174482.full.pdf>.
- [67] X. Hinaut and P. F. Dominey, “Real-time parallel processing of grammatical structure in the fronto-striatal system: A recurrent network simulation study using reservoir computing”, *PLOS ONE*, vol. 8, no. 2, pp. 1–18, Feb. 2013.
- [68] S. Frank, P. Monaghan, and C. Tsoukala, “Neural network models of language acquisition and processing”, in *P. Hagoort (Ed.), Human Language: from Genes and Brains to Behavior*, MIT Press, 2019, pp. 277–291.
- [69] M. R. Schomers, M. Garagnani, and F. Pulvermüller, “Neurocomputational consequences of evolutionary connectivity changes in perisylvian language cortex”, *Journal of Neuroscience*, vol. 37, no. 11, pp. 3045–3055, 2017. eprint: <https://www.jneurosci.org/content/37/11/3045.full.pdf>.
- [70] A. D. Friederici, *Language in our brain: The origins of a uniquely human capacity*. MIT Press, 2017.
- [71] S. M. Frankland and J. D. Greene, “An architecture for encoding sentence meaning in left mid-superior temporal cortex”, *Proceedings of the National Academy of Sciences*, vol. 112, no. 37, pp. 11 732–11 737, 2015.
- [72] I. A. Blank and E. Fedorenko, “No evidence for differences among language regions in their temporal receptive windows”, *NeuroImage*, vol. 219, p. 116 925, 2020.
- [73] E. Fedorenko and S. L. Thompson-Schill, “Reworking the language network”, *Trends in Cognitive Sciences*, vol. 18, no. 3, pp. 120–126, 2014.
- [74] L. Pyllkkänen, “Neural basis of basic composition: What we have learned from the red-boat studies and their extensions”, *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 375, no. 1791, Feb. 2020, Publisher Copyright: © 2019 The Author(s) Published by the Royal Society.
- [75] E. Zaccarella and A. D. Friederici, “Merge in the human brain: A sub-region based functional investigation in the left pars opercularis”, *Frontiers in Psychology*, vol. 6, p. 1818, 2015.

- [76] N. Ding, L. Melloni, H. Zhang, X. Tian, and D. Poeppel, “Cortical tracking of hierarchical linguistic structures in connected speech”, *Nature neuroscience*, vol. 19, no. 1, p. 158, 2016.
- [77] D. Grodner and E. Gibson, “Consequences of the serial nature of linguistic input for sentential complexity”, *Cognitive science*, vol. 29, pp. 261–90, Mar. 2005.
- [78] F. Karlsson, “Constraints on multiple center-embedding of clauses”, *Journal of Linguistics*, vol. 43, no. 2, 365–392, 2007.
- [79] L. Wang, M. Bastiaansen, Y. Yang, and P. Hagoort, “Information structure influences depth of syntactic processing: Event-related potential evidence for the chomsky illusion”, *PLoS one*, vol. 7, e47917, Oct. 2012.
- [80] E. T. Rolls and G. Deco, “Networks for memory, perception, and decision-making, and beyond to how the syntax for language might be implemented in the brain”, *Brain Research*, vol. 1621, pp. 316–334, 2015, *Brain and Memory: Old Arguments and New Perspectives*.
- [81] D. Mitropolsky, A. Ejaz, M. Shi, C. Papadimitriou, and M. Yannakakis, “Center-embedding and constituency in the brain and a new characterization of context-free languages”, in *Proceedings of the 3rd Natural Logic Meets Machine Learning Workshop (NALOMA III)*, Galway, Ireland: Association for Computational Linguistics, Aug. 2022, pp. 26–37.
- [82] F. d’Amore, D. Mitropolsky, P. Crescenzi, E. Natale, and C. H. Papadimitriou, “Planning with biological neurons and synapses”, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 1, pp. 21–28, 2022.
- [83] N. Gupta and D. S. Nau, “Complexity results for blocks-world planning”, in *Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 2*, ser. AAAI’91, Anaheim, California: AAAI Press, 1991, 629–633, ISBN: 0262510596.
- [84] J. Slaney and S. Thiébaux, “Blocks world revisited”, *Artificial Intelligence*, vol. 125, no. 1, pp. 119–153, 2001.
- [85] N. Gupta and D. S. Nau, “On the complexity of blocks-world planning”, *Artificial Intelligence*, vol. 56, no. 2, pp. 223–254, 1992.
- [86] C. H. Papadimitriou, S. S. Vempala, D. Mitropolsky, M. Collins, and W. Maass, “Brain computation by assemblies of neurons”, *Proceedings of the National Academy of Sciences*, vol. 117, no. 25, pp. 14 464–14 472, 2020. eprint: <https://www.pnas.org/content/117/25/14464.full.pdf>.
- [87] D. Mitropolsky, M. J. Collins, and C. H. Papadimitriou, “A Biologically Plausible Parser”, in *Transactions of the Association for Computational Linguistics*, arXiv: 2108.02189, Aug. 2021.

- [88] T. Winograd, “Procedures as a representation for data in a computer program for understanding natural language”, Massachusetts Inst Of Tech Cambridge Project Mac, Tech. Rep., 1971.
- [89] H. Dong, J. Mao, T. Lin, C. Wang, L. Li, and D. Zhou, “Neural logic machines”, in *International Conference on Learning Representations*, 2019.
- [90] A. Graves, G. Wayne, and I. Danihelka, *Neural turing machines*, 2014. arXiv: 1410.5401 [cs.NE].
- [91] R. Doursat, “Bridging the Mind-Brain Gap by Morphogenetic "Neuron Flocking": The Dynamic Self-Organization of Neural Activity into Mental Shapes”, in *AAAI Fall Symposia*, 2013.
- [92] M. Chady, “Modelling higher cognitive functions with hebbian cell assemblies”, in *Proceedings of AAAI/IAAI 1999, July 18-22, 1999, Orlando, Florida, USA*, J. Hendler and D. Subramanian, Eds., AAAI Press, 1999, p. 943.
- [93] W. G. Kennedy and J. G. Trafton, “Long-term symbolic learning in soar and act-r”, in *Proceedings of the Seventh International Conference on Cognitive Modeling*, Trieste, Italy, 2006, 166–171.
- [94] U. Kurup, “Design and use of a bimodal cognitive architecture for diagrammatic reasoning and cognitive modeling”, Ph.D. diss. Graduate School of the Ohio State University, 2008.
- [95] A. I. Panov, “Behavior planning of intelligent agent with sign world model”, *Biologically Inspired Cognitive Architectures*, vol. 19, pp. 21–31, 2017.
- [96] M. Hayashi, “Stacking of blocks by chimpanzees: Developmental processes and physical understanding”, *Animal Cognition*, vol. 10, pp. 89–103, 2007.
- [97] M. Tian, T. Luo, and H. Cheung, “The development and measurement of block construction in early childhood: A review”, *Journal of Psychoeducational Assessment*, vol. 38, no. 6, pp. 767–782, 2020.
- [98] E. Rueckert, D. Kappel, D. Tanneberg, D. Pecevski, and J. Peters, “Recurrent spiking networks solve planning tasks”, *Scientific Reports*, vol. 6, 2016.
- [99] R. Basanisi, A. Brovelli, E. Cartoni, and G. Baldassarre, “A generative spiking neural-network model of goal-directed behaviour and one-step planning”, *PLOS Computational Biology*, vol. 16, no. 12, pp. 1–32, Dec. 2020.
- [100] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing”, *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

- [101] V. Koeman, *The blocks world*, <https://github.com/eishub/blocksworld#readme>, [Online; last access 08-September-2021], 2020.
- [102] jBrain, <https://github.com/piluc/jBrain>, version 0.1, Dec. 2021.
- [103] F. d’Amore, D. Mitropolsky, P. Crescenzi, E. Natale, and C. H. Papadimitriou, *Planning with Biological Neurons and Synapses*, <https://hal.archives-ouvertes.fr/hal-03479582>, Research Report, Dec. 2021.
- [104] D. E. Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988, ISBN: 093461394X.
- [105] M. Dabagia, S. S. Vempala, and C. Papadimitriou, “Assemblies of neurons learn to classify well-separated distributions”, in *Proceedings of Thirty Fifth Conference on Learning Theory*, P.-L. Loh and M. Raginsky, Eds., ser. Proceedings of Machine Learning Research, vol. 178, PMLR, 2022, pp. 3685–3717.
- [106] D. Kemmerer, *Cognitive Neuroscience of Language*. Psychology Press, 2015, ISBN: 9781848726208.
- [107] J. Brennan, *Language and the Brain: A Slim Guide to Neurolinguistics* (Oxford linguistics). Oxford University Press, 2022, ISBN: 9780198814757.
- [108] P Indefrey and W. Levelt, “The spatial and temporal signatures of word production components”, *Cognition*, vol. 92, no. 1, pp. 101–144, 2004, Towards a New Functional Anatomy of Language.
- [109] A. Martin, “The representation of object concepts in the brain”, *Annual Review of Psychology*, vol. 58, no. 1, pp. 25–45, 2007, PMID: 16968210. eprint: <https://doi.org/10.1146/annurev.psych.57.102904.190143>.
- [110] M. Kiefer and F. Pulvermüller, “Conceptual representations in mind and brain: Theoretical developments, current evidence and future directions”, *Cortex*, vol. 48, no. 7, pp. 805–825, 2012, Language and the Motor System.
- [111] S. Popham *et al.*, “Visual and linguistic semantic representations are aligned at the border of human visual cortex”, *Nature Neuroscience*, vol. 24, pp. 1628–1636, Nov. 2021.
- [112] S. Gennari, “Representing motion in language comprehension: Lessons from neuroimaging”, *Language and Linguistics Compass*, vol. 6, pp. 67–84, Feb. 2012.
- [113] C. Watson, E. Cardillo, G. Ianni, and A. Chatterjee, “Action concepts in the brain: An activation likelihood estimation meta-analysis”, *Journal of cognitive neuroscience*, vol. 25, Apr. 2013.

- [114] D. Kemmerer and J. Gonzalez-Castillo, “The two-level theory of verb meaning: An approach to integrating the semantics of action with the mirror neuron system”, *Brain and Language*, vol. 112, pp. 54–76, Jan. 2010.
- [115] L. Fernandino and M. Iacoboni, “Are cortical motor maps based on body parts or coordinated actions? implications for embodied semantics”, *Brain and Language*, vol. 112, no. 1, pp. 44–53, 2010, Mirror Neurons: Prospects and Problems for the Neurobiology of Language.
- [116] S. Mätzig, J. Druks, J. Masterson, and G. Vigliocco, “Noun and verb differences in picture naming: Past studies and new evidence”, *Cortex*, vol. 45, no. 6, pp. 738–758, 2009.
- [117] G. Vigliocco, D. P. Vinson, J. Druks, H. Barber, and S. F. Cappa, “Nouns and verbs in the brain: A review of behavioural, electrophysiological, neuropsychological and imaging studies”, *Neuroscience & Biobehavioral Reviews*, vol. 35, no. 3, pp. 407–426, 2011.
- [118] G. Hickok, B. Buchsbaum, C. Humphries, and T. Muftuler, “Auditory–motor interaction revealed by fmri: Speech, music, and working memory in area spt”, *J. Cognitive Neuroscience*, vol. 15, no. 5, 673–682, 2003.
- [119] K. Okada and G. Hickok, “Left posterior auditory-related cortices participate both in speech perception and speech production: Neural overlap revealed by fmri”, *Brain and Language*, vol. 98, no. 1, pp. 112–117, 2006.
- [120] W. J. M. Levelt, A. Roelofs, and A. S. Meyer, “A theory of lexical access in speech production”, *Behavioral and Brain Sciences*, vol. 22, no. 1, 1–38, 1999.
- [121] G. Hickok and D. Poeppel, “The cortical organization of speech processing”, *Nature reviews. Neuroscience*, vol. 8, pp. 393–402, Jun. 2007.
- [122] M. Ralph, “Neurocognitive insights on conceptual knowledge and its breakdown”, *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, vol. 369, p. 20120392, Jan. 2014.
- [123] H. Hambly, Y. Wren, S. McLeod, and S. Roulstone, “The influence of bilingualism on speech production: A systematic review”, *International Journal of Language & Communication Disorders*, vol. 48, no. 1, pp. 1–24, 2013. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1460-6984.2012.00178.x>.
- [124] C. D. Dye, Y. Kedar, and B. Lust, “From lexical to functional categories: New foundations for the study of language development”, *First Language*, vol. 39, pp. 32–9, 2018.

## **Appendix A: Experimental Equipment**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

## **Appendix B: Data Processing**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.