Abstract

Title of dissertation: A Neurocomputational Model of Grounded Language Comprehension and Production at the Sentence Level

Derek Monner, Doctor of Philosophy, 2011

Dissertation directed by: Professor James A. Reggia
Department of Computer Science

While symbolic and statistical approaches to natural language processing have become undeniably impressive in recent years, such systems still display a tendency to make errors that are inscrutable to human onlookers. This disconnect with human processing may stem from the vast differences in the substrates that underly natural language processing in artificial systems versus biological systems.

To create a more relatable system, this dissertation turns to the more biologically inspired substrate of neural networks, describing the design and implementation of a model that learns to comprehend and produce language at the sentence level. The model's task is to ground simulated speech streams, representing a simple subset of English, in terms of a virtual environment. The model learns to understand and answer full-sentence questions about the environment by mimicking the speech stream of another speaker, much as a human language learner would. It is the only known neural model to date that can learn to map natural language questions to full-sentence natural language answers, where both question and answer are represented sublexically as phoneme sequences.

The model addresses important points for which most other models, neural and otherwise, fail to account. First, the model learns to ground its linguistic knowledge using human-like sensory representations, gaining language understanding at a deeper level than that of syntactic structure. Second, analysis provides evidence that the model learns combinatorial internal representations, thus gaining the compositionality of symbolic approaches to cognition, which is vital for computationally efficient encoding and decoding of meaning. The model does this while retaining the fully distributed representations characteristic of neural networks, providing the resistance to damage and graceful degradation that are generally lacking in symbolic and statistical approaches. Finally, the model learns via direct imitation of another speaker, allowing it to emulate human processing with greater fidelity, thus increasing the relatability of its behavior.

Along the way, this dissertation develops a novel training algorithm that, for the first time, requires only local computations to train arbitrary second-order recurrent neural networks. This algorithm is evaluated on its overall efficacy, biological feasibility, and ability to reproduce peculiarities of human learning such as age-correlated effects in second language acquisition.

A Neurocomputational Model of Grounded Language Comprehension and Production at the Sentence Level

by
Derek Monner

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

Advisory Committee:

Professor James A. Reggia	Committee Chair
Professor William J. Idsardi	Dean's Representative
Professor Dana S. Nau	Committee Member
Professor Amy Weinberg	Committee Member
Asst Professor William Rand	Committee Member

To my parents,
for the constant love and support,
car advice, and careful editing.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Rationale

Symbolic and statistical approaches currently dominate the field of natural language processing. Systems based on these approaches have made exciting progress recently, utilizing ever-larger corpora and ever-more computational resources to attempt to automatically make sense of the wealth of speech and text data that exists in the world, digitally and otherwise. One recent and well-known example is Watson, IBM's question-answering system designed to play the popular quiz show *Jeopardy!* (Ferrucci et al., 2010). Watson famously bested some of the most skilled human competitors in a series of televised games in February 2011. While question-answering systems like Watson are undoubtedly impressive, they still frequently make mistakes, and these errors are often baffling and inscrutable to onlookers. For example, during one of Watson's television appearances, the contestants received the following clue: "*The New Yorker*'s 1959 review of this said in its brevity and clarity, it is 'unlike most such manuals, a book as well as a tool.'" Whereas a correct answer might have been "What is *Elements of Style*?" Watson buzzed in first, responding "Who is Dorothy Parker?" A human competitor would never give an answer like this, and indeed, the looks of bewilderment on the faces of the competitors and the host were proof that they could not follow whatever chain of logic had led Watson to generate its reply. This anecdote, taken together with many other instances of incomprehensible answers, suggests that the strategies that systems like Watson use are far different from those that humans employ.

Watson and its brethren generally base their skills on expansive corpora that are preprocessed and compressed into an easily searchable knowledge base. Human memory seems, on the surface, to be built on a similar idea, with learners building their own content-addressable knowledge bases populated by their life experiences. So why, then, do artificial systems make mistakes no human ever would—ones that humans cannot even fathom? This dissociation may stem from the fact that artificial and natural systems are built upon vastly different substrates. Most symbolic and statistical approaches to language processing build upon the conceptual substrate of modern computing machinery, in which the elementary operations include storing information and retrieving perfect copies of said information. The human mind, on the other hand, is built on the brain, which admits much fuzzier conceptions of storage and retrieval. The brain's stored memories are routinely split, merged, altered, lost altogether, or even spuriously created. While these may seem to be dire weaknesses of the brain's approach to memory, the inherently distributed, associative nature of this substrate may confer benefits that cannot be achieved through other means. If the goal of natural language processing is to create systems that function like humans, and to which humans can readily relate, this warrants the investigation of systems that more closely identify with what is known about the human mind and brain.

The scientific community studies the human brain and mind at many different levels of abstraction—from individual ion channels of neurons to high-level cognitive processes, to the actions we take in the world. All of these types of studies have their place, and cognitive scientists must strive to eventually unite them vertically, with the lower-level explanations giving rise to the next higher level, from voltage gradients and chemical messengers all the way up to consciousness, intelligence, and behavior. This dissertation focuses mainly on intermediate to high levels within this hierarchy. Specifically, the research presented here is concerned with the dynamics necessary to turn a large collection of autonomous processors—neurons—into a cognitive machine, capable of understanding and acting in the wider world. That is the overarching theme, at least. At a more practical level, this dissertation develops a model, based on a brain-like substrate, that attempts to implement those parts of a mind that learn to understand language, relate it to the world, and produce it. The hypothesis is that this level of fidelity to the human substrate of language processing will enable more relatable and human-like models of language processing.

Understanding language, and how humans come to acquire and use it, has been a unifying theme in the cognitive sciences, and is actively studied at many levels in many different disciplines. Turing (1950) chose language as the basis for his famous test for recognizing an artificial intelligence, due in part to the fact that we humans use the same conversational criteria as a part of our everyday judgments about the natural intelligence of our peers. Though not everyone agrees that language ability is either a necessary or sufficient condition for intelligence (e.g., Searle, 1980), language is widely viewed as one of the faculties that sets human intelligence beyond others we have so far encountered.

Theories of human language processing have been the subject of considerable scientific work, as has the task of engineering an artificial system capable of such processing. Though such work spans the scale of possible abstraction levels, a great deal of it occurs as high-level psycholinguistic theories describing how a cognitive-level language system could cause observed behavioral phenomena. This research is often supplemented by work that begins at the lower neural level and focuses on directly implementing a specific cognitive-level theory. Much less work spans the two levels, mapping the neural level directly to behavior. This dissertation focuses on this last category to discover what kinds of cognitive-level systems emerge when a dynamical neural system learns to perform language tasks. Reasons for studying language learning, and cognition in general, in this level-spanning way include:

1. **Observability** of the building blocks. Neurons and neural networks in human brains are becoming ever more observable as the pace of

innovation in neuroimaging accelerates and its spatial and temporal resolutions increase. The goal is to understand how interconnected networks of simple processing elements can create and maintain the rich cognitive world that humans possess in a way that will, in the near future, be directly verifiable.

2. **Learnability** of cognitive systems. Humans are not born with a fully developed cognitive system, yet cognitive theories tend to focus on the function of normal, or occasionally, impaired adult minds. Much psychological and psycholinguistic research is concerned with how children develop, but the explanations these studies provide are usually couched in cognitive terms as well. While these are useful ways to approach the problem, they do not often address the question of what these cognitive systems are made of and how they come into being. Studying the level below cognition provides the opportunity to observe how cognitive systems emerge from learning, what forms they take under varying conditions, and what constraints the neural substrate places upon their formation.

3. **Functionality** and robustness to damage. Neural network modeling provides a natural way to implement a theoretical system and compare its functionality to that of a human. Though many theoretical systems can be simulated directly at the cognitive level, such systems are often brittle in ways that humans are not. Put another way, it is often

difficult to determine how one could damage a cognitive-level system to produce observed human deficits. In a neural system, on the other hand, the question is reduced to where to place the lesion, after which behavior can be directly observed and compared to empirical human data.

4. **Referentiality** of cognitive states to the world. Neural systems can readily receive input from the world in the same ways that humans do: through a grid of retinal cell activities for vision, a distribution of frequency-based cochlear cell activations for audition, and so on. Such a system can also be designed to act on the world via motor controls that are similar to a human's, such as a simulated vocal tract that can be manipulated to produce speech. The experiences of such a system would parallel human experiences to a substantial degree. This similarity has the potential to greatly aid human understanding of the system's cognitive functions.

A model that exemplifies the preceding four qualities was termed the Wernicke-Lichtheim-Geschwind (WLG) model of word and picture learning (Weems and Reggia, 2006), as its neural structure was inspired by the work begun by these three scientists. The WLG model implemented a neural network approximating the structure of the classical model of language's neurobiological basis, centered upon the eponymous brain regions studied by Broca (1861) and Wernicke (1874) and including the parietal association areas identified by Geschwind (1965). The model received auditory and visual inputs similar to those of human sensation—in the form of auditory phonetic features and grid-based black-and-white images—and produced verbal output by generating a series of motor commands for a simulated vocal tract. Weems and Reggia trained the network to repeat words—that is, produce a sequence of motor actions corresponding to a heard sequence of sounds—and to recognize and name pictures, which it successfully learned to do for approximately 50 word/picture pairs. Following training, Weems and Reggia lesioned the model in various ways corresponding to the classic aphasia syndromes (Lichtheim, 1885) and measured the model's behavior on recognition, repetition, and naming tasks. Remarkably, the lesioned models displayed the same types of deficits as humans with similar lesions.

The WLG model satisfied all four of the preceding criteria: It was built of neural-level building blocks in a configuration suggested by a long history of aphasia literature; it learned to produce interesting behavior despite having no such functionality built in; it responded to damage in ways similar to humans; and it utilized human-like input and output representations. While it is a convincing first step towards a functional model of the brain's language architecture, it needs to be improved and extended before it can be fully assessed. The model's foremost limitation is that it only deals with single words and objects. Even as a model of aphasia, it is incomplete because its inability to understand or produce sentence-level speech puts many relevant diagnostic metrics, such as grammaticality and fluency, out of reach.

The research described in this dissertation began as an attempt to build an updated version of the WLG model that would possess sentence-level language abilities. Such a model would have to address many issues that were not present in the design of the first model. For example, a sentence-level language ability requires a strong generalization capability, since language users frequently encounter novel sentences, yet have little trouble interpreting them. The word-level WLG model, on the other hand, had no need of generalization, being content to memorize its input/output patterns. Another issue is more practical: Sentence-level language requires the ability to retain long sequences of inputs and produce long sequences of outputs—a task at which neural networks have not traditionally excelled. Thus, the first challenge to meet in building such a model was to evaluate the capabilities of various types of neural networks in the domain of processing and producing temporal sequences.

Initial investigations involved self-organizing maps (SOMs; Kohonen, 1990), which are desirable due to their ability to organize incoming input without supervision. Of particular interest were varieties of SOMs capable of sustaining multiple simultaneous local winners (Schulz and Reggia, 2004); these types of networks have been shown to readily produce units with feature-detection properties mimicking those found in the visual system (Schulz and Reggia, 2005; von der Malsburg, 1973). This type of SOM had been used in the original WLG model to aggregate temporal sequences of auditory and visual input.

Application of multi-winner SOMs to temporal sequences of sentence-level speech data showed the creation of phoneme- and word-detector units as one might expect to find in the human auditory system (Monner and Reggia, 2009). However, SOMs are designed to map their inputs to the most similar group exemplar previously seen. This property is perfect for mapping fuzzy speech input to phonemes and words, or for cleaning up a noisy visual image of a known object. However, this tendency becomes counterproductive as part of a system that needs to systematically generalize to combinatorial input.

An initial investigation subjected SOM-based networks, in either single or multi-layered configurations, to sentences drawn from a limited grammar, presented temporally as sequences of phonemes. The SOM would learn to produce a single spatial representation of a heard sentence, which was then interpreted by a separate readout network—a simple feed-forward network trained with back-propagation. The SOMs readily learned to recognize familiar sentences but were utterly unable to generalize to new ones. In essence, a trained SOM would incorrectly attempt to interpret a novel sentence as a different but similar sentence with which it was already familiar. The method of SOM training was directly responsible for this preference for the familiar, as revealed by an experiment in which the readout network tried to interpret sentences passed through an untrained SOM, only to find that its ability to generalize to novel sentences had improved dramatically. This made it clear that the SOM approaches in question are not appropriate for tasks that require extensive generalization to new inputs.

The search for a more useful set of neural network methods warranted brief experiments with echo state networks (Jaeger, 2001, 2002; Prokhorov, 2005), which, with their untrained self-recurrent reservoirs of neurons and simple readout networks, superficially resembled the untrained SOMs that had recently yielded the best results to date. Unfortunately, the experiments with echo state networks did not produce significantly better results. This fact, combined with concerns about the biological feasibility of large collections of nonadaptive neurons, indicated that echo state networks were not the method of choice for the task.

At this point, the search expanded to neural network methods that have been traditionally classified as **supervised learning**, meaning that the network relies on an external teacher to provide a gold-standard signal that should be imitated. The most prevalent of these fall under the umbrella of gradient descent methods, where the training procedure attempts to minimize an error function—defined as the difference between the network output and the teaching signal—using an inverse hill-climbing approach involving the gradient, or derivative, of the error function.

Most critics of gradient descent methods begin from a cognitive plausibility standpoint, citing the rarity or absence during human learning of a teacher who can give reliable, direct feedback. Such critics insist on **unsupervised learning** methods, of which the SOMs from earlier are an example, or **reinforcement learning**, where detailed error feedback is replaced with a coarse-grained reward signal, leaving the learner with a difficult credit-assignment problem. However, there are situations where gradient descent training can make use of teaching signals that are inherent in the input, thus getting around the lack of an explicit teacher. Elman (1993), for example, used direct error feedback to train simple recurrent networks (SRNs) to process an input sequence and, at each point, predict which input would come next. Such an approach can hardly be called supervised in the traditional sense, since the teaching signal can be internally generated from observable input. One can think of this approach as **self-supervised learning**, and the language learning problem is indeed one where self-supervised learning is feasible. A learner need only observe the linguistic interactions of others and attempt to emulate them to generate teaching signals for gradient descent training. This idea is discussed in the context of the models in Chapters 3 and 5.

Supervised gradient descent methods are also often criticized in terms of biological plausibility. In particular, the popular method of error back-propagation (Rumelhart et al., 1986) was recognized by its own authors as particularly implausible for biological neurons because no signal is known by which neurons can transmit error information backward through the network. However, a recent discovery by Xie and Seung (2003) demonstrates the near equivalence of weight changes due to back-propagation and those produced by contrastive Hebbian learning (CHL; Movellan, 1990). By this logic, back-propagation can be seen as a computationally expedient form of Hebbian learning, which is widely believed to occur in biological neurons and is used in such biologically motivated systems as Leabra (O'Reilly, 2001; O'Reilly and Frank, 2006). So while the precise method of weight-change calculation may have no known biological basis, gradient descent training methods powered by back-propagating error retain their biological plausibility, at least in terms of the results they generate.

With gradient descent methods on the table, the investigation shifted course to include training an SRN-based model on the language tasks. The results, however, were largely disappointing. The network required multiple hidden layers to efficiently encode multiple levels of linguistic representation, from phonemes and morphemes up to words, phrases, and finally entire sentences. Since the error signal in traditional back-propagation training degrades as it passes backward through the network, training these deep networks proved to be difficult. Additionally, with input sequences routinely reaching lengths of 40 or more for individual trials, the network had trouble maintaining enough of the sequence in working memory to generate a response at the end of the trial.

Fortunately, others had already discovered solutions to these problems. Hochreiter and Schmidhuber (1997) developed a variant of the SRN, called the long short-term memory (LSTM), to address both issues outlined above. This type of network has a special architecture designed to allow information to persist unperturbed in the network across long time lags. The network is still trained with gradient descent, but individual units function in such a way as to prevent the degradation of the error signal when propagated backward through space or time. Initial tests showed that this type of network, when faced with a task involving long temporal sequences, learns and generalizes drastically better than all previous methods encountered.

The original LSTM training algorithm and its subsequent variations (Gers and Cummins, 2000; Gers and Schmidhuber, 2001), however, were not designed to train networks with more than a single hidden layer. To train these types of networks, researchers had fallen back to less biologically plausible methods, such as back-propagation through time (BPTT; Werbos, 1990) and real-time recurrent learning (RTRL; Williams and Zipser, 1989). All of these methods perform computations that are nonlocal in space, time, or both, rendering them unlike both the original LSTM training algorithm and the current neuroscientific understanding of biological neural computation. Most neural networks used in artificial intelligence research already eschew close neurobiological ties, substituting highly idealized units and connections for real neurons and synapses. However, the locality constraints in real neural networks rank high in the list of properties that make them interesting subjects of study. Training methods that discard them are likely to be impossible to implement in brain-like networks, at least given current neurobiological understanding. Furthermore, it is easy to imagine such training methods, by virtue of their nonlocality, giving rise to different types of learned representations that are less distributed and less competitive than those produced by their localized counterparts.

In light of these objections, the existing methods of training deep LSTM networks do not capture essential aspects of the human neural substrate and are thus unsatisfactory for the task at hand. Replacing these is a new version of the LSTM training algorithm, appropriately generalized to handle networks with multiple hidden layers in series. This novel algorithm, called generalized long short-term memory (LSTM-g), provides a straightforward training method that remains brain-like in terms of spatial and temporal locality and is the subject of Chapter 2.

After initial tests confirmed LSTM-g's efficacy at training networks to learn speech-like data, the investigation turned to the extent to which LSTM-g learning resembled certain aspects of human learning. The importance of human-like learning should be readily apparent when considering that the ultimate goal is to use LSTM-g to create language processing models to which humans can more easily relate. A particularly interesting metric that can be used to measure the algorithm's fidelity to human learning is its ability to reproduce high-level human-like learning artifacts. One such artifact is an age-correlated decline in the ability of human learners to acquire a second language: Those who start later in life have poorer ultimate attainment of many aspects of the language than do native speakers. Chapter 3, after covering the relevant linguistic and computational background, describes two neural network models, trained by LSTM-g, that examine the influence of first-language entrenchment and memory development on language learning. The **article-prediction model** does this by learning a gender assignment task in two languages, while the **phoneme-prediction model** learns a much more involved gender agreement task. Though these models are limited in many ways when compared to a human language learner, they both independently reproduce patterns of language aptitude that strongly suggest that they successfully capture aspects of the human language learning process that would be difficult to observe in higher-level models.

The remaining chapters of this dissertation deal with the development of a detailed neural model that learns to understand and produce sentence-level language, represented and processed as individual speech sounds. Chapter 4 describes how an early version of the model, the **grounded-meaning model**, learned to interpret sentences in terms of a visual micro-world, forming grounded internal representations of sentence meanings. Grounding of linguistic representations in other forms of experience is key to forming a true understanding of language at levels beyond that of syntax; however, symbolic and statistical natural language processing systems do not generally address this problem.

Distinguishing it from many past neural network models, the grounded-meaning model performs systematically, making the correct generalizations after observing only a tiny fraction of the input space. The discussion in this chapter presents evidence that the learned representations appear to be largely compositional in nature, demonstrating that a neural network can learn distributed representations that it can manipulate and combine much like the symbols used in high-level cognitive models. This finding is important because compositional representations provide the means of efficiently encoding and computing over very high-dimensional spaces—a quality that connectionist models are often accused of lacking.

Chapter 5, after discussing relevant previous work, presents two more advanced versions of the model from Chapter 4. The first of these, termed the **meaning-answer model**, not only understands the sentences it hears, but recognizes questions and shows its ability to answer them by producing a grounded representation of the meaning of a complete-sentence answer. The final version of the model, termed the **spoken-answer model**, speaks its answers, directly producing sequences of speech sounds without recourse to the experimenter-imposed internal meaning representations utilized by the previous versions of the model. This last and most complex model is also the most plausible in terms of its learning method, as both its input and its training signal would be readily observable in any human linguistic interaction.

The spoken-answer model fulfills the criteria set out earlier for a useful neural-level model of language learning at the sentence level. Currently, it is the only known neural network model capable of simultaneously learning to understand and produce sentence-level language that is represented sublexically as temporal sequences of phonemes. When compared to symbolic or statistical models of natural language processing, the model's computational substrate and its training regimen more closely resemble those of humans, lending credence to the idea that scaling such a system up would produce a more human-like natural language processing system. Unfortunately, large-scale natural language applications are not yet feasible, pending either greater computational resources or special-purpose neural network hardware. As the concluding remarks in Chapter 6 discuss, however, the models presented here have immediate potential applicability in a number of areas, including brain-level investigations as a successor to the WLG aphasia model, as well as higher-level models of psycholinguistic phenomena.

Chapter 2

Generalizing a Training Algorithm

2.1 Introduction

The second-order recurrent neural network models in this dissertation have two major requirements that existing neural training methods cannot satisfy simultaneously. The first requirement is the ability to train architectures with deep serial structure involving multiple internal layers of units, which is important for tasks that decompose hierarchically or involve multiple levels of representation. The second requirement is informational locality, both in space and in time, which is a physical requirement of biological brains that induces important distributive and competitive properties within the network. Both of these are essential properties that differentiate the neural substrate from the traditional substrate of computing machinery. The goal of creating a natural language processing model that is truly grounded in a human-like neural substrate, then, necessitates the creation of a new training algorithm that can fulfill both requirements. This chapter describes such an algorithm, which generalizes the spatially and temporally local training method originally used with networks of the **long short-term memory** (LSTM; Hochreiter and Schmidhuber, 1997) architecture.

The LSTM is a recurrent neural network architecture that combines fast training with efficient learning on tasks that require sequential short-term memory storage for many time-steps during a trial. Since its inception, LSTM has been augmented and improved with forget gates (Gers and Cummins, 2000) and peephole connections (Gers and Schmidhuber, 2000); despite this, the usefulness of the LSTM training algorithm is limited in that it can only train a small set of second-order recurrent network architectures. The term **second-order neural network** in this context means one that not only allows normal weighted connections that propagate activation from one sending unit to one receiving unit, but also allows second-order connections: weighted connections from *two* sending units to one receiving unit, where the signal received is dependent upon the product of the activities of the sending units with each other and the connection weight (Miller and Giles, 1993). When LSTM is described in terms of connection gating, as discussed later, it becomes clear that the gate units serve as the additional sending units for the second-order connections.

The original LSTM architecture has an input layer, a hidden layer consisting of cell assemblies called memory blocks, and an output layer. Each memory block is composed of memory cell units that retain state across time-steps, as well as three types of specialized gate units that learn to protect, utilize, or destroy this state as appropriate. The LSTM training algorithm back-propagates errors from the output units through the memory blocks, adjusting incoming connections of all units in the blocks, but then truncates the back-propagated errors. As a consequence, LSTM's training algorithm cannot be used to effectively train second-order networks with units placed between the memory blocks and the input layer. More generally, LSTM's training algorithm cannot be used to train arbitrary second-order recurrent neural architectures, as the error propagation and weight updates it prescribes are dependent upon the specific network architecture described in the original paper (Hochreiter and Schmidhuber, 1997).

While there exist other methods capable of training arbitrary second-order networks, none share a principal advantage of LSTM's training algorithm: locality in time and space. **Spatial locality** here means that the changes to connection weights that happen at some location in the network should be directly computable from information available within the spatial neighborhood of the connection in question. Similarly, **temporal locality** means that weight changes cannot rely upon precise records of information from arbitrarily far in the past. A training algorithm that possesses these properties can, if it is general enough, be applied without modification to any network architecture, and in fact even to architectures that change during training. The human brain is an obvious example of an architecture that changes during learning, and concerns about spatial and temporal locality might be summarized as a desire to maintain the locality constraints that brains appear to have.

Error back-propagation for feed-forward networks (Rumelhart et al., 1986) and simple recurrent networks (SRNs; Elman, 1990) are examples of training algorithms that exhibit locality in time and space while generalizing to a wide variety of architectures. To date, it appears as if no such algorithm for arbitrary second-order networks has been described. Algorithms like back-propagation through time (BPTT; Werbos, 1990) that have been used to train second-order architectures (e.g., Graves and Schmidhuber, 2008) violate temporal locality by basing weight updates on perfect records of network activations extending back in time to the beginning of arbitrarily long input sequences; the same goes for the evolutionary training method known as Evolino (Schmidhuber et al., 2007). Real-time recurrent learning (RTRL; Williams and Zipser, 1989) is not local spatially since the gradient term for a given weight depends directly on every other weight in the network. Decoupled extended Kalman filters (DEKF; Gers et al., 2003; Puskorius and Feldkamp, 1994) utilize a host of external matrix operations to control training and thus are not spatially local. While this list is not exhaustive, every such training algorithm examined, other than LSTM, was either spatially or temporally nonlocal.

The desire for an architecture-independent training algorithm with these properties led to the development of the training algorithm described in this chapter, the **generalized long short-term memory** (LSTM-g; Monner and Reggia, in press a). This approach retains the spatial and temporal locality of the original LSTM training algorithm and can be applied, without modification, to a much wider range of second-order recurrent neural networks. Each unit in a network trained by LSTM-g has the same set of operating instructions, relying only on its local network environment to determine whether it will fulfill the role of memory cell, gate unit, both, or neither. In addition, every unit is trained by LSTM-g in the same way—a sharp contrast from the original LSTM training algorithm, where each of several different types of units has a unique training regimen. LSTM-g reinterprets the gating of unit activations seen in LSTM; instead of gate units modulating other unit activations directly, they are viewed as modulating the weights on connections between units. This change in perspective, while mathematically equivalent to the original design, offers increased flexibility to network designers who wish to explore arbitrary architectures where gates can temporarily isolate one part of the network from another. While previous work (Bayer et al., 2009) has examined alternative architectures for LSTM-style second-order networks—as derived by network evolution to suit a particular task—that work relies on the nonlocal BPTT algorithm to train the evolved networks. While this chapter focuses on alternative second-order architectures as well, the primary aim is to provide a local algorithm to train them.

In addition to its expanded architectural applicability, LSTM-g provides all of the benefits of the LSTM training algorithm when applied to the right type of architecture. LSTM-g was designed to perform exactly the same weight updates as the original algorithm when applied to identical network architectures. However, on LSTM architectures with peephole connections, LSTM-g often performs better than the original algorithm by utilizing a source of back-propagated error that appears to have heretofore gone unnoticed.

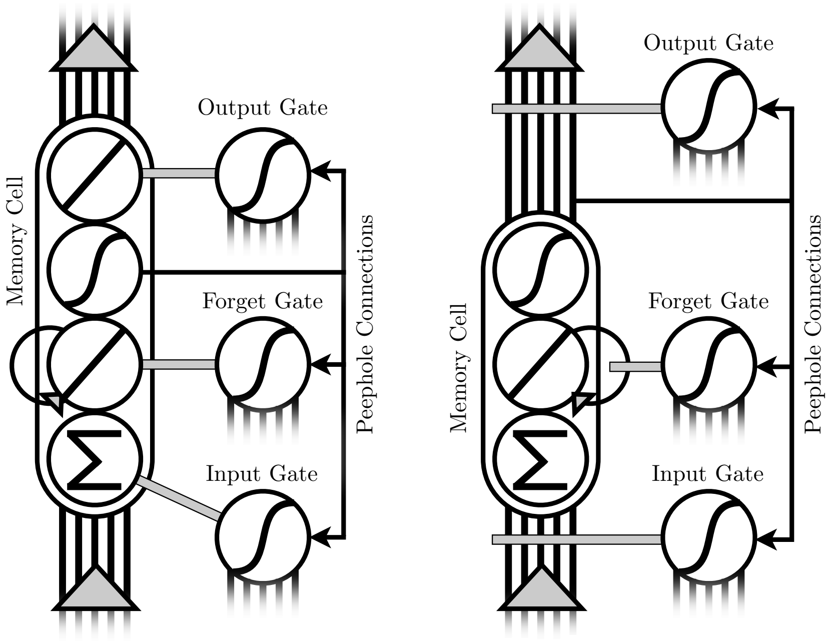
Section 2.2 presents LSTM and its training algorithm as previously described, followed by the generalized version in Section 2.3. Section 2.4 provides the mathematical derivation of the LSTM-g learning rules, and Section 2.5 then proves that LSTM-g is a generalization of LSTM training. Section 2.6 presents an analysis of the additional error signals that LSTM-g uses to its advantage, followed by experimental evidence in Section 2.7 that LSTM-g often performs better than the LSTM algorithm when using the original architecture. Further experiments show that LSTM-g performs well on two architectures specifically adapted to two computational problems.

2.2 LSTM

LSTM was developed as a neural network architecture for processing long temporal sequences of data and is trained using a hybrid descendant of truncated BPTT and RTRL. Other recurrent neural networks trained with various gradient methods proved to be ineffective when the input sequences were too long (Hochreiter and Schmidhuber, 1997). Analysis showed that for neural networks trained with back-propagation or other gradient-based methods, the error signal is likely to vanish or diverge as error travels backward through network space or through time. This is because, with every pass backward through a unit, the error signal is scaled by the derivative of the unit's activation function times the weight that the forward signal traveled along. The further the error travels back in space or time, the more times this scaling factor is multiplied into the error term. If the factor is consistently less than 1, the error will vanish, leading to small, ineffective weight updates; if it is greater than 1, the error term will diverge, potentially leading to weight oscillations or other types of instability. One way to preserve the value of the error is by requiring the scaling factor to be equal to 1, which can only be consistently enforced with a linear activation function (whose derivative is 1) and a fixed weight of $w_{jj} = 1$. LSTM adopts this requirement for its **memory cell** units, which have linear activation functions and self-connections with a fixed weight of 1 (Figure 2.1(a)). This allows them to maintain unscaled activations and error derivatives across arbitrary time lags if they are not otherwise disturbed. Since back-propagation networks require nonlinear hidden unit activation functions to be effective, each memory cell's state is passed through a squashing function—such as the standard logistic function—before being passed on to the rest of the network.

The processing of long temporal sequences is complicated by the issue of interference. If a memory cell is currently storing information that is not useful now but will be invaluable later, this currently irrelevant information may interfere with other processing in the interim. This in turn may cause the information to be discarded, improving performance in the near term but harming it in the long term. Similarly, a memory cell may be perturbed by an irrelevant input, and the information that would have been useful later in the sequence can be lost or obscured. To help mitigate these issues, each memory cell has its net input modulated by the activity of another unit, termed an **input gate**, and has its output modulated by a unit called an **output gate** (Figure 2.1(a)). Each input gate and output gate unit modulates one or a small number of memory cells; the collection of memory cells together with the gates that modulate them is termed a **memory block**. The input gates provide a context-sensitive way to update the contents of a memory cell and protect those contents from interference, while the output gates protect downstream units from perturbation by stored information that has not become relevant yet. A later innovation was a third gate, termed the **forget gate**, which modulates the amount of activation a memory cell keeps from the previous time-step, providing a method to quickly discard the contents of a memory cell after this information has served its purpose (Gers and Cummins, 2000).

In the original formulation of LSTM, the gate units responsible for isolating the contents of a given memory cell face a problem. These gates may receive input connections from the memory cell itself, but the memory cell's value is gated by its output gate. The result is that, when the output gate is closed (i.e., has activity near zero), the memory cell's visible activity is near zero, hiding its contents even from those cells—the associated gates—that are supposed to be controlling its information flow. Recognition of this fact resulted in the inclusion of **peephole connections**—direct weighted connections originating from an intermediate stage of processing in



(a) LSTM memory block

(b) Equivalent for LSTM-g

Figure 2.1: Architectural comparison of LSTM's memory block to the equivalent in an LSTM-g network. Weighted connections are shown as black lines, and gating relationships are shown as thicker gray lines. The elongated enclosure represents the extent of the memory cell. In (a), connections into the LSTM memory cell are first summed (the input-squashing function is taken to be the identity); the result is gated by the input gate. The gated net input progresses to the self-recurrent linear unit, whose activity is gated by the forget gate. The state of the recurrent unit is passed through the output-squashing function, which is then modulated by the output gate. This modulated value is passed to all receiving units via weighted connections. The peephole connections project from an internal stage in the memory cell to the controlling gate units; this is an exception to the rule that only the final value of the memory cell is visible to other units. In (b), the weights on the input connections to the LSTM-g memory cell are modulated directly by the input gate before being summed by the linear unit. Unmodulated output leaves the memory cell via weighted connections. Connections to downstream units can have their weights modulated directly by the output gate, but this is not required, as can be seen with the equivalent of LSTM's peephole connections proceeding normally from the output of the memory cell. This scheme is capable of producing the same results as the LSTM memory block, but allows greater architectural flexibility.

the memory cell and projecting to each of the memory cell's gates (Gers and Schmidhuber, 2000). Unlike all other connections originating at the memory cell, the peephole connections see the memory cell's state *before* modulation by the output gate, and thus are able to convey the true contents of the memory cell to the associated gates at all times. By all accounts, peephole connections improve LSTM performance significantly (Gers and Schmidhuber, 2001), leading to their adoption as a standard technique employed in applications (Graves et al., 2004; Gers et al., 2003).

The LSTM network ostensibly has only three layers: an input layer, a layer of memory block cell assemblies, and an output layer. For expository purposes, however, it will be useful to think of the memory block assemblies as composed of multiple separate layers (see Figure 2.2): the input gate layer (ι), the forget gate layer (φ), the memory cell layer (c), and the output gate layer (ω). For notational simplicity, assume each of these layers has the same number of elements, implying that a single memory cell c_j is associated with the set of gates ι_j, φ_j, and ω_j; it is trivial, though notationally messy, to generalize to the case where a set of gates can control more than one memory cell. The input layer projects a full set of connections to each of these layers; the memory cell layer projects a full set of connections to the output layer (θ). In addition, each memory cell c_j projects a single ungated peephole connection to each of its associated gates (see Figure 2.1(a)). The architecture can be augmented with direct input-to-output connections and/or delayed recurrent connections among the memory cell and gate layers. As will become evident

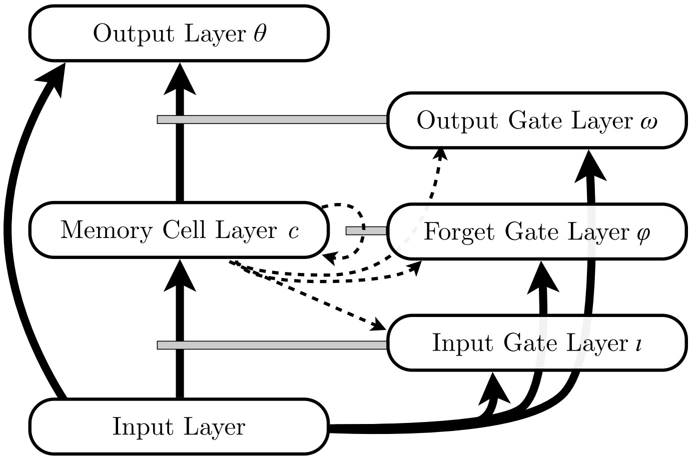


Figure 2.2: The standard LSTM architecture in terms of layers. The memory block assemblies are broken up into separate layers of memory cells, input gates, forget gates, and output gates, in addition to the input and output layers. Solid arrows indicate full all-to-all connectivity between units in a layer, and dashed arrows indicate connectivity only between the units in the two layers that have the same index (i.e., the first unit of the sending layer only projects to the first unit of the receiving layer, the second unit only projects to the second, and so forth). The gray bars denote gating relationships and are displayed as they are conceived in LSTM-g, with the modulation occurring at the connection level. Units in each gate layer modulate only those connections into or out of their corresponding memory cell. The circular dashed connection on the memory cell layer indicates the self-connectivity of the units therein. This diagram shows the optional peephole connections—the dashed arrows originating at the memory cells and ending at the gate layers—as well as the optional direct input-to-output layer connections on the left.

in the following sections, the operation of the LSTM training algorithm is very much dependent upon the specifics of the LSTM architecture.

The following equations detail the operation of the LSTM network and its original training algorithm through a single time-step. A time-step is defined to consist of the presentation of a single input, followed by the activation of all subsequent layers of the network in order. This notion of a time-step, most often seen when discussing feed-forward networks, enables a more intuitive description of the activation and learning phases that occur due to each input. Following Graves et al. (2004), all variables in the following refer to the most recently calculated value of that variable (whether during this time-step or the last), with the exception that variables with a hat ($\hat{\ }$) always refer to the value calculated one time-step earlier; this only happens in cases where the new value of a variable is being defined in terms of its immediately preceding value. Following Gers and Schmidhuber (2000), the following definitions deviate from the original description of LSTM by reducing the number of squashing functions for the memory cells; here, however, the definition omits the input-squashing function g (equivalent to defining $g(x) = x$) and retains the output-squashing function, naming it f_{c_j} for memory cell j. In general, the formalism that follows will use the subscript index j to refer to individual units within the layer in question, with i running over all units that project connections to unit j, and k running over all units that receive connections from j.

2.2.1 Activation dynamics

When an input is presented, the network proceeds through an entire time-step, activating each layer in order: $\iota, \varphi, c, \omega$, and finally the output layer θ. In general, when some layer λ is activated, each unit λ_j in that layer calculates its **net input** x_{λ_j} as the weighted sum over all its input connections from units i (Equation 2.1). The units i vary for each layer and potentially include recurrent connections; the most recent activation value of the sending unit is always used, even if it is from the previous time-step as for recurrent connections. For units that are not in the memory cell layer, the **activation** y_{λ_j} of the unit is the result of applying the unit's **squashing function** f_{λ_j} (generally taken to be the logistic function) to its net input (Equation 2.2). Each memory cell unit, on the other hand, retains its previous state $\widehat{s_{c_j}}$ in proportion to the activation of the associated forget gate; the current **state** s_{c_j} is updated by the net input modulated by the activation of the associated input gate (Equation 2.3). A memory cell's state is passed through its squashing function and modulated by the activation of its output gate to produce the cell's activation (Equation 2.4).

$$x_{\lambda_j} = \sum_i w_{\lambda_{ji}}\, y_i \text{ for } \lambda \in \{\iota, \varphi, c, \omega, \theta\} \tag{2.1}$$

$$y_{\lambda_j} = f_{\lambda_j}(x_{\lambda_j}) \text{ for } \lambda \in \{\iota, \varphi, \omega, \theta\} \tag{2.2}$$

$$s_{c_j} = y_{\varphi_j}\, \widehat{s_{c_j}} + y_{\iota_j} x_{c_j} \tag{2.3}$$

$$y_{c_j} = y_{\omega_j}\, f_{c_j}(s_{c_j}) \tag{2.4}$$

When considering peephole connections in the context of the equations in this section, one should replace the sending unit activation y_i with the memory cell state s_{c_i} since peephole connections come directly from the internal state of the memory cells rather than their activations.

2.2.2 Learning rules

To learn effectively, each unit needs to keep track of the activity flow over time through each of its connections. To this end, each unit maintains an **eligibility trace** for each of its input connections and updates this trace immediately after calculating its activation. The eligibility trace for a given connection is an aggregate record of the amount of activation that has crossed the connection and that still has influence over the state of the network. It is similar in spirit to those used in temporal-difference learning (Sutton and Barto, 1998), except that here the decay of these traces is variable and learnable. When a target vector is presented, the eligibility traces are used to help assign error responsibilities to individual connections. For the output gates and output units, the eligibility traces are instantaneous—they are simply the most recent activation value that crossed the connection (Equation 2.5). For the memory cells (Equation 2.6), forget gates (Equation 2.7), and input gates (Equation 2.8), the eligibility traces are partial derivatives of the state of the memory cell with respect to the connection in question; simplifying these partial derivatives results in Equations 2.6–2.8. Previous eligibility traces are retained in proportion to the amount of state that the memory cell retains (i.e., the forget gate activation y_{φ_j}), and each is incremented according to the effect it has on the memory cell state.

$$\varepsilon_{\lambda_{ji}} = y_i \text{ for } \lambda \in \{\omega, \theta\} \tag{2.5}$$

$$\varepsilon_{c_{ji}} = y_{\varphi_j} \, \widehat{\varepsilon_{c_{ji}}} + y_{\iota_j} \, y_i \tag{2.6}$$

$$\varepsilon_{\varphi_{ji}} = y_{\varphi_j} \, \widehat{\varepsilon_{\varphi_{ji}}} + \widehat{s_{c_j}} \, f'_{\varphi_j}(x_{\varphi_j}) \, y_i \tag{2.7}$$

$$\varepsilon_{\iota_{ji}} = y_{\varphi_j} \, \widehat{\varepsilon_{\iota_{ji}}} + x_{c_j} \, f'_{\iota_j}(x_{\iota_j}) \, y_i \tag{2.8}$$

Between time-steps of the activation dynamics (i.e., after the network has generated an output for a given input), the network may be given a target vector t to compare against, where all values in t are in the range $[0, 1]$. The difference between the network output and the target is calculated using the cross-entropy function in Equation 2.9 (Hinton, 1989). Since $E \leq 0$ when the t and y values fall in the range $[0, 1]$ as required, one trains the network by driving this function towards zero from below, or alternatively driving its negation towards zero from above. Deriving Equation 2.9 with respect to the output unit activations reveals the **error responsibility** δ_{θ_j} for the output units (Equation 2.10). One obtains the deltas for the output gates (Equation 2.11) and the remaining units (Equation 2.12) by propagating the error backward through the network.

$$E = \sum_{j \in \theta} \big(t_j \log(y_{\theta_j}) + (1 - t_j) \log(1 - y_{\theta_j}) \big) \tag{2.9}$$

$$\delta_{\theta_j} = t_j - y_{\theta_j} \tag{2.10}$$

$$\delta_{\omega_j} = f'_{\omega_j}(x_{\omega_j}) \, f_{c_j}(s_{c_j}) \sum_{k \in \theta} \delta_{\theta_k} \, w_{\theta_k c_j} \tag{2.11}$$

$$\delta_{\lambda_j} = f'_{c_j}(s_{c_j}) \, y_{\omega_j} \sum_{k \in \theta} \delta_{\theta_k} \, w_{\theta_k c_j} \text{ for } \lambda \in \{\iota, \varphi, c\} \tag{2.12}$$

Finally, the connection weights into all units in each layer λ are updated according to the product of the learning rate α, the unit's error responsibility δ_{λ_j}, and the connection's eligibility trace $\varepsilon_{\lambda_{ji}}$ (Equation 2.13).

$$\Delta w_{\lambda_{ji}} = \alpha \, \delta_{\lambda_j} \, \varepsilon_{\lambda_{ji}} \text{ for } \lambda \in \{\iota, \varphi, c, \omega, \theta\} \tag{2.13}$$

2.3 Generalized LSTM

The long short-term memory algorithm, as presented in Section 2.2, is an efficient and powerful recurrent neural network training method, but is limited in applicability to the architecture shown in Figure 2.2 and sub-architectures thereof.[1] In particular, any architectures with multiple hidden layers (where another hidden layer projects to the memory block layer) cannot be efficiently trained because error responsibilities are truncated at the memory blocks instead of being passed to upstream layers. This section details the new generalized version of LSTM training, which confers all the benefits of the original algorithm, yet can be applied without modification to arbitrary second-order neural network architectures.

The generalized long short-term memory (LSTM-g) approach reinterprets the gating mechanism employed by LSTM. In LSTM, gate units directly act on the states of individual units—a memory cell's net input in the case

[1] LSTM can also train architectures with additional layers that operate in *parallel* with the memory block layer, but the important point here is that LSTM cannot effectively train architectures containing layers that operate in *series* with the memory block layer.

of the input gate, the memory cell state for the forget gate, and the memory cell output for the output gate (Equations 2.3–2.4). By contrast, units in LSTM-g can gate at the level of individual connections. The effect is that, when passing activity to unit j from unit i across a connection gated by k, the result is not simply $w_{ji}\, y_i$, but instead $w_{ji}\, y_i\, y_k$. In this sense, LSTM-g is similar in form to traditional second-order networks (e.g., Giles and Maxwell, 1987; Psaltis et al., 1988; Shin and Ghosh, 1991; Miller and Giles, 1993), but with an asymmetry: The notation used here considers the connection in this example to be primarily defined by j and i (the weight is denoted w_{ji} and not w_{jki}), where k provides a temporary **gain** on the connection by modulating its weight multiplicatively. This notation is convenient when considering connections that require an output and an input but may or may not be gated; in other words, the notation can refer to a connection without specifying whether it is a first- or second-order connection.

In LSTM-g, every unit has the potential to be like LSTM's memory cells, gate units, both, or neither. That is to say, all units contain the same set of operating instructions for both activation and learning. Self-connected units can retain state like a memory cell, and any unit can directly gate any connection. The role each unit takes is completely determined by its placement in the overall network architecture, leaving the choice of responsibilities for each unit entirely up to the architecture designer.

Equations 2.14–2.24 describe the operation of a network trained by LSTM-g through a single time-step. Just as in the description of the original

LSTM training algorithm, a time-step consists of the presentation of a single input pattern, followed by the ordered activation of all noninput layers of the network. The order in which layers are activated is predetermined and remains fixed throughout training. If a layer to be activated receives recurrent connections from a layer that has not yet been activated during this time-step, the sending layer's activations from the previous time-step are used. The full derivation of the LSTM-g training algorithm can be found in Section 2.4.

2.3.1 Activation dynamics

LSTM-g performs LSTM-like gating by having units modulate the effectiveness of individual connections. As such, the formalism begins by specifying the **gain** g_{ji} on the connection from unit i to unit j (Equation 2.14).

$$g_{ji} = \begin{cases} 1 & \text{if connection from } i \text{ to } j \text{ is not gated} \\ y_k & \text{if unit } k \text{ gates the connection from } i \text{ to } j \end{cases} \tag{2.14}$$

Much as with memory cells in LSTM, any unit in an LSTM-g network is capable of retaining state from one time-step to the next, based only on whether or not it is self-connected. The **state** s_j of a unit j (Equation 2.15) is the sum of the weighted, gated activations of all the units that project connections to it. If the unit is self-connected, it retains its state in proportion to the gain on the self-connection. As in LSTM, self-connections in LSTM-g, where they exist, have a fixed weight of 1; otherwise $w_{jj} = 0$. Given the state s_j, the **activation** y_j is calculated via the application of the unit's **squashing function** f_j (Equation 2.16).

$$s_j = g_{jj}\, w_{jj}\, \widehat{s_j} + \sum_{i \neq j} g_{ji}\, w_{ji}\, y_i \tag{2.15}$$

$$y_j = f_j(s_j) \tag{2.16}$$

When considering these equations as applied to the LSTM architecture, for a unit $j \notin c$, it is clear that Equation 2.15 is a generalization of Equation 2.1. This is because the first term of Equation 2.15 evaluates to zero on this architecture (since there is no self-connection w_{jj}), and all the $g_{ji} = 1$ since no connections into the unit are gated. The equivalence of Equation 2.16 and Equation 2.2 for these units follows immediately. For a memory cell $j \in c$, on the other hand, Equation 2.15 reduces to Equation 2.3 after three simplifications: the self-connection gain g_{jj} is just y_{φ_j}; the self-connection weight w_{jj} is 1; and the g_{ji} are all equal to y_{ι_j} and can thus be pulled outside the sum. However, for the memory cell units, Equation 2.16 is not equivalent to Equation 2.4, since the latter already multiplies in the activation y_{ω_j} of the output gate, whereas this modulation is performed at the connection level in LSTM-g.

2.3.2 Learning rules

As in LSTM, each unit keeps an **eligibility trace** ε_{ji} for each of its incoming connections (Equation 2.17). This quantity keeps track of how activity that has crossed this connection has influenced the current state of the unit and is equal to the partial derivative of the state with respect to the connection weight in question. For units that do not have a self-connection, the eligibility trace ε_{ji} reduces to the most recent input activation modulated by the gating signal.

$$\varepsilon_{ji} = g_{jj}\, w_{jj}\, \widehat{\varepsilon_{ji}} + g_{ji}\, y_i \tag{2.17}$$

In the context of the LSTM architecture, Equation 2.17 reduces to Equation 2.5 for the output gates and output units; in both cases, the lack of self-connections forces the first term to zero, and the remaining $g_{ji}\, y_i$ term is equivalent to LSTM's y_i.

If unit j gates connections into other units k, it must maintain a set of **extended eligibility traces** ε_{ji}^{k} for each such k (Equation 2.18). A trace of this type captures the effect that the connection from i potentially has on the state of k through its influence on j. Equation 2.18 is simpler than it appears, as the remaining partial derivative term is 1 if and only if j gates k's self-connection, and is 0 otherwise. Further, the index a, by definition, runs over only those units whose connections to k are gated by j; this set of units may be empty.

$$\varepsilon_{ji}^{k} = g_{kk}\, w_{kk}\, \widehat{\varepsilon_{ji}^{k}} + f_j'(s_j)\, \varepsilon_{ji}\, \left(\frac{\partial g_{kk}}{\partial y_j} w_{kk}\, \widehat{s_k} + \sum_{a \neq k} w_{ka}\, y_a \right) \tag{2.18}$$

It is worth noting that LSTM uses traces of exactly this type for the forget gates and input gates (Equations 2.7–2.8); it just so happens that each such unit gates connections into exactly one other unit, thus requiring each unit to keep only a single, unified eligibility trace for each input connection. This will also be the case for the alternative architectures explored in later sections, but is not required. A complete explanation of the correspondence between LSTM's eligibility traces and the extended eligibility traces utilized by LSTM-g can be found in Section 2.5.

When a network is given a target vector, each unit must calculate its **error responsibility** δ_j and adjust the weights of its incoming connections accordingly. Output units, of course, receive their δ values directly from the environment based on the global error function (Equation 2.9), just as in LSTM (Equation 2.10). The error responsibility δ_j for any other unit j in the network can be calculated by back-propagating errors. Since each unit keeps separate eligibility traces corresponding to projected activity (ε_{ji}) and gating activity (ε_{ji}^{k}), the error responsibility is divided accordingly. First, let P_j be the set of units k which are downstream from j—that is, activated after j during a time-step—and to which j projects weighted connections (Equation 2.19). Similarly, let G_j be the set of units k which are downstream from j that receive connections gated by j (Equation 2.20). Both of these sets are restricted to downstream units because an upstream unit k has its error responsibility updated after j during backward error propagation, meaning that the error responsibility information provided by k is not available when j would need to use it.

$$P_j = \{k \mid j \text{ projects a connection to } k \text{ and } k \text{ is downstream of } j\} \quad (2.19)$$

$$G_j = \{k \mid j \text{ gates a connection into } k \text{ and } k \text{ is downstream of } j\} \quad (2.20)$$

The error responsibility of unit j with respect to the projected connections in P_j (Equation 2.21) is calculated as the sum of the error responsibilities of the receiving units weighted by the gated connection strengths that activity from j passed over to reach k.

$$\delta_{P_j} = f_j'(s_j) \sum_{k \in P_j} \delta_k \, g_{kj} \, w_{kj} \quad (2.21)$$

Since the memory cells in LSTM only project connections and perform no gating themselves, δ_{P_j} of Equation 2.21 translates directly into Equation 2.12 for these units, which can be seen by noting that the gain terms g_{kj} are all equal to the output gate activation y_{ω_j} and can be pulled out of the sum.

The error responsibility of j with respect to gating activity is the sum of the error responsibilities of each unit k receiving gated connections times a quantity representing the gated, weighted input that the connections provided to k (Equation 2.22). This quantity, as with the same quantity in Equation 2.18, is simpler than it appears. The partial derivative evaluates to 1 only when j is gating k's self-connection and to 0 otherwise, so this term either simplifies or drops completely. The index a runs over only those units projecting a connection to k on which j is the gate.

$$\delta_{G_j} = f_j'(s_j) \sum_{k \in G_j} \delta_k \left(\frac{\partial g_{kk}}{\partial y_j} w_{kk}\, \widehat{s_k} + \sum_{a \neq k} w_{ka}\, y_a \right) \tag{2.22}$$

To find j's total error responsibility (Equation 2.23), the error responsibilities due to projections and gating are added.

$$\delta_j = \delta_{P_j} + \delta_{G_j} \tag{2.23}$$

To obtain weight changes similar to LSTM training, δ_j is not used directly in weight adjustments; its purpose is to provide a unified δ value that can be used by upstream units to calculate their error responsibilities due to unit j. Instead, the weights are adjusted by combining the error responsibilities and eligibility traces for projected activity and adding the products of extended eligibility traces and error responsibilities of each unit receiving gated connections. The result is multiplied by the learning rate α (Equation 2.24).

$$\Delta w_{ji} = \alpha\, \delta_{P_j}\, \varepsilon_{ji} + \alpha \sum_{k \in G_j} \delta_k\, \varepsilon_{ji}^{k} \tag{2.24}$$

Section 2.4 provides a detailed derivation of these learning rules, including a complete treatment of all deviations from the true gradient. Section 2.5 shows that the weight changes made by both the LSTM (Equation 2.13) and LSTM-g (Equation 2.24) algorithms are identical when used on an LSTM architecture without peephole connections. This establishes that LSTM-g is indeed a generalization of LSTM's training algorithm.

2.4 Learning rule derivation

This section derives the LSTM-g learning algorithm by calculating the gradient of the cross-entropy function (Equation 2.9) for a general unit in an LSTM-g network. Such a unit may both project normal weighted connections to other units and multiplicatively modulate the weights of other connections. The error responsibility for a general unit j can be obtained by approximating the gradient of the error function with respect to the unit's state s_j (Equation 2.25). This approximation posits that the error responsibility of unit j depends only upon units k that are immediately downstream from j (Equation 2.26). The remaining error gradients for k are δ_k by definition. Equation 2.27 breaks up the second partial derivative into a product that includes j's activation directly so as to account for the effects of j's squashing function separately. The dependence of j's activation on its state is simply the derivative of the squashing function, which is constant across the sum and thus can be moved outside (Equation 2.28). Equation 2.29 separates the set of units k into two (possibly overlapping) sets—those units to which j projects weighted connections (Equation 2.19) and those units that receive connections gated by j (Equation 2.20). The derivation will thus handle error responsibilities for projection and gating separately, even in cases where j both projects and gates connections into the same unit k, thereby defining δ_{P_j} and δ_{G_j} (Equation 2.30, cf. Equation 2.23).

$$\begin{aligned}
\delta_j &= \frac{\partial E}{\partial s_j} && (2.25)\\
&\approx \sum_k \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial s_j} && (2.26)\\
&= \sum_k \delta_k \frac{\partial s_k}{\partial y_j} \frac{\partial y_j}{\partial s_j} && (2.27)\\
&= f_j'(s_j) \sum_k \delta_k \frac{\partial s_k}{\partial y_j} && (2.28)\\
&= f_j'(s_j) \sum_{k \in P_j} \delta_k \frac{\partial s_k}{\partial y_j} + f_j'(s_j) \sum_{k \in G_j} \delta_k \frac{\partial s_k}{\partial y_j} && (2.29)\\
&= \delta_{P_j} + \delta_{G_j} && (2.30)
\end{aligned}$$

In calculating δ_{P_j}, Equation 2.32 expands s_k from Equation 2.31 using the definition from Equation 2.15. This part of the derivation is only concerned with cases where j projects connections; as such, g_{kk} and $g_{kj'}$

do not depend on j and are treated as constants. Since the previous state of k does not depend on the current activation of j, the first term in the parentheses vanishes completely. Individual terms in the sum vanish as well, except for the one case when $j' = j$, leaving $g_{kj}\, w_{kj}$ as the entire derivative (Equation 2.33, cf. Equation 2.21).

$$\begin{aligned}
\delta_{P_j} &= f'_j(s_j) \sum_{k \in P_j} \delta_k \frac{\partial s_k}{\partial y_j} && (2.31)\\
&= f'_j(s_j) \sum_{k \in P_j} \delta_k \frac{\partial}{\partial y_j} \left(g_{kk}\, w_{kk}\, \widehat{s_k} + \sum_{j' \neq k} g_{kj'}\, w_{kj'}\, y_{j'} \right) && (2.32)\\
&= f'_j(s_j) \sum_{k \in P_j} \delta_k\, g_{kj}\, w_{kj} && (2.33)
\end{aligned}$$

To find δ_{G_j}, Equation 2.35 similarly begins by expanding s_k from Equation 2.34. In this case, the derivation concerns only connections that j gates. Now the g_{kk} term is in play, since it is equal to y_j if j gates k's self-connection (see Equation 2.14); this leads to the first term inside the parentheses in Equation 2.36, where the derivative can take the values 1 or 0. For individual terms in the sum, $y_{j'} \neq y_j$ since this derivation does not deal with connections j projects to k. However, $g_{kj'}$ may be equal to y_j in some cases; where it is not, j does not gate the connection and the term goes to zero. Thus, the sum is reindexed to include only those units a whose connections to k are gated by j (Equation 2.36, cf. Equation 2.22).

$$\delta_{G_j} = f'_j(s_j) \sum_{k \in G_j} \delta_k \frac{\partial s_k}{\partial y_j} \tag{2.34}$$

$$= f'_j(s_j) \sum_{k \in G_j} \delta_k \frac{\partial}{\partial y_j} \left(g_{kk}\, w_{kk}\, \widehat{s_k} + \sum_{j' \neq k} g_{kj'}\, w_{kj'}\, y_{j'} \right) \tag{2.35}$$

$$= f'_j(s_j) \sum_{k \in G_j} \delta_k \left(\frac{\partial g_{kk}}{\partial y_j} w_{kk}\, \widehat{s_k} + \sum_{a \neq k} w_{ka}\, y_a \right) \tag{2.36}$$

This allows the calculation of the error responsibility δ_j of any unit j by back-propagation. Starting at the level of units k that are immediately downstream from j (Equation 2.37), Equation 2.38 separates the k units by function, and Equation 2.39 breaks up the remaining partial derivative in the first sum. Rearranging terms in the first sum (Equation 2.40) reveals a grouping equal to δ_{P_j} (see Equation 2.31). The derivative in the first term is equal to the eligibility trace ε_{ji}; the second term is the extended eligibility trace ε^k_{ji}. These substitutions give Equation 2.41 (cf. Equation 2.24).

$$\Delta w_{ji} \approx \alpha \sum_k \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial w_{ji}} \tag{2.37}$$

$$= \alpha \sum_{k \in P_j} \delta_k \frac{\partial s_k}{\partial w_{ji}} + \alpha \sum_{k \in G_j} \delta_k \frac{\partial s_k}{\partial w_{ji}} \tag{2.38}$$

$$= \alpha \sum_{k \in P_j} \delta_k \frac{\partial s_k}{\partial y_j} \frac{\partial y_j}{\partial s_j} \frac{\partial s_j}{\partial w_{ji}} + \alpha \sum_{k \in G_j} \delta_k \frac{\partial s_k}{\partial w_{ji}} \tag{2.39}$$

$$= \alpha \left(f'_j(s_j) \sum_{k \in P_j} \delta_k \frac{\partial s_k}{\partial y_j} \right) \frac{\partial s_j}{\partial w_{ji}} + \alpha \sum_{k \in G_j} \delta_k \frac{\partial s_k}{\partial w_{ji}} \tag{2.40}$$

$$= \alpha\, \delta_{P_j}\, \varepsilon_{ji} + \alpha \sum_{k \in G_j} \delta_k\, \varepsilon^k_{ji} \tag{2.41}$$

To calculate the eligibility trace ε_{ji}, Equation 2.43 substitutes s_j from Equation 2.15 into Equation 2.42. Assuming unit j does not gate its own self-connection, g_{jj} is a constant. The previous state $\widehat{s_j}$ depends on w_{ji} producing a partial derivative that simplifies to the previous value of the eligibility trace, $\widehat{\varepsilon_{ji}}$. The only term in the sum with a nonzero derivative is the case where $i' = i$, so a simplification produces Equation 2.44 (cf. Equation 2.17).

$$\begin{aligned}
\varepsilon_{ji} &= \frac{\partial s_j}{\partial w_{ji}} && (2.42)\\
&= \frac{\partial}{\partial w_{ji}} \left(g_{jj}\, w_{jj}\, \widehat{s_j} + \sum_{i' \neq j} g_{ji'} w_{ji'} y_{i'} \right) && (2.43)\\
&= g_{jj}\, w_{jj}\, \widehat{\varepsilon_{ji}} + g_{ji}\, y_i && (2.44)
\end{aligned}$$

Working on the extended eligibility trace ε^k_{ji}, Equation 2.46 again expands s_k from Equation 2.45. When performing the partial derivative on the first term, g_{kk} may be equal to y_j, which depends on w_{ji}; also, $\widehat{s_k}$ depends on w_{ji} via its effect on s_j, requiring the product rule to derive the first term. For terms in the sum, unit j can gate the connection from j' to k, but $k \neq j$ so $w_{kj'}$ can be treated as constant, as can $y_{j'}$ since this part of the derivation is not concerned with connections that j projects forward. In Equation 2.47, what remains is the result of the product rule and the remaining terms of the sum, where the a index runs over only those connections into k that j does in fact gate. Equation 2.48 shows that the first partial derivative is simply the previous value of the extended eligibility trace. Equation 2.49

(cf. Equation 2.18) pulls out partial derivatives common to the latter two terms and replaces them with the names of their stored variable forms. The partial derivatives in the sum are all 1 by the definition of the a index. The remaining partial derivative inside the parentheses reduces to 1 when j gates k's self-connection and 0 otherwise.

$$\begin{aligned}
\varepsilon_{ji}^{k} &= \frac{\partial s_k}{\partial w_{ji}} && (2.45)\\
&= \frac{\partial}{\partial w_{ji}}\left(g_{kk}\, w_{kk}\, \widehat{s_k} + \sum_{j' \neq k} g_{kj'} w_{kj'} y_{j'}\right) && (2.46)\\
&= g_{kk} w_{kk}\, \frac{\partial \widehat{s_k}}{\partial w_{ji}} + \frac{\partial g_{kk}}{\partial w_{ji}} w_{kk}\, \widehat{s_k} + \sum_{a \neq k} \frac{\partial g_{ka}}{\partial w_{ji}} w_{ka} y_a && (2.47)\\
&= g_{kk}\, w_{kk}\, \widehat{\varepsilon_{ji}^{k}} + \frac{\partial y_j}{\partial s_j}\frac{\partial s_j}{\partial w_{ji}}\left(\frac{\partial g_{kk}}{\partial y_j} w_{kk}\, \widehat{s_k} + \sum_{a \neq k} \frac{\partial g_{ka}}{\partial y_j} w_{ka} y_a\right) && (2.48)\\
&= g_{kk}\, w_{kk}\, \widehat{\varepsilon_{ji}^{k}} + f_j'(s_j)\, \varepsilon_{ji}\, \left(\frac{\partial g_{kk}}{\partial y_j} w_{kk}\, \widehat{s_k} + \sum_{a \neq k} w_{ka}\, y_a\right) && (2.49)
\end{aligned}$$

2.5 LSTM as a special case of LSTM-g

The following shows that LSTM is a special case of LSTM-g in the sense that when LSTM-g is applied to a particular class of network architectures, the two algorithms produce identical weight changes. For notational simplicity, this derivation addresses the case where the target LSTM architecture has only one memory cell per block; this explanation can be trivially extended to the case where the architecture has multiple memory cells per block, though this unnecessarily complicates the notation. The input layer projects weighted connections forward to four distinct layers of units (see

Figure 2.2), which are activated in the following order during a time-step, just as in LSTM: the input gate layer ι, the forget gate layer φ, the memory cell layer c, and the output gate layer ω. Each of these layers has the same number of units; a group of parallel units are associated via the pattern of network connectivity and collectively function like an LSTM memory block. Inputs to each cell in the memory cell layer are gated by the associated input gate unit. The memory cell layer is the only layer in which each unit has a direct self-connection; these each have a fixed weight of 1 and are gated by the appropriate forget gate. A final output layer receives weighted connections from the memory cell layer, which are gated by the output gates.

With this simple LSTM network, LSTM-g produces the same weight changes as LSTM. On a similar architecture augmented with peephole connections, the error responsibilities would differ, as LSTM-g is able to use error back-propagated from the output gates across these connections, whereas LSTM does not. This is discussed further in Section 2.6.

2.5.1 State and activation equivalence

The following demonstrates the equivalence of the activation dynamics of LSTM and LSTM-g when the latter is applied to the standard LSTM architecture.

2.5.1.1 Gate units

The first goal is to show that the activation for each gate unit (i.e., for a general unit λ_j where $\lambda \in \{\iota, \varphi, \omega\}$) is the same here as it is in LSTM. Starting from the LSTM-g state definition (Equation 2.50, cf. Equation 2.15), Equation 2.51 gets rid of the first term because the activations of the gate units are stateless due to their lack of self connections, meaning $w_{\lambda_j \lambda_j} = 0$. On this architecture, the only connections into any of the gate units come from the input layer and are ungated, so all the $g_{\lambda_j i}$ terms are 1 (Equation 2.52). What remains is the definition of the net input to a normal LSTM unit (Equation 2.53, cf. Equation 2.1).

$$s_{\lambda_j} = g_{\lambda_j \lambda_j}\, w_{\lambda_j \lambda_j}\, \widehat{s_{\lambda_j}} + \sum_{i \neq j} g_{\lambda_j i}\, w_{\lambda_j i}\, y_i \tag{2.50}$$

$$= \sum_{i \neq j} g_{\lambda_j i}\, w_{\lambda_j i}\, y_i \tag{2.51}$$

$$= \sum_{i \neq j} w_{\lambda_j i}\, y_i \tag{2.52}$$

$$= x_{\lambda_j} \text{ for } \lambda \in \{\iota, \varphi, \omega\} \tag{2.53}$$

2.5.1.2 Output units

The situation is similar for the output units θ_j. Starting from the same LSTM-g state equation (Equation 2.54, cf. Equation 2.15), the first term drops out due to lack of output unit self-connections (Equation 2.55). Since the connections into the output units come from the memory cells, each connection is gated by its corresponding output gate (Equation 2.56). Equation 2.57 regroups the terms, and Equation 2.58 substitutes the LSTM memory cell activation for the output gate activation times the LSTM-g memory cell activation (by Equation 2.4). The result is equal to the net input of an LSTM output unit (Equation 2.59, cf. Equation 2.1).

$$s_{\theta_j} = g_{\theta_j \theta_j}\, w_{\theta_j \theta_j}\, \widehat{s_{\theta_j}} + \sum_{i \in c} g_{\theta_j i}\, w_{\theta_j i}\, y_i \tag{2.54}$$

$$= \sum_{i \in c} g_{\theta_j i}\, w_{\theta_j i}\, y_i \tag{2.55}$$

$$= \sum_{i \in c} y_{\omega_i}\, w_{\theta_j i}\, y_i \tag{2.56}$$

$$= \sum_{i \in c} w_{\theta_j i}\, (y_{\omega_i}\, y_i) \tag{2.57}$$

$$= \sum_{i \in c} w_{\theta_j i}\, y_{c_i} \tag{2.58}$$

$$= x_{\theta_j} \tag{2.59}$$

The preceding proves the equivalence of LSTM's x_{λ_j} and s_{λ_j} in LSTM-g for all units except memory cells. It follows directly for these units that the activation y_{λ_j} in LSTM is equivalent to the quantity of the same name in LSTM-g (Equation 2.60, cf. Equation 2.16), assuming equivalent squashing functions f_{λ_j} (Equation 2.61, cf. Equation 2.2).

$$y_{\lambda_j} = f_{\lambda_j}(s_{\lambda_j}) \tag{2.60}$$

$$= f_{\lambda_j}(x_{\lambda_j}) \text{ for } \lambda \in \{\iota, \varphi, \omega, \theta\} \tag{2.61}$$

2.5.1.3 Memory cells

The next goal is to demonstrate the equivalence of memory cell states in LSTM and LSTM-g. Starting again from the unit state equation for LSTM-g (Equation 2.62, cf. Equation 2.15), Equation 2.63 substitutes the self-connection weight with its value of 1 and replaces the self-connection gate with the associated forget gate. Moving to Equation 2.64 requires recognizing that all of the connections coming into the memory cell in question are gated by the memory cell's associated input gate, so $g_{c_j i} = y_{\iota_j} \forall i$ and the term can move outside the sum. The remaining sum is equal to the net (ungated) input to the memory cell, resulting in the equation for LSTM memory cell states (Equation 2.65, cf. Equation 2.3).

$$\begin{aligned} s_{c_j} &= g_{c_j c_j}\, w_{c_j c_j}\, \widehat{s_{c_j}} + \sum_{i \neq c_j} g_{c_j i}\, w_{c_j i}\, y_i && (2.62) \\ &= y_{\varphi_j}\, \widehat{s_{c_j}} + \sum_{i \neq c_j} g_{c_j i}\, w_{c_j i}\, y_i && (2.63) \\ &= y_{\varphi_j}\, \widehat{s_{c_j}} + y_{\iota_j} \sum_{i \neq c_j} w_{c_j i}\, y_i && (2.64) \\ &= y_{\varphi_j}\, \widehat{s_{c_j}} + y_{\iota_j}\, x_{c_j} && (2.65) \end{aligned}$$

The activation variable y_{c_j} does not line up directly in LSTM-g and LSTM, since LSTM requires the output gate to modulate the activation of the memory cell directly, while LSTM-g defers the modulation until the activation is passed on through a gated connection. The distinction has already been noted and appropriately dealt with in the discussion of Equation 2.57 and is not problematic for the proof at hand.

2.5.2 Weight change equivalence

The previous section showed the equivalence of activation dynamics when LSTM-g is used on the LSTM architecture. The following section shows the equivalence of the weight changes performed by each algorithm.

2.5.2.1 Output units

Since the output units in LSTM-g get the same error responsibility from the environment as in LSTM, the proof need only consider whether each connection in question has the same eligibility trace in the two schemes; proving this will trivially show weight change equivalence. Only the general LSTM-g eligibility trace equation is relevant, as the output units perform no gating (Equation 2.66, cf. Equation 2.17). The first term drops out because the output units have no self-connections (Equation 2.67). Since the connection in question is from the memory cell layer, the gating term becomes the output gate activation (Equation 2.68). The two remaining factors are equal to LSTM's definition of the memory cell activation (Equation 2.69), which is consistent with LSTM using only the sending unit's activation as an output unit eligibility trace (Equation 2.70, cf. Equation 2.5).

$$\varepsilon_{ji} = g_{jj}\, w_{jj}\, \widehat{\varepsilon_{ji}} + g_{ji}\, y_i \tag{2.66}$$

$$= g_{ji}\, y_i \tag{2.67}$$

$$= y_{\omega_i}\, y_i \tag{2.68}$$

$$= y_{c_i} \tag{2.69}$$

$$= \varepsilon_{\theta_{ji}} \tag{2.70}$$

2.5.2.2 Output gates

To prove equivalent changes to weights into the output gate units, Equation 2.71 begins with the generalized eligibility trace equation from LSTM-g (cf. Equation 2.17). Output gate units have no self-connections, so the first term drops out (Equation 2.72). The incoming connections to the output gate are not gated, so the gating term is 1 (Equation 2.73). The result is the most recent activity of the sending unit on this connection, which is the same eligibility trace as in LSTM (Equation 2.74, cf. Equation 2.5).

$$\varepsilon_{ji} = g_{jj}\, w_{jj}\, \widehat{\varepsilon_{ji}} + g_{ji}\, y_i \tag{2.71}$$

$$= g_{ji}\, y_i \tag{2.72}$$

$$= y_i \tag{2.73}$$

$$= \varepsilon_{\omega_{ji}} \tag{2.74}$$

Next, Equation 2.75 begins with the extended eligibility traces for output gates, starting from LSTM-g (cf. Equation 2.18). Output gates only modulate connections to the output layer, implying $k \in \theta$, and none of these units have self-connections; thus, the first term outside the parentheses drops, as does the first term inside the parentheses (Equation 2.76). Each output gate will modulate only a single connection into each output unit—the connection from its associated memory cell to the output unit in question—so the sum reduces to a single term (Equation 2.77).

$$\varepsilon_{ji}^{k} = g_{kk}\, w_{kk}\, \widehat{\varepsilon_{ji}^{k}} + f_j'(s_j)\, \varepsilon_{ji} \left(\frac{\partial g_{kk}}{\partial y_j} w_{kk}\, \widehat{s_k} + \sum_{a \neq k} w_{ka}\, y_a \right) \tag{2.75}$$

$$= f_j'(s_j)\, \varepsilon_{ji} \sum_{a \neq k} w_{ka}\, y_a \tag{2.76}$$

$$= f_j'(s_j)\, \varepsilon_{ji}\, w_{\theta_k c_j}\, y_{c_j} \tag{2.77}$$

Finally, starting from the LSTM-g weight change (Equation 2.78, cf. Equation 2.24), simplification reveals the equation for LSTM weight change. Equation 2.79 shows that for output gates, which project no weighted connections, the set P_j is empty, making δ_{P_j} zero and eliminating the first term. For output gates, the set G_j is precisely the set of output units θ; thus, Equation 2.80 replaces the eligibility trace term with the simplified version from Equation 2.77, moving common terms in the sum to the outside. Rearranging the equation reveals a term (in parentheses, Equation 2.81) that is equal to LSTM's formulation of the error responsibility for an output gate

(Equation 2.11). Making that replacement and the replacement of the eligibility trace (Equation 2.82) produces the exact weight change prescribed by LSTM (Equation 2.83, cf. Equation 2.13).

$$
\begin{aligned}
\Delta w_{ji} &= \alpha\, \delta_{P_j}\, \varepsilon_{ji} + \alpha \sum_{k \in G_j} \delta_k\, \varepsilon_{ji}^{k} && (2.78)\\
&= \alpha \sum_{k \in G_j} \delta_k\, \varepsilon_{ji}^{k} && (2.79)\\
&= \alpha\, f'_{\omega_j}(s_{\omega_j})\, \varepsilon_{ji}\, y_{c_j} \sum_{\theta_k} \delta_{\theta_k}\, w_{\theta_k c_j} && (2.80)\\
&= \alpha \left(f'_{\omega_j}(s_{\omega_j})\, y_{c_j} \sum_{\theta_k} \delta_{\theta_k}\, w_{\theta_k c_j} \right) \varepsilon_{ji} && (2.81)\\
&= \alpha\, \delta_{\omega_j}\, \varepsilon_{\omega_{ji}} && (2.82)\\
&= \Delta w_{\omega_{ji}} && (2.83)
\end{aligned}
$$

2.5.2.3 Memory cells

As with the output units, the proof for the memory cells need only consider the basic eligibility trace (Equation 2.84, cf. Equation 2.17), since the memory cells perform no gating functions. Equation 2.85 notes that the self-connection weight is 1, and the self-connection gate is the associated forget gate. Equation 2.86 shows that the input connection must be gated by the appropriate input gate, revealing precisely the form of the eligibility trace from LSTM (Equation 2.87, cf. Equation 2.6).

$$
\begin{aligned}
\varepsilon_{ji} &= g_{jj}\, w_{jj}\, \widehat{\varepsilon_{ji}} + g_{ji}\, y_i && (2.84)\\
&= y_{\varphi_j}\, \widehat{\varepsilon_{ji}} + g_{ji}\, y_i && (2.85)\\
&= y_{\varphi_j}\, \widehat{\varepsilon_{ji}} + y_{\iota_j}\, y_i && (2.86)\\
&= \varepsilon_{c_{ji}} && (2.87)
\end{aligned}
$$

Again due to the lack of gating, the proof need only consider $\delta_j = \delta_{P_j}$ (Equation 2.88, cf. Equation 2.21). Recognizing that all connections forward are gated by the associated output gate, Equation 2.89 replaces all the g_{kj} terms in the sum with y_{ω_j}, making this a factor outside the sum. Equation 2.90 simply renames the derivative and weight terms based on the architecture, recovering δ_{c_j} from LSTM (Equation 2.91, cf. Equation 2.12).

$$
\begin{aligned}
\delta_{P_j} &= f'_j(s_j) \sum_{k \in P_j} \delta_k\, g_{kj}\, w_{kj} && (2.88)\\
&= f'_j(s_j)\, y_{\omega_j} \sum_{k \in P_j} \delta_k\, w_{kj} && (2.89)\\
&= f'_{c_j}(s_{c_j})\, y_{\omega_j} \sum_{k \in \theta} \delta_{\theta_k}\, w_{\theta_k c_j} && (2.90)\\
&= \delta_{c_j} && (2.91)
\end{aligned}
$$

Starting with the weight change equation from LSTM-g (Equation 2.92, cf. Equation 2.24), Equation 2.93 removes the second term since G_j is empty, and Equation 2.94 substitutes the equal quantities derived earlier to reproduce LSTM's weight change for memory cells (Equation 2.95, cf. Equation 2.13).

$$\Delta w_{ji} = \alpha\, \delta_{P_j}\, \varepsilon_{ji} + \alpha \sum_{k \in G_j} \delta_k\, \varepsilon_{ji}^k \tag{2.92}$$

$$= \alpha\, \delta_{P_j}\, \varepsilon_{ji} \tag{2.93}$$

$$= \alpha\, \delta_{c_j}\, \varepsilon_{c_{ji}} \tag{2.94}$$

$$= \Delta w_{c_{ji}} \tag{2.95}$$

2.5.2.4 Forget gates

Similarly for forget gates, the general LSTM-g eligibility trace (Equation 2.96, cf. Equation 2.17) can be simplified by noting that the forget gates themselves do not self-connect (Equation 2.97), and that the inputs are not gated (Equation 2.98).

$$\varepsilon_{ji} = g_{jj}\, w_{jj}\, \widehat{\varepsilon_{ji}} + g_{ji}\, y_i \tag{2.96}$$

$$= g_{ji}\, y_i \tag{2.97}$$

$$= y_i \tag{2.98}$$

The extended eligibility trace (Equation 2.99, cf. Equation 2.18) can be simplified by noting first that the only relevant k is the single memory cell whose self-connection this unit gates, which suggests rewriting the first gate term as the forget gate activation. In the parentheses, the derivative term and w_{kk} both go to 1, and since there are no non-self-connections that this unit gates, the sum is empty (Equation 2.100). Reorganizing and renaming some terms—as well as replacing the eligibility trace as calculated in Equation 2.98—reveals the form in Equation 2.101. The inductive assumption that the eligibility trace from the previous time-step, $\widehat{\varepsilon^{c_j}_{ji}}$, is equal to the previous LSTM eligibility trace, $\widehat{\varepsilon_{\varphi_{ji}}}$ (Equation 2.102), reveals that the eligibility traces are the same for the current step as well (Equation 2.103, cf. Equation 2.7).

$$\varepsilon^{k}_{ji} = g_{kk}\, w_{kk}\, \widehat{\varepsilon^{k}_{ji}} + f'_j(s_j)\, \varepsilon_{ji} \left(\frac{\partial g_{kk}}{\partial y_j} w_{kk}\, \widehat{s_k} + \sum_{a \neq k} w_{ka}\, y_a \right) \tag{2.99}$$

$$\varepsilon^{c_j}_{ji} = y_{\varphi_j}\, \widehat{\varepsilon^{c_j}_{ji}} + f'_j(s_j)\, \varepsilon_{ji}\, \widehat{s_{c_j}} \tag{2.100}$$

$$= y_{\varphi_j}\, \widehat{\varepsilon^{c_j}_{ji}} + \widehat{s_{c_j}}\, f'_{\varphi_j}(x_{\varphi_j})\, y_i \tag{2.101}$$

$$= y_{\varphi_j}\, \widehat{\varepsilon_{\varphi_{ji}}} + \widehat{s_{c_j}}\, f'_{\varphi_j}(x_{\varphi_j})\, y_i \tag{2.102}$$

$$= \varepsilon_{\varphi_{ji}} \tag{2.103}$$

As it was for the output gates, δ_{P_j} is zero (Equation 2.104, cf. Equation 2.24), since P_j is empty for the forget gates, resulting in Equation 2.105. Here, G_j contains only the memory cell whose self-connection this unit gates (Equation 2.106). As shown earlier, the eligibility traces are equal (Equation 2.107), and $\delta_{c_j} = \delta_{\varphi_j}$ as shown in Equation 2.12, revealing LSTM's weight change for forget gates (Equation 2.108, cf. Equation 2.13).

$$\Delta w_{ji} = \alpha\, \delta_{P_j}\, \varepsilon_{ji} + \alpha \sum_{k \in G_j} \delta_k\, \varepsilon_{ji}^{k} \tag{2.104}$$

$$= \alpha \sum_{k \in G_j} \delta_k\, \varepsilon_{ji}^{k} \tag{2.105}$$

$$= \alpha\, \delta_{c_j}\, \varepsilon_{ji}^{c_j} \tag{2.106}$$

$$= \alpha\, \delta_{\varphi_j}\, \varepsilon_{\varphi_{ji}} \tag{2.107}$$

$$= \Delta w_{\varphi_{ji}} \tag{2.108}$$

2.5.2.5 Input gates

For input gates, as for forget gates, the LSTM-g eligibility trace (Equation 2.109, cf. Equation 2.17) is simplified by dropping the first term, since input gates do not self-connect (Equation 2.110), and letting g_{ji} go to 1, since the inputs are not gated (Equation 2.111).

$$\varepsilon_{ji} = g_{jj}\, w_{jj}\, \widehat{\varepsilon_{ji}} + g_{ji}\, y_i \tag{2.109}$$

$$= g_{ji}\, y_i \tag{2.110}$$

$$= y_i \tag{2.111}$$

Equation 2.112 again begins with the extended eligibility trace (cf. Equation 2.18), where the only k in play is the single memory cell whose input connections are modulated by this input gate. Thus, the first gain term can be replaced by the forget gate activation, and the derivative term inside the parentheses goes to zero (Equation 2.113). Next, Equation 2.114 renames the f'_j term appropriately, replaces the eligibility trace with its simplification from Equation 2.111, and recognizes that the remaining sum is in fact the ungated net input to the memory cell in question. Under the inductive assumption that the previous step's eligibility traces are equal (Equation 2.115), those of the current step are equal as well (Equation 2.116, cf. Equation 2.8).

$$\varepsilon_{ji}^{k} = g_{kk}\, w_{kk}\, \widehat{\varepsilon_{ji}^{k}} + f'_j(s_j)\, \varepsilon_{ji} \left(\frac{\partial g_{kk}}{\partial y_j} w_{kk}\, \widehat{s_k} + \sum_{a \neq k} w_{ka}\, y_a \right) \tag{2.112}$$

$$\varepsilon_{ji}^{c_j} = y_{\varphi_j}\, \widehat{\varepsilon_{ji}^{c_j}} + f'_j(s_j)\, \varepsilon_{ji} \sum_{a \neq c_j} w_{c_j a}\, y_a \tag{2.113}$$

$$= y_{\varphi_j}\, \widehat{\varepsilon_{ji}^{c_j}} + f'_{\iota_j}(x_{\iota_j})\, y_i\, x_{c_j} \tag{2.114}$$

$$= y_{\varphi_j}\, \widehat{\varepsilon_{\iota_j i}} + x_{c_j}\, f'_{\iota_j}(x_{\iota_j})\, y_i \tag{2.115}$$

$$= \varepsilon_{\iota_j i} \tag{2.116}$$

Starting again with the unified LSTM-g weight update equation (Equation 2.117, cf. Equation 2.24), Equation 2.118 drops the first term since δ_{P_j} is zero. Equation 2.119 reflects the fact that the only cell in the set G_j is the associated memory cell c_j of this input gate, simplifying the sum to a single term. Using Equation 2.12, Equation 2.120 can be obtained by replacing δ_{c_j} with δ_{ι_j} (which is equivalent for this network), as well as replacing $\varepsilon_{ji}^{c_j}$ according to Equation 2.116. The result is precisely the weight change specified by LSTM for input gate connections (Equation 2.121, cf. Equation 2.13).

$$\begin{aligned}
\Delta w_{ji} &= \alpha\,\delta_{P_j}\,\varepsilon_{ji} + \alpha \sum_{k \in G_j} \delta_k\,\varepsilon_{ji}^{k} && (2.117)\\
&= \alpha \sum_{k \in G_j} \delta_k\,\varepsilon_{ji}^{k} && (2.118)\\
&= \alpha\,\delta_{c_j}\,\varepsilon_{ji}^{c_j} && (2.119)\\
&= \alpha\,\delta_{\iota_j}\,\varepsilon_{\iota_{ji}} && (2.120)\\
&= \Delta w_{\iota_{ji}} && (2.121)
\end{aligned}$$

This section has shown that every unit in a canonical LSTM network architecture calculates identical activations and weight updates under LSTM and LSTM-g, thus concluding the proof that LSTM is a special case of LSTM-g.

2.6 Comparison of LSTM and LSTM-g

As stated in Section 2.3 and proved in Section 2.5, an LSTM-g network with the same architecture as an LSTM network will produce the same weight changes as LSTM training would, provided that peephole connections are not present. When peephole connections are added to the LSTM architecture, however, LSTM-g utilizes a source of error that LSTM training neglects: error responsibilities back-propagated from the output gates across the peephole connections to the associated memory cells and beyond. To demonstrate this, the next paragraphs calculate the error responsibility for a memory cell j in an LSTM-g network and compare the answer to the error responsibility for that same unit as prescribed by LSTM training.

Beginning with the generic error responsibility equation from LSTM-g (Equation 2.23), the cell in question is the architectural equivalent of a memory cell, and as such, it performs no gating functions. Thus, the set of gated cells G_j is empty and δ_{G_j} is zero, leaving δ_{P_j} alone as the error responsibility. Equation 2.122 substitutes Equation 2.21 for δ_{P_j}. The constituents of P_j can be determined by recalling that the memory cell in question projects connections to all the output units and sends peephole connections to its controlling input gate, forget gate, and output gate. From this set of receiving units, only the output units and the output gate are downstream from the memory cell, so they comprise P_j. Taking each type of unit in P_j individually, Equation 2.123 expands the sum. Finally, Equation 2.14 shows that, because the peephole connection to the output gate is not gated, the $g_{\omega_j c_j}$ term goes to 1; in addition, all the output connections are gated by the output gate, so every $g_{\theta_k c_j}$ term becomes y_{ω_j}, and the term can move outside the sum. The resulting Equation 2.124 should be equal to δ_{c_j} as shown in Equation 2.125 (derived from Equation 2.12) to make LSTM-g equivalent to LSTM training in this case.

$$\delta_j = \delta_{P_j} = f'_{c_j}(s_{c_j}) \sum_{k \in P_j} \delta_k \, g_{kj} \, w_{kj} \tag{2.122}$$

$$= f'_{c_j}(s_{c_j}) \left(\sum_{k \in \theta} \delta_{\theta_k} \, g_{\theta_k c_j} \, w_{\theta_k c_j} + \delta_{\omega_j} \, g_{\omega_j c_j} \, w_{\omega_j c_j} \right) \tag{2.123}$$

$$= f'_{c_j}(s_{c_j}) \left(y_{\omega_j} \sum_{k \in \theta} \delta_{\theta_k} \, w_{\theta_k c_j} + \delta_{\omega_j} \, w_{\omega_j c_j} \right) \tag{2.124}$$

$$\delta_{c_j} = f'_{c_j}(s_{c_j}) \, y_{\omega_j} \sum_{k \in \theta} \delta_{\theta_k} \, w_{\theta_k c_j} \tag{2.125}$$

Upon inspection, it becomes clear that LSTM-g includes a bit of extra back-propagated error—as indicated by the term $\delta_{\omega_j} \, w_{\omega_j c_j}$—originating from the output gate. Besides giving a more accurate weight update for connections into memory cell j, this change in error will be captured in δ_j and passed upstream to the forget gates and input gates. As demonstrated in Section 2.7, this extra information helps LSTM-g perform slightly better than the original algorithm on an LSTM architecture with peephole connections.

2.7 Experiments

Examining the effectiveness of LSTM-g involves performing a number of experimental comparisons using various neural network architectures trained with either the original LSTM algorithm or the LSTM-g algorithm from Section 2.3, as detailed in the following sections.

2.7.1 Distracted sequence recall on the standard architecture

The first set of experiments trains different neural networks on a task called the **distracted sequence recall** task. This task is a variation of the **temporal order** task, which is arguably the most challenging task demonstrated by Hochreiter and Schmidhuber (1997). The distracted sequence recall task involves 10 symbols, each represented locally by a single active unit in an input layer of 10 units: 4 **target** symbols, which must be recognized and remembered by the network, 4 **distractor** symbols, which never need to be remembered, and 2 **prompt** symbols, which direct the network to give an answer. A single trial consists of a presentation of a temporal sequence of 24 input symbols. The first 22 consist of 2 randomly chosen target symbols and 20 randomly chosen distractor symbols, all in random order; the remaining 2 symbols are the prompts, which direct the network to produce the first and second targets in the sequence, in order, regardless of when they occurred. The targets may appear at any point in the sequence, so the network cannot rely on their temporal position as a cue; rather, the network must recognize the symbols as targets and preferentially save them, along with temporal order information, to produce the correct output sequence. The network is trained to produce no output for all symbols except the prompts, and for each prompt symbol, the network must produce the output symbol that corresponds to the appropriate target from the sequence.

The major difference between the temporal order task and the distracted sequence recall task is as follows. In the former, the network is required to activate one of 16 output units, each of which represents a possible ordered sequence of both target symbols. In contrast, the latter task requires the network to activate one of only 4 output units, each representing a single target symbol; the network must activate the correct output unit for each of the targets, in the same order they were observed. Requiring the network to produce outputs in sequence adds a layer of difficulty; however, extra generalization power may be imparted by the fact that the network is now using the same output weights to indicate the presence of a target, regardless of its position in the sequence. Because the temporal order task was found to be unsolvable by known architectures other than the LSTM architecture when trained by gradient descent-based methods (Hochreiter and Schmidhuber, 1997), the comparisons here do not include methods other than LSTM and LSTM-g. Further, the comparisons do not include second-order networks trained with BPTT, RTRL, or other such methods, restricting the focus to spatially and temporally local training methods.

The first experiment examines the impact of the extra error information utilized by LSTM-g and involves training two types of networks on the distracted sequence recall task. The first network serves as a control and is a typical LSTM network with forget gates, peephole connections, and direct input-to-output connections (see Figure 2.2), and is trained by the LSTM algorithm. The second network has the same architecture as the first, but is trained by the LSTM-g algorithm, allowing it to take advantage of back-propagated peephole connection error terms.

All runs of each network use the same basic approach and parameters: a single unit in the input layer for each of the 10 input symbols, 8 units in the memory cell layer and associated gate layers, 4 units in the output layer for the target symbols, and a learning rate $\alpha = 0.1$. Both networks are augmented with peephole connections and direct input-to-output connections. Thus, both algorithms are training networks with 416 weights. Networks are allowed to train on random instances of the distracted sequence recall task until they achieve the performance criterion of 95% accuracy on a test set of 1000 randomly selected sequences that the network had never encountered during training. To get credit for processing a sequence correctly, the network is required to keep all output units below an activation level of 0.5 during all symbols in the sequence except prompts, activating only the correct target symbol in these cases—meaning that all units must have activations on the same side of 0.5 as the target associated with the prompt. This correctness criterion is used both for networks trained by the LSTM algorithm and for those trained by the LSTM-g algorithm to ensure that the two types are compared on an equal footing. Each network type is run 50 times, using randomized initial weights in the range $[-0.1, 0.1)$.

Figure 2.3 compares the number of trials required to train the two networks. All runs of both networks reach the performance criterion as ex-

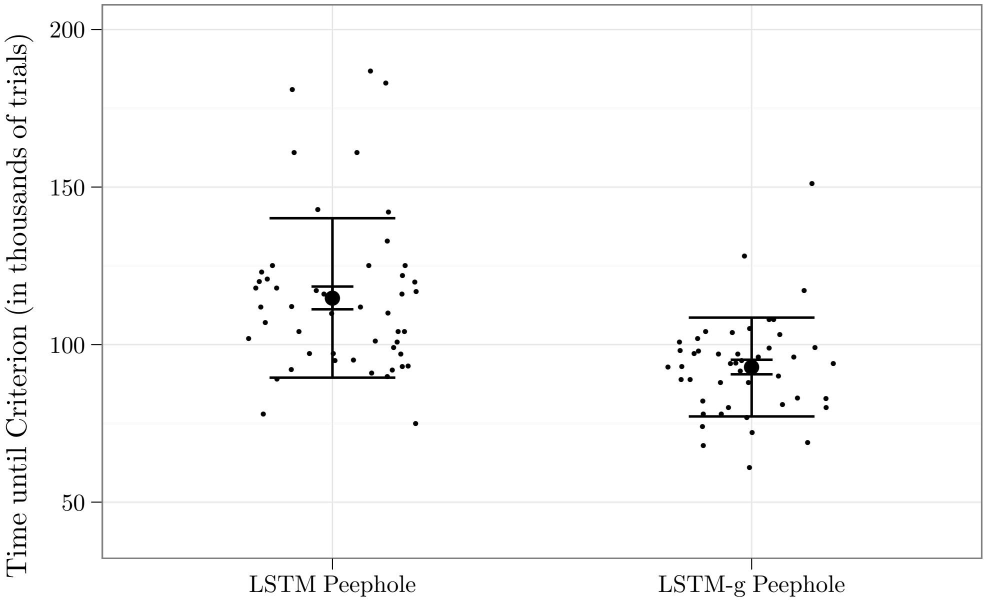


Figure 2.3: Plot of the results of an experiment that pitted LSTM-g against LSTM, each training an identical standard peephole LSTM architecture to perform the distracted sequence recall task. Small points are individual network runs, jittered to highlight their density. The large black point for each network type is the mean over all 50 runs, with standard error (small bars) and standard deviation (large bars). The results show that LSTM-g's utilization of extra error information is clearly beneficial.

pected, but there are differences in how quickly they achieve this. In particular, LSTM-g is able to train the LSTM network architecture significantly faster than the original algorithm (as evaluated by a Welch two-sample t-test after removing outliers greater than 2 standard deviations from the sample mean, with $t \approx 5.1, df \approx 80.9, p < 10^{-5}$). This demonstrates that LSTM-g can provide a clear advantage over LSTM in terms of the amount of training required, even on an LSTM-compatible architecture.

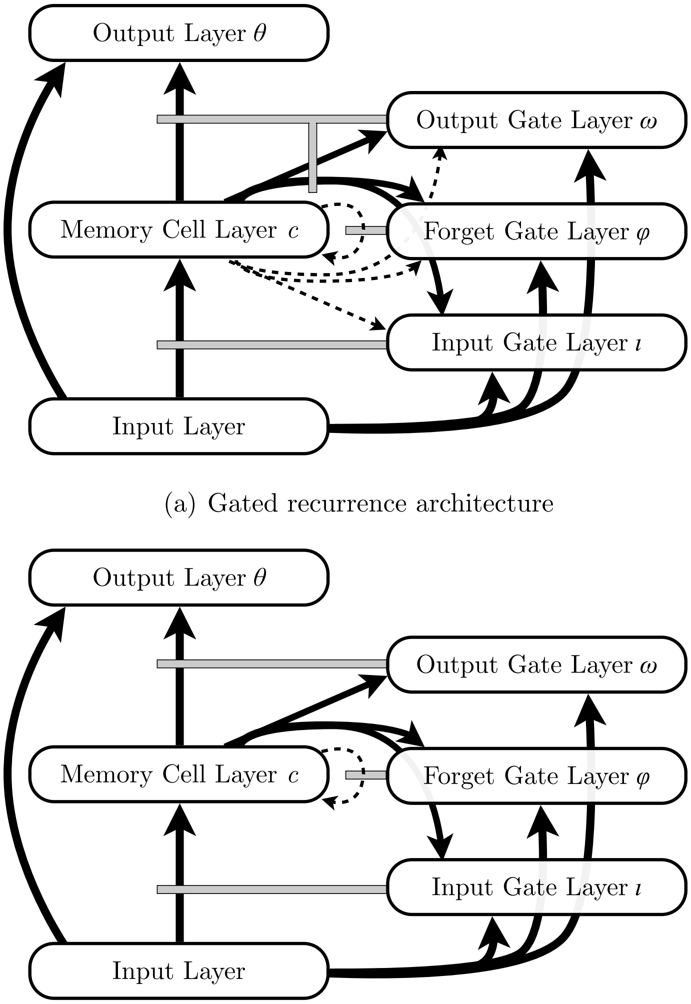
2.7.2 Distracted sequence recall on a customized architecture

The second experiment investigates the relative performance of the standard LSTM architecture compared to other network architectures that would require modifications to the LSTM training paradigm. This experiment involves training three additional types of networks on the same distracted sequence recall task. The first network serves as the control and utilizes the LSTM algorithm to train a standard LSTM architecture that is the same as in the previous experiment except for the addition of recurrent connections from all (output-gated) memory cells to all the gate units. This architecture is termed **gated recurrence architecture** and is depicted in Figure 2.4(a). The second network also uses the gated recurrence architecture but is trained by LSTM-g. The third network is a new **ungated recurrence architecture**, shown in Figure 2.4(b), which starts with the standard LSTM architecture and adds direct, ungated connections from each memory cell to all gate units. These connections come from the ungated memory cell output like peephole connections would, but unlike peephole connections, these are projected to gate units both inside and outside of the local memory block. The intuition behind this architecture comes from the idea that a memory cell should be able to communicate its contents not only to its controlling gates but also to the other memory blocks in the hidden layer, while still hiding these contents from downstream units. Such communication would intuitively be a major boon for sequential storage and retrieval tasks because it allows a memory block to choose what to store based on what is already stored in other blocks, even if the contents of those blocks are not yet ripe for consideration in calculating the network output. These ungated cell-to-gate connections are a direct generalization of peephole connections, but the new architecture that results could only be trained by the LSTM algorithm if it were modified to suit the architecture. As such, the following presents only the results of training the ungated recurrence architecture with LSTM-g, which requires no special treatment of these ungated cell-to-gate connections.

Each of the three networks uses the same approach and parameters as in the previous experiment. This means both types of networks using the gated recurrence architecture have 608 trainable connections, and the ungated recurrence architecture has only 584 because the 24 peephole connections used in the gated recurrence architecture would be redundant.[2]

Though these networks are more complex than those in the first experiment, they are able to learn the task more quickly. Figure 2.5 shows that for each run, the number of trials necessary before each of three networks reaches the performance criterion. Again, LSTM-g appears to have a slight speed advantage over LSTM when applied to the LSTM-compatible gated

[2] The experiments reported here were also run with a variant of the gated recurrence architecture without the 24 peephole connections, leaving it with the same 584 weights as the ungated recurrence architecture; however, the lack of peephole connections caused a severe learning slowdown. In the interest of comparing LSTM-g against the strongest possible control, the following paragraphs report only the results from the gated recurrence architecture with peephole connections as described above.



(a) Gated recurrence architecture

(b) Ungated recurrence architecture

Figure 2.4: The network architectures used in the second experiment (c.f. Figure 2.2). In (a), the previous LSTM architecture is augmented with a full complement of recurrent connections from each memory cell to each gate, regardless of memory block associations; all these connections are gated by the appropriate output gate. In (b), the architecture's peephole connections are replaced with a full complement of ungated connections from each memory cell to every gate. This second architectural variant is incompatible with the LSTM training algorithm, as it requires all connections out of the memory cell layer to be gated by the output gate. The network can still be trained by LSTM-g, however.

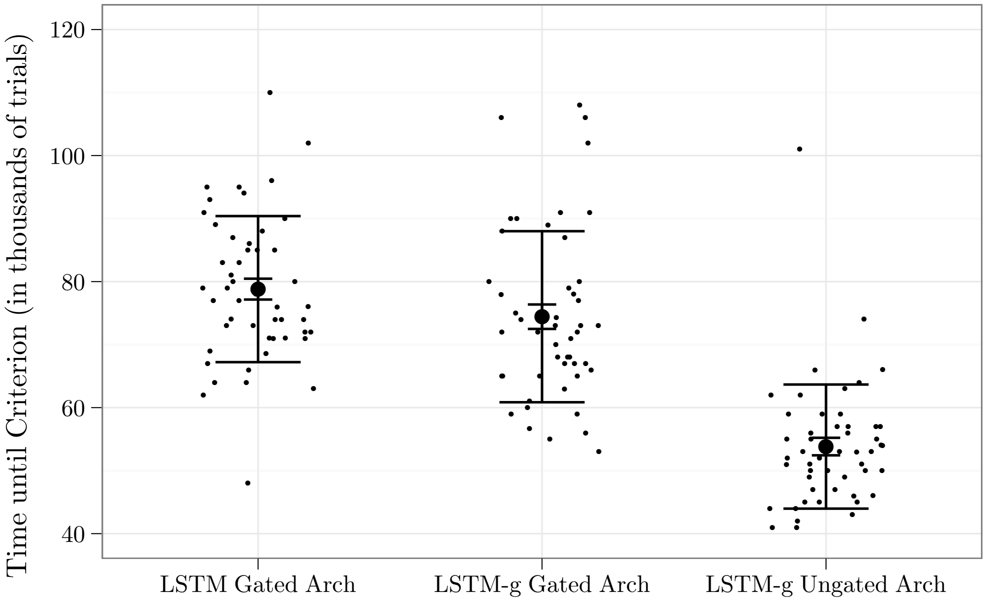


Figure 2.5: Plot of the results on the distracted sequence recall task for three networks: an LSTM network augmented with peephole connections and gated recurrent connections from all memory cells to all gates, trained by either LSTM or LSTM-g; and an LSTM-g network with ungated recurrent connections from all memory cells to all gates. The ungated recurrence network, trainable only with LSTM-g, clearly reaches the performance criterion after less training than the comparable gated recurrence network as trained by either LSTM or LSTM-g.

recurrence architecture, though this difference narrowly misses statistical significance ($t \approx 1.8, df \approx 87.9, p < 0.08$). More interesting is the improvement that LSTM-g achieves on the novel ungated recurrence architecture, which reaches significance easily compared to both the gated recurrence architecture trained with LSTM ($t \approx 15.3, df \approx 79.6, p < 10^{-15}$) and with LSTM-g ($t \approx 10.2, df \approx 69.6, p < 10^{-14}$). LSTM-g is able to train the ungated recurrence architecture faster than either it or LSTM can train the gated recurrence architecture. This difference illustrates the potential benefits of the wide range of customized architectures that LSTM-g can train.

2.7.3 Language recognition on a two-stage architecture

The final experiment adopts a more involved and substantially different **language recognition** task, similar to those studied recently using other neural network models (Monner and Reggia, 2009). In this task, a network is given an English sentence as input and is expected to produce a set of predicates describing that sentence as output. The input sentence is represented as a temporal sequence of phonemes, each of which is a vector of binary acoustic features, borrowed directly from Weems and Reggia (2006). The network should produce as output a temporal sequence of predicates that bind key concepts in the sentence into coherent entities. For example, for the input sentence *the red pyramid is on the blue block*, the network should produce the predicates `(red X)`, `(pyramid X)`, `(blue Y)`, `(block Y)`, and `(on X Y)`. The variables are simply identifiers used to associate the various predicates with each other; in this example, the three predicates containing variable `X` come together to signify that a single object in the world is a red pyramid that is on top of something else. However, these predicate representations are not grounded, since they do not correspond to objects in a world that the network can observe; the issue of grounding is addressed in Chapter 4.

In the actual output representation used by the network, each predicate type is represented as a single one in a vector of zeros, and the variables required by each predicate are also represented in this way. In other words, a network performing this task requires a set of output neurons to represent the types of predicates, with each unit standing for a single predicate type, and two additional independent sets of neurons that each represent a variable, since there can be at most two variables involved in any predicate. The task demands that the network produce the required predicates in a temporal fashion to avoid imposing architectural limits on the number of predicates that a given input sentence could entail.

Part of the appeal of this task is that it is hierarchically decomposable. To discover a readily generalizable solution, common wisdom suggests that the best strategy for the network to use is to aggregate the incoming phonemes incrementally into morphemes, words, phrases, and finally, entire sentences. The intuition is that architectures capable of directly supporting this type of hierarchical decomposition would be superior to those that do not. Testing this notion involves a **two-stage architecture**, shown in Figure 2.6. At first it may appear complex, but it is essentially the standard LSTM architecture with peephole connections, except with a second hidden layer of memory block assemblies in series with the first. LSTM cannot efficiently train such an architecture because the back-propagated error signals would be truncated and never reach the earlier layer of memory blocks. LSTM-g, on the other hand, trains the two-stage architecture without difficulty. A standard LSTM network with peephole connections (see Figure 2.2) serves as a control to the two-stage architecture trained by LSTM-g, with its parameters appropriately adjusted to match the resources of the latter network as closely as possible.

Both networks in this experiment use essentially the same parameters as in the previous two experiments—the only difference is in the size of the networks. The two-stage architecture has 34 input units (corresponding to the size of the phoneme feature vectors used as input), 40 memory blocks in the first stage, 40 additional memory blocks in the second stage, and 14 output units, giving a total of 13,676 trainable connections. The standard LSTM control has the same 34 input units and 14 output units, with a single hidden layer of 87 memory blocks, giving it a slight edge with 7 more total memory blocks and 13,787 trainable connections. These numbers were selected to give the two networks parity in terms of computational resources, to the extent that the architecture designs and problem constraints allow.

During a single trial in the language recognition task, the network being tested is given each individual phoneme from a sentence as input in consecutive time-steps, and after the entire input sequence has been processed, the network must output one complete predicate on each subsequent time-step, until the network produces a special "done" predicate to signify that it is finished producing relevant predicates. The networks are trained to produce the predicates for a given sentence in a specified order but are scored in such

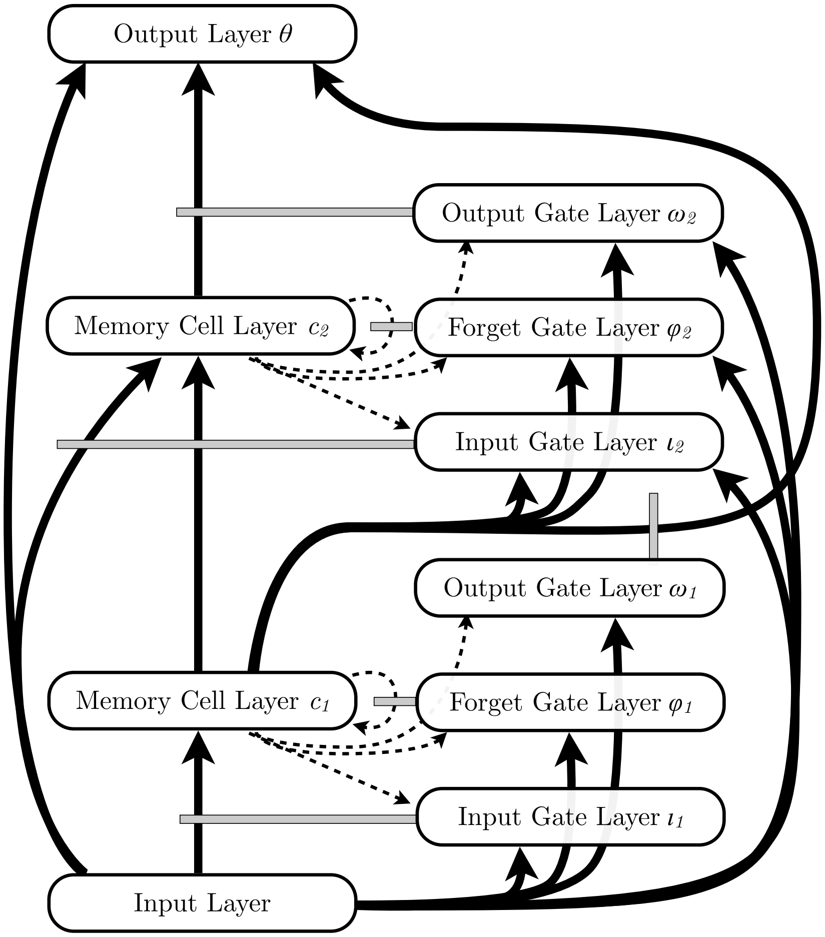

Output Layer θ
Output Gate Layer ω_2
Memory Cell Layer c_2
Forget Gate Layer φ_2
Input Gate Layer ι_2
Output Gate Layer ω_1
Memory Cell Layer c_1
Forget Gate Layer φ_1
Input Gate Layer ι_1
Input Layer

Figure 2.6: The two-stage network architecture, used in the third experiment. This architecture is a variant of the standard LSTM architecture with peephole connections (Figure 2.2) that has a second layer of memory block assemblies in series with the first. The traditional LSTM training algorithm cannot effectively train this architecture due to the truncation of error signals, which would never reach the earlier layer of memory blocks. Intuition suggests that the two self-recurrent layers allow this network to excel at hierarchically decomposable tasks such as the language recognition task.

a way that correct predicates produced in any order count as correct answers. A predicate is deemed to be correct if all units have activations on the correct side of 0.5. The testing program also records the number of unrelated predicates that the networks generate; however, this number closely tracks the inverse of the fraction of correct predicates produced, and as such, the reports that follow only include the number correct.

A small, mildly context-sensitive grammar generates the sentences and corresponding predicates for this simple version of the language recognition task. The sentences contain combinations of 10 different words, suggesting meanings involving 8 different types of predicates with up to 3 distinct objects referenced per sentence. The simplest sentences require only 3 output predicates to express their meanings, while the most complex require the networks to produce as many as 9 predicates in sequence as output. Thirty-two copies of each network train independently on this task. On each run, the network in question begins with random weights and is allowed to train through 1 million trials. The performance of the network is gauged periodically on a battery of 100 test sentences that the network has never previously encountered during training. The duration of training is more than sufficient to ensure that all networks reach their peak performance levels.

Figure 2.7 shows a comparison of the peak performance rates of the two types of networks, based on the fraction of correct predicates produced on the novel test sentences. The two-stage network trained with LSTM-g was able

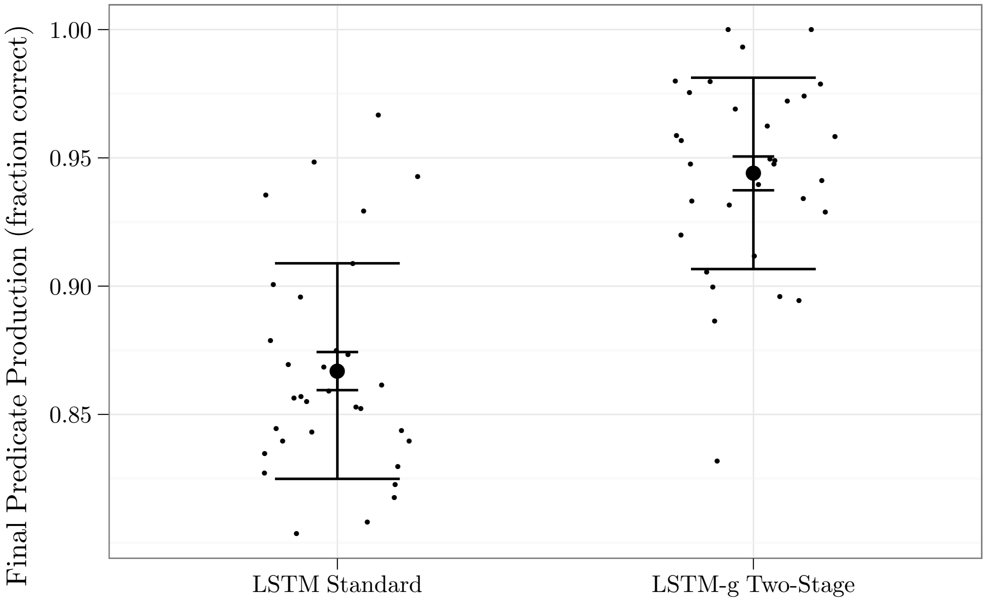


Figure 2.7: Plot of the performance results on the language recognition task produced by a standard LSTM network and the two-stage architecture trained by LSTM-g. The standard LSTM networks were able to produce approximately 87% of predicates correctly at peak performance, while the two-stage LSTM-g networks garnered 94% on average.

to achieve significantly better generalization performance than the standard LSTM network on average ($t \approx 9.4$, $df \approx 57.9$, $p < 10^{-12}$). In addition, the two-stage network was able to achieve this performance much more quickly than the control. Figure 2.8 plots the number of trials required for each network to produce 80% of predicates correctly; this threshold was chosen because every run of every network was able to achieve this performance level. The two-stage network required approximately 4 times fewer trials to reach the 80% performance criterion, which is a significant difference ($t \approx 10.9$, $df \approx 31.4$, $p < 10^{-11}$). These results underscore the value of using LSTM-g to train customized architectures that traditional LSTM cannot.

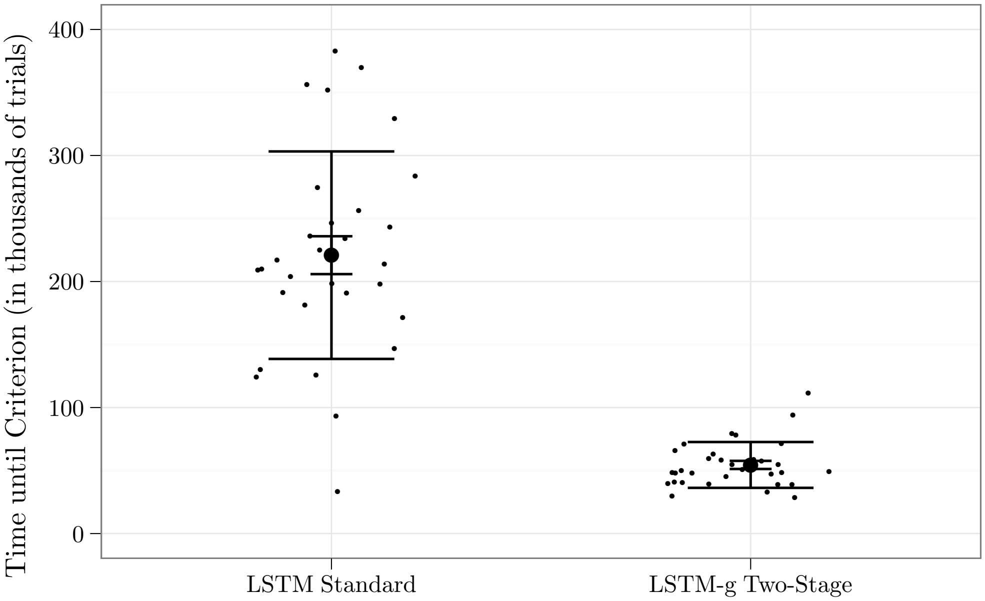


Figure 2.8: Plot of the training duration required for each type of network to reach the criterion of producing 80% of the required predicates for input sentences. The standard LSTM network required an average of about 220,000 trials to reach this performance criterion, while the two-stage network trained by LSTM-g required fewer than 60,000.

2.8 Discussion

The original LSTM architecture and the associated training algorithm was an important advance in gradient training methods for recurrent neural networks that must learn to handle temporal input and output series, even across long time lags. While the LSTM architecture and extensions thereof have proven useful in a variety of contexts, the LSTM training algorithm has often been replaced in these studies by the nonlocal BPTT, which was necessary to train more complex network architectures. Thus, the original LSTM training algorithm was limited in scope to a small family of second-order recurrent neural architectures. This chapter has introduced LSTM-g, a generalized algorithm that provides the power, speed, and spatial and temporal locality of the LSTM algorithm, while also being applicable to arbitrary second-order recurrent neural networks.

In addition to the increased architectural applicability it provides compared to the original training algorithm, LSTM-g makes use of extra back-propagated error when applied to the canonical LSTM network architecture with peephole connections. This error can be put to good use, with LSTM-g converging after less training than the LSTM training algorithm required in experiments that utilize the standard LSTM network architecture. Further, customized network architectures trained with LSTM-g can produce better performance than either LSTM or LSTM-g can produce when restricted to the standard architecture. In light of previous research that shows LSTM to outperform other recurrent network architectures when using training methods of comparable computational complexity (Hochreiter and Schmidhuber, 1997), the results contained herein suggest that LSTM-g may find productive application in many areas where customizable or dynamically changing network architectures are desirable.

LSTM-g is designed to provide a robust training algorithm for second-order neural networks where maintaining brain-like spatial and temporal locality is essential. In the future, however, it will likely be worth investigating whether LSTM-g may also be of use in situations where spatial and temporal locality are not hard requirements—situations in which BPTT, RTRL, DEKF, or other methods are generally used to train second-order network architectures. While it is unlikely that LSTM-g would outperform these algorithms in terms of final error rates, it is plausible that the locality properties of LSTM-g would lead to a better performance-to-computation ratio, resulting in faster convergence in terms of required computation time, and possibly to the formation of qualitatively different types of internal representations. Regardless of the outcome of such comparisons, LSTM-g broadens the applicability of local training algorithms to a wide variety of second-order recurrent neural networks.

Chapter 3

Modeling Critical Period Effects[1]

3.1 Introduction

Given the creation and initial testing of the generalized long short-term memory (LSTM-g) in Chapter 2, the extent to which the type of learning produced by the algorithm resembles human learning remains to be investigated. This question's importance is immediately clear for applications modeling human behaviors, like the neurocomputational language models that are the main focus of this dissertation. However, the question also has relevance for machine learning in general. Many machine learning algorithms, especially when they fail, often produce results that are inscrutable to humans, who have trouble following the chain of reasoning the machine uses to derive its answer. The extent to which a particular algorithm learns in a human-like way impacts humans' ability to relate to the results that the algorithm produces. This is one reason to prefer neural network algorithms—which depend on a distributed structure that resembles the human brain—to strict rule-based algorithms: The results produced tend to be softer, more graded, or more human-like than those produced by rules. But even among

[1] The research described in this chapter is joint work with Karen Vatz, Giovanna Morini, So-One Hwang, and Robert DeKeyser (Monner et al., under review).

neural network algorithms, one must consider the extent of this analogy. Do the networks suffer from the same learning biases, deficits, and artifacts that humans experience? This chapter explores this question with two neural network models that learn gender assignment and gender agreement in multiple languages with the overarching goal of reproducing a well-known human learning artifact called the critical period.

The term **critical period** is often used by language researchers to describe the observation that people who learn a second language (L2) later in life tend to have poorer ultimate attainment than those who learn the same language earlier in life. Cross-linguistically, there is a clear downward trend in many, although not all, measures of language proficiency as age of acquisition increases (DeKeyser, in press). These declines are not necessarily limited to an early period in the lifespan which is critical for language acquisition, as was originally thought. This phenomenon has been referred to by many names, usually based on the author's thoughts on the phenomenon's likely cause. Those who suspect some human maturational process to be responsible often refer to a critical period in language learning or age effects on language acquisition. Others, who view the issue as a problem inherent in the process of learning, speak of cross-linguistic interference or entrenchment effects. Still others couch the problem in terms of individual differences of the language learners and quality and form of the L2 input. While there is support for all of these accounts of this phenomenon, it is generally difficult to study any of the potential causes in isolation. Without committing to any particular view of the relative contributions of the causes or temporal durations of the effects, this chapter will refer collectively to age-correlated phenomena that impede second language acqusition as "critical period effects."

The research described in this chapter uses neural network models to investigate the individual and compound effects that two of these potential causes of critical period effects have on ultimate attainment of a learner's first and second languages. The first factor of study, **entrenchment**, can best be understood as previous knowledge of the first language that is difficult to change and can perhaps only be altered slowly, thus interfering with the rapid acquisition of newly available information about the second language. In this scenario, the longer that learners are exposed to their first language (L1) before a second language is introduced, the more L1 becomes entrenched, making L2 more difficult to learn.

The second factor examined is **memory development**—specifically, the aspects of working memory capacity and long-term memory capacity, as implemented by the periodic addition of new units and connections, respectively, to the neural network models. Working memory development is particularly interesting in light of evidence that the period of rapid growth of working memory capacity in children (Gathercole, 1999) coincides with the period of rapid deterioration of L2 learning ability, a theory that is usually known as the **"less is more" hypothesis** (Newport, 1988, 1990; Goldowsky and Newport, 1993).

Using only experimentation on human subjects, it is difficult to get a complete picture of the relative contributions of entrenchment and memory development. While there are exceptions, specifically in the sign language domain, language learning almost invariably starts very early in life, causing L1 acquisition and early L2 acquisition to coincide with many aspects of development. Thus, the contributions of these two factors to the observed differences in ultimate attainment between early and late L2 learners cannot be readily separated from each other. A computational model, on the other hand, enables study of the interaction of the two chosen factors from all sides, describing the effects of each in isolation as well as their combined impact.

Of course, at present, a computer model cannot learn an entire natural language as human learners can. As such, the models presented here learn the linguistic subtasks of **gender assignment** (learning to attribute gender to words) and **gender agreement** (learning to recognize phrases where related words all have the same gender). These gender-related tasks were chosen for two reasons: Native and non-native speakers of a language tend to differ significantly on these tasks, and ultimate attainment tends to vary with age of acquisition. The models described here learn to perform gender assignment and gender agreement tasks from naturalistic training data based on word co-occurrence, without having any built-in knowledge of the existence or form of grammatical gender and without being given explicit instruction in the genders of particular words or phrases. The goal is that these models will provide a better understanding of how the two potential factors studied here—entrenchment and memory development—contribute individually and in tandem to differences in ultimate language attainment. Comparing the patterns of attainment of the models with those of human learners can also begin to answer the important question of the extent to which results produced by LSTM-g will be relatable and intelligible to humans.

The remainder of this chapter is structured as follows. Section 3.2 reviews previous research on critical period effects and its relation to the acquisition of grammatical gender, as well as hypotheses relating this phenomenon to working memory and to L1 entrenchment. Section 3.3 describes the base neural network used in the models, as well as the variations used in the experimental conditions. Section 3.4 illustrates separate experiments and results for the gender assignment and gender agreement tasks. Finally, Section 3.5 discusses the implications of the simulation results.

3.2 Background

3.2.1 Critical period effects

As mentioned in Section 3.1, during the past few decades, researchers have accumulated a preponderance of evidence suggesting that learning a second language becomes increasingly difficult with age, resulting in poorer ultimate attainment—compared to native speakers—for individuals who begin learning a language later in life (see Hyltenstam and Abrahamsson, 2003, for an overview). This effect was first termed the critical period, indicating the period of time up until about age five during which children learning a second language do so with minimal impairment and no noticeable accent. Subsequent research has shown that a learner's ability to attain new languages continue to decline throughout the lifespan.

The potential causes of this phenomenon are many, and the data are difficult to disentangle due to unavoidable confounds associated with the development and maturation of the learners being studied (DeKeyser and Larson-Hall, 2005). Oft-cited maturational explanations of critical period effects include those at the level of neurobiology: hemispheric specialization (Lenneberg, 1967), myelination (Long, 1990), and neuro- and synaptogenesis (Uylings, 2006); and those at the level of psychology: working memory development (Newport, 1990), susceptibility to interference (Iverson et al., 2003), and progression from procedural to declarative learning (DeKeyser, 2000; Paradis, 2009; Ullman, 2004). Explanations that do not rely on the development of the learner include entrenchment of the L1 (MacWhinney, 2006), learner motivation to sound native (Bley-Vroman, 1988), amount of practice in the L2 (Jia et al., 2002; Jia and Aaronson, 2003), amount of formal education in the L2 (Hakuta et al., 2003), and the inherent variability of individuals (Abrahamsson and Hyltenstam, 2008; DeKeyser et al., 2010).

Since the various developmental explanations tend to correlate with each other and also with many of the nondevelopmental variables, experimental studies on human subjects that facilitate clean separation of all these theories are generally only possible in rare and specialized circumstances such as late acquisition of a native sign language (Mayberry et al., 2002). This confluence of confounds makes the issue ripe for computational simulations in which each factor can be controlled independently.

3.2.2 Grammatical gender

The size of the observed critical period effect varies across different types of linguistic competence. For example, the syntactic proficiency of a late learner seems to suffer less than morphological proficiency (Johnson and Newport, 1989). Low-level details of phonology, such as specific voice onset time (Abrahamsson and Hyltenstam, 2009), seem to be particularly difficult for late learners to master. Successfully modeling critical period effects, then, requires choosing a linguistic competency that is not only simple enough to model but also degrades as L2 age-of-onset increases.

The linguistic phenomenon that the models will learn about is grammatical gender, which refers to an arbitrary partitioning of words into classes referred to as genders. Several studies suggest that grammatical gender is subject to critical period effects. Late learners of French (Guillelmon and Grosjean, 2001), Spanish (Lew-Williams and Fernald, 2010), and German (Scherag et al., 2004) have all been shown to be slower than native speakers when processing gender agreement relations. Even children who start learning L2 French at an early age show accuracy deficits in gender assignment

English:	*The little book is white.*
	The little table is white.
French:	*Le petit livre* (masc.) *est blanc.*
	La petite table (fem.) *est blanche.*
Spanish:	*El libro* (masc.) *pequeño es blanco.*
	La mesa (fem.) *pequeña es blanca.*

Figure 3.1: Examples of gender agreement.

and agreement tasks when compared with native speakers (Holmes and de la Bâtie, 1999; Lapkin and Swain, 1977; Harley, 1979). These results and many others show that grammatical gender is subject to robust critical period effects and is thus an excellent candidate for computational modeling.

In particular, the models presented here will be learning to perform gender assignment and gender agreement tasks in two languages, French and Spanish. These languages have similar gender systems in that both have two gender classes for nouns, labeled "masculine" and "feminine," and determiners, adjectives, and pronouns related to these nouns take different forms to agree with the gender of the noun; see Figure 3.1 for an example. In both languages, the gender of a noun is highly predictable from its phonological form, though this predictability is more reliable in Spanish, due in part to the simplicity of the rules (Surridge, 1993, 1995; Teschner and Russell, 1984). Morphological and semantic cues are also predictive above and beyond the phonological form (Surridge, 1989).

Grammatical gender in other languages is sometimes nonexistent, as in English, and sometimes more complex, involving more gender classes, more marked works, or fewer reliable cues to a word's gender (Corbett, 1991).

French and Spanish, however, are an ideal combination for a first attempt at modeling critical period effects, due both to their simplicity and the immediate availability of human expertise in both languages.

3.2.3 Memory development

Working memory is the name given to humans' ability to store information for a short time to use in near-term processing. Working memory capacity is a fuzzy psychological measure of the number of separate items that can be held in memory at once and successfully recalled. A higher working memory capacity is generally associated with improved performance on a wide variety of tasks (Baddeley, 2003) and is often described as a key component of general intelligence (Duncan et al., 2000).

Psychological measures of working memory capacity show that it grows rapidly throughout childhood until plateauing in late adolescence (Gathercole, 1999). As mentioned in Section 3.1, this rapid growth accompanies a similarly rapid decrease in the final proficiency of languages introduced at these ages. It is initially puzzling that these two measures would have inverse trajectories. The "less is more" hypothesis (Newport, 1990) provides a solution to this puzzle by insisting that smaller working memory capacity is essential to fully internalizing the morphological, generative structure of language. The idea is that a smaller working memory capacity provides a bottleneck to force the learner to analyze linguistic input in smaller chunks, making low-level morphological features more salient and thus more easily internalized. Learners who are first exposed to a language when their working memory capacities are larger, in contrast, process larger bits of language as "unanalyzed wholes," thus missing some of the compositional structure.

The neural network models presented in this chapter undergo memory development, in the form of changes in both working memory capacity and long-term memory capacity, to examine the effects of maturation on critical period effects. In a recurrent neural network, the equivalent of working memory is the retention of activation across time-steps due to recurrent connectivity. The amount of activity that can be maintained is proportional to the total number of units that can hold activation. Thus, changes in working memory capacity can be modeled in a neural network as changes in the number of units in the hidden layers. Of course, when varying the number of units, one is generally forced to also vary the number of connections between units. Since these connections are modified slowly, they correspond to the network's long-term memory. The combination of experimental conditions detailed in Section 3.3.2 is designed to tease apart the contributions of changes in working memory capacity and long-term memory capacity.

3.2.4 Past computational modeling approaches

As discussed in Section 3.1, it is often difficult to experimentally separate the various possible causes of age effects when performing empirical research on human subjects. Computational modeling has a key advantage in its ability to independently manipulate a number of variables and to observe their main effects and interactions. Early attempts at computational modeling of critical period effects (Goldowsky and Newport, 1993) show support for the "less is more" hypothesis in that a model with a smaller working memory was better able to learn certain grammatical patterns; this conclusion was supported by later studies, computational and otherwise (Cochran et al., 1999; Kareev et al., 1997; Kersten and Earles, 2001).

Previous neural network models that have dealt with aspects of memory development have used varying approaches to limiting working memory. Elman (1993) trained simple recurrent networks (SRNs) on a complex subset of English. This type of network uses recurrent connections to allow the network to access its own previous states, creating an analog of working memory. Elman found that these networks had better eventual performance when this working memory was initially limited to a discrete window of a few steps and gradually increased, consistent with the "less is more" hypothesis. While others have failed to find a difference between developing and mature networks on similar tasks (e.g., Rohde and Plaut, 1999), Elman's study shows one way in which working memory capacity can be modeled in a neural network. The models presented here use a different approach, directly limiting the number of units whose maintained activations provide access to previous states, as opposed to limiting the network's temporal window of access to these states. This approach is, in a sense, similar to that of the DevLex models of word and meaning acquisition (Li et al., 2004, 2007), which utilize growing self-organizing maps to represent semantics and phonology. These maps grow by adding new units to accommodate storage of new lexical and semantic representations; as such, the growth involved more closely resembles long-term memory growth. The models in this chapter, in contrast, grow by adding new units that form the substrate for working memory.

There have also been a few notable neural network models that involve grammatical gender. MacWhinney et al. (1989) presented two neural network models of the acquisition of gender, case, and number in German. Both of these models learned to predict the article associated with a given noun, one using hand-coded semantic, phonological, morphological, and case cues, and the other using only observable data in the form of a complete phonological representation of the input noun along with some semantic and case cues. Both models succeeded at learning the nouns they were trained on and also generalized very well to new nouns. The second model, without the hand-coded cues, outperformed the first. Unfortunately, the static phonological representations in this model only allow it to be applied to words of two syllables or fewer. The models developed here, on the other hand, employ temporal phonological representations that allow any word of virtually any spoken language to be encoded.

Another neural network model of grammatical gender was advanced by Sokolik and Smith (1992), who trained a feed-forward neural network to identify a corpus of French nouns as either masculine or feminine. Their study, however, has been widely criticized (Carroll, 1995; Matthews, 1999) for, among other things, using orthographic input, giving explicit gender feedback, and building in language-specific knowledge about gender classes. The approach demonstrated in this chapter addresses these and other concerns, resulting in models that utilize only the information actually available to language learners.

3.3 Methods

3.3.1 Neural network methods

The models described in this chapter approximate a language learner using a long short-term memory neural network (LSTM; Hochreiter and Schmidhuber, 1997; Gers and Cummins, 2000; Gers and Schmidhuber, 2001) trained with the LSTM-g algorithm described in Chapter 2. The specifics of the models, including the number of hidden layers and the number and type of outputs, vary between the models designed for the gender assignment and gender agreement tasks, and as such are described in detail for each case in Section 3.4. However, the basic input and processing of the models is the same. Since the aim of these models is to learn gender properties from speech stimuli, they are each given an input layer able to represent one phoneme of speech at a time. The network hears a sequence of such phonemes, one after another, with the sequence as a whole representing a word or noun phrase. This process is analogous to listening to spoken sentence fragments.

The experiments that follow use these neural network models to understand any critical period effects that arise due to the effects of entrenchment and aspects of memory development. The first of these two factors is straightforward to implement: Simply teach a network to perform the same task in two languages. Varying the amount of time before the L2 is introduced also varies the expected amount of entrenchment of the L1. The second factor is developmental, and involves changes to a neural network's structure and connectivity over the course of the experiment, above and beyond the connection-weight changes that occur during normal training. The different development conditions and their effects on the neural architecture of the models are discussed in the next section.

3.3.2 Development and network architecture

Most neural network models have a fixed number of units and connections for the duration of training. Training such a network, starting from randomly assigned connection weights, is tantamount to waiting until a human learner is an adult, or at least fully neurologically developed in the relevant areas, before exposing him or her to any language stimuli. Addressing cases where language learning happens along with development requires the examination of situations where the network structure develops during training. The following paragraphs explain a few ways of doing this.

The default case just described, where all of the network's units and connections are present at the start of training, is termed the **no growth** condition. In a second **unit growth** condition, the network begins with a much smaller number of units and connections (see Figure 3.2). During the training regimen, new units and their associated connections are gradually added to the network until it reaches maturity with its maximum number of units and connections, equivalent to the numbers present in the no growth condition. Here, a new unit being added to the network is not necessarily analogous to neurogenesis in humans; an alternative is that some of the new connections, created through a process analogous to dendritic outgrowth (Uylings, 2006), happen to project to existing units outside the current connected component of the network, thus recruiting them for use in the current task.

In this type of recurrent neural network, the persistent activations of units are the basis of working memory. The network recruits new units during the maturation process, increasing the amount of information it can process at any given instant. One would reasonably expect this to correlate with an increase in cognitive measures of working memory capacity during training. Since these networks start with a small working memory and increase this capacity during training, this condition will help test Newport's "less is

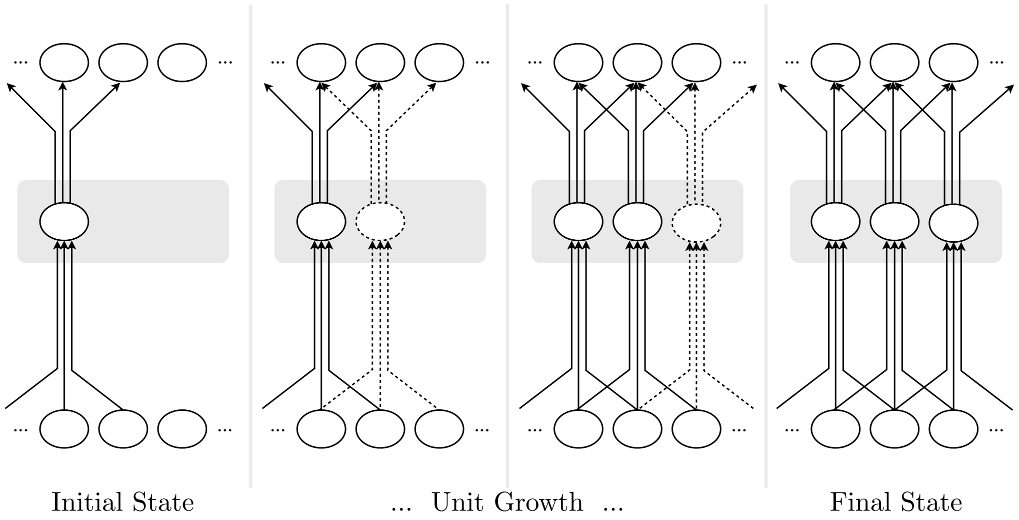


Figure 3.2: A depiction of a network from the unit growth condition at various states of maturity. On the left is the network's initial state, with few units in the hidden layer (delineated by the gray box). As time passes, the network begins to recruit new units and their associated connections for this layer (dashed lines), finally ending up in its mature state on the right, where the number of units and connections is equivalent to a network from the no growth condition.

more" hypothesis. This hypothesis admits two distinct and independently controllable factors that could lead to better final language performance: (1) starting with a small working memory, and (2) allocation of new working memory resources during learning. The unit growth condition possesses both factors, so separate investigation of these factors requires a third network development condition, termed **unit replacement**, that has only the second factor. In this condition, depicted in Figure 3.3, the network starts in the same state as the no growth condition, with its full complement of units and connections, and thus its full working memory capacity. Periodically, units and their associated connections are removed from the network and replaced with new units and fresh, untrained connections. This happens at a rate

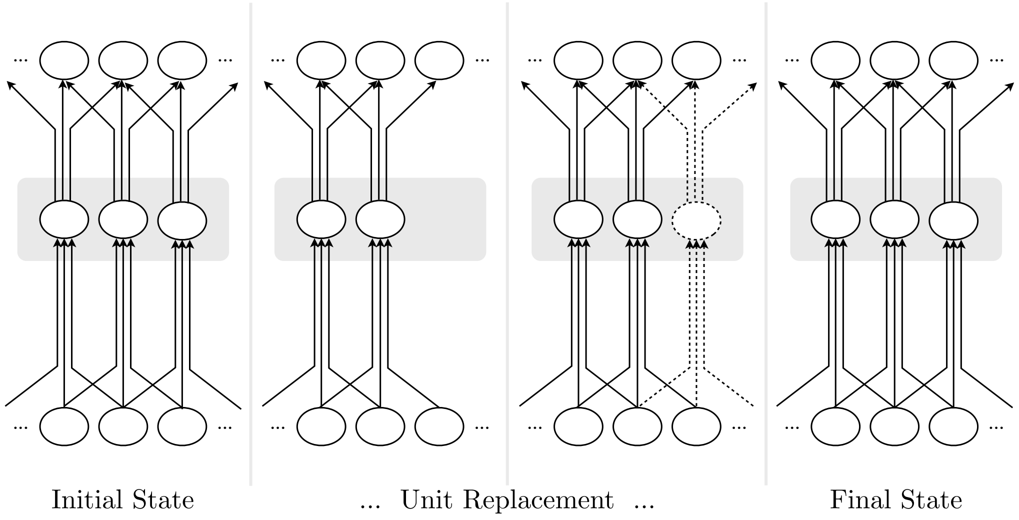


Figure 3.3: A network from the unit replacement condition starts with a full complement of units that are periodically removed and replaced with fresh units and untrained connections.

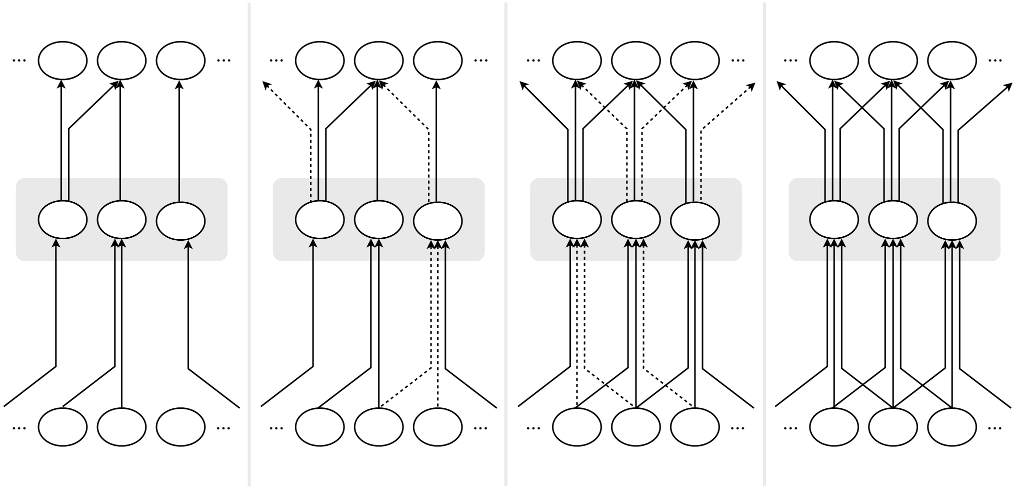
commensurate with the rate at which units are added in the unit growth condition. Thus, in both conditions fresh resources are introduced over time at the same rate, but where the unit growth condition uses these resources to grow the network from its initially small size, the unit replacement condition accepts these fresh resources and discards an equal amount of its existing, trained resources, thereby maintaining a constant size. Since the effective size of the working memory does not change in the unit replacement condition, it will help determine if periodic introduction of fresh working memory resources alone, without starting small, can produce any significant benefits.

The unit growth condition described earlier confounds two variables of interest on the cognitive level. The new units that each network recruits must be wired up using new connections. These connections, of course, are the basis of long-term memory capacity in a neural network. Thus, a network from the unit growth condition adds both working memory and long-term memory capacity during training. Teasing apart these variables necessitates a fourth condition, termed the **connection growth** condition, in which all units are present from the beginning but few of the possible connections exist (see Figure 3.4). Since all units are incorporated via active connections from the beginning, the network's working memory capacity is fully developed from the start. During training, the network grows new connections at the same rate as in the unit growth condition, giving the network access to new long-term memory storage and allowing direct measurement of the effects of long-term memory maturation. In addition, this facilitates indirect assessment of the contributions of working memory maturation (and compound effects) by subtractive analysis with the unit growth and no growth conditions.

3.4 Experiments and results

3.4.1 Gender assignment task

The first set of experiments investigates how well a neural network model can learn to perform a gender assignment task using realistic sources of information. This model, termed the **article-prediction model**, takes single nouns as input and uses that information to predict which determiners can appear with the noun. Since nouns rarely occur without determiners in the target languages of French and Spanish, both the input and the output



Initial State ... Connection Growth ... Final State

Figure 3.4: Diagram of a network from the connection growth condition. This network begins with a full complement of units in the hidden layer (gray box), but they are sparsely connected to the other units. As time passes, these units develop new connections with other units, finally ending up in a mature state where the number of connections equals that found in networks from the no growth and unit growth conditions.

data are readily available to any learner by simply listening to everyday speech. After training, the article-prediction model is tested by presenting it with nouns and asking it to predict the associated determiners. The gender of the most strongly predicted determiner is taken to be the model's gender assignment for the input noun.

This approach resembles that taken by the third model from MacWhinney et al. (1989) in that both models use the complete phonological form of a noun to predict the article to be used with that noun. The article-prediction model diverges from this earlier model in a few important ways. First, it eschews semantic features to investigate what can be learned from phonology alone. Second, the article-prediction model represents each input noun as a temporal sequence of phonemes instead of a single phonological pattern, the latter of which will always have trouble representing long words or those that do not conform to the prespecified representational form. In addition, this approach corrects the most severe issues with the model of gender assignment by Sokolik and Smith (1992). Where their approach was criticized (Carroll, 1995; Matthews, 1999) for using orthographic input, the article-prediction model uses phonemic input instead. Where their network came *a priori* equipped with knowledge of the genders of the training language—and indeed the knowledge that grammatical gender exists at all—the article-prediction model has no such built-in knowledge. Finally, where their model required explicit feedback about the genders of individual words, the article-prediction model relies instead upon the co-occurrence of gendered articles with nouns to deduce gender assignments. As a result of these differences, the article-prediction model is more closely aligned with the real-world circumstances of human language learning in most contexts.

An input noun is presented to the network as a temporal sequence of phonemes. Each such phoneme is represented as a set of binary auditory features, with the activations of the network's input layer adjusted to reflect the feature set of each phoneme in turn. The model uses this representation because such features are universal in the sense that various configurations of these features can represent virtually any phoneme. As such, units representing these features could potentially be a built-in component of the brain of a language learner or could be learned. That said, the model only includes enough features here to distinguish all phonemes in the target languages; this set of phonemes and features is detailed in Table 3.1. After processing an entire sequence of phonemes that represents the input noun, the network activates output-layer units corresponding to its predictions of determiners compatible with that input noun. The network learns to perform this behavior by observing determiner-noun pairings and adjusting its connection weights accordingly.

Figure 3.5 shows the general architecture of the networks trained to perform this gender assignment task. The networks have an input layer of units corresponding to the features that make up the input phonemes. Units in the input layer project to units in a single hidden layer of memory cells. The intrinsic self-recurrence of the memory cells forms the substrate for working memory in the network. Finally, the hidden layer projects forward to the output layer, which consists of 9 units representing the definite and indefinite singular determiners of the target languages: *le*, *la*, *l'*, *un*, and *une* in French and *el*, *la*, *un*, and *una* in Spanish. This should not be interpreted as an assertion that units representing these words could be built into the brains of language learners, nor that the words are represented in single units. However, since these determiners form a small closed class of words, it is not too large a leap to presume that the learner represents these frequent determiners as distinct entities before much gender learning takes place. The single-unit representation for each determiner is the simplest possible in this context, though other representations would likely work as well. This model specifi-

Table 3.1: Binary Auditory Feature Representations of Phonemes

	consonantal	sonorant	continuant	strident	nasal	lateral	trill	voice	labial	round	coronal	anterior	distributed	dorsal	high	low	back	radial	ATR
h	−	+	+	−	−	−	−	−	−	−	−	−	−	+	+	−	+	−	−
ɉ	−	+	+	−	−	−	−	+	−	−	−	−	−	+	+	−	−	−	−
ɛ	−	+	+	−	−	−	−	+	−	−	−	−	−	+	−	−	−	+	−
e	−	+	+	−	−	−	−	+	−	−	−	−	−	+	−	−	−	+	+
ə	−	+	+	−	−	−	−	+	−	−	−	−	−	+	−	−	+	−	−
a	−	+	+	−	−	−	−	+	−	−	−	−	−	+	−	+	+	+	−
i	−	+	+	−	−	−	−	+	−	−	−	−	−	+	+	−	−	+	+
ɔ	−	+	+	−	−	−	−	+	+	+	−	−	−	+	−	−	+	+	−
o	−	+	+	−	−	−	−	+	+	+	−	−	−	+	−	−	+	+	+
y	−	+	+	−	−	−	−	+	+	+	−	−	−	+	+	−	−	+	+
u	−	+	+	−	−	−	−	+	+	+	−	−	−	+	+	−	+	+	+
æ	−	+	+	−	−	−	−	+	+	+	−	−	−	+	−	−	−	+	−
ϕ	−	+	+	−	−	−	−	+	+	+	−	−	−	+	−	−	−	+	+
ɛ̃	−	+	+	−	+	−	−	+	−	−	−	−	−	+	−	−	−	+	−
ã	−	+	+	−	+	−	−	+	−	−	−	−	−	+	−	+	+	+	−
ɔ̃	−	+	+	−	+	−	−	+	+	+	−	−	−	+	−	−	+	+	−
æ̃	−	+	+	−	+	−	−	+	+	+	−	−	−	+	−	−	−	+	−
k	+	−	−	−	−	−	−	−	−	−	−	−	−	+	+	−	+	−	−
t	+	−	−	−	−	−	−	−	−	−	+	+	−	−	−	−	−	−	−
p	+	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−
g	+	−	−	−	−	−	−	+	−	−	−	−	−	+	+	−	+	−	−
d	+	−	−	−	−	−	−	+	−	−	+	+	−	−	−	−	−	−	−
b	+	−	−	−	−	−	−	+	+	−	−	−	−	−	−	−	−	−	−
ɣ	+	−	+	−	−	−	−	+	−	−	−	−	−	+	+	−	+	−	−
ð	+	−	+	−	−	−	−	+	−	−	+	+	−	−	−	−	−	−	−
θ	+	−	+	−	−	−	−	−	−	−	+	+	−	−	−	−	−	−	−
β	+	−	+	−	−	−	−	+	+	−	−	−	−	−	−	−	−	−	−
ʃ	+	−	+	+	−	−	−	−	−	−	+	−	+	−	−	−	−	−	−
s	+	−	+	+	−	−	−	−	−	−	+	+	−	−	−	−	−	−	−
f	+	−	+	+	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−
ʀ	+	−	+	+	−	−	−	+	−	−	−	−	−	+	−	−	+	−	−
ʒ	+	−	+	+	−	−	−	+	−	−	+	−	+	−	−	−	−	−	−
z	+	−	+	+	−	−	−	+	−	−	+	+	−	−	−	−	−	−	−
v	+	−	+	+	−	−	−	+	+	−	−	−	−	−	−	−	−	−	−
ʎ	+	+	−	−	−	+	−	+	−	−	−	−	−	+	+	−	−	−	−
l	+	+	−	−	−	+	−	+	−	−	+	+	−	−	−	−	−	−	−
ɲ	+	+	−	−	+	−	−	+	−	−	−	−	−	+	+	−	−	−	−
ɳ	+	+	−	−	+	−	−	+	−	−	−	−	−	+	+	−	+	−	−
n	+	+	−	−	+	−	−	+	−	−	+	+	−	−	−	−	−	−	−
m	+	+	−	−	+	−	−	+	+	−	−	−	−	−	−	−	−	−	−
j	+	+	+	−	−	−	−	+	−	−	−	−	−	+	+	−	−	−	−
ɹ	+	+	+	−	−	−	−	+	−	−	+	+	−	−	−	−	−	−	−
w	+	+	+	−	−	−	−	+	+	−	−	−	−	+	+	−	+	−	−
r	+	+	+	−	−	−	+	+	−	−	+	+	−	−	−	−	−	−	−

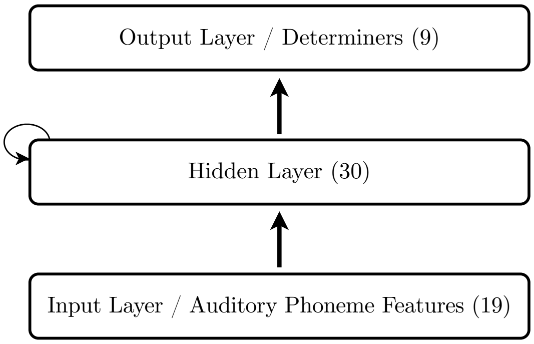

Output Layer / Determiners (9)
Hidden Layer (30)
Input Layer / Auditory Phoneme Features (19)

Figure 3.5: The network architecture of the article-prediction model used in the gender assignment task. The network takes features of auditory phonemes as input, passes them through a hidden layer of self-recurrent memory cells, and maps a sequence of such inputs onto an output layer of units representing singular definite and indefinite determiners in two languages. Since this is an LSTM network, the hidden layer is a group of memory blocks, each of which consists of a memory cell and several multiplicative gates; for simpler presentation than in Chapter 2, the memory cells and gate units are not depicted as separate layers. The self-connection shown on the hidden layer signifies that individual memory cells in this layer are self-recurrent, remembering their activations from the previous step. The network has approximately 5,000 trainable connection weights.

cally avoids using a feature-based determiner representation because features like gender and definiteness are not explicitly present in speech signals.

For this set of experiments, the input data for the model consists of the 600 French words from the Sokolik and Smith (1992) paper as well as 600 equivalent words from Spanish. Each trial during training starts by selecting a language and then selecting a noun at random from that language's corpus. The noun is paired with either a gender-matched definite or indefinite determiner from the appropriate language to form a simple noun phrase. The noun is given as input to the network, which then predicts applicable determiners and adjusts its weights in such a way that, in the future, it will be more likely to predict the determiner that actually co-occurred with the input noun. A network is considered to have assigned the correct gender for an input noun if an article of the appropriate gender is most active after presentation. While stronger criteria are possible, this choice reflects the common practice of testing human learners on forced-choice gender assignment tasks; furthermore, this choice ensures that any deficits observed in the models cannot be attributed to overly strong assignment criteria.

To determine a baseline level of performance on the gender assignment task, the first round of networks learned the task in either French or Spanish only. The networks' performance results are shown in Figure 3.6. As one would expect, networks trained on French alone scored well in excess of 90% after training, while scoring at chance on Spanish; similarly, Spanish-trained networks performed well on their native language and at chance on French. It is worth noting that Spanish performance was consistently a few percentage points better than French performance, likely due to the phonemic cues to gender assignment in Spanish being simpler and more reliable than those in French. Performance was consistent across the four development conditions, suggesting that network development alone has little impact on outcomes for the gender assignment task, at least in the first language.

With a baseline level of performance established for networks that are native to either French or Spanish, the investigation turned to the performance of bilingual networks under a number of different learning conditions

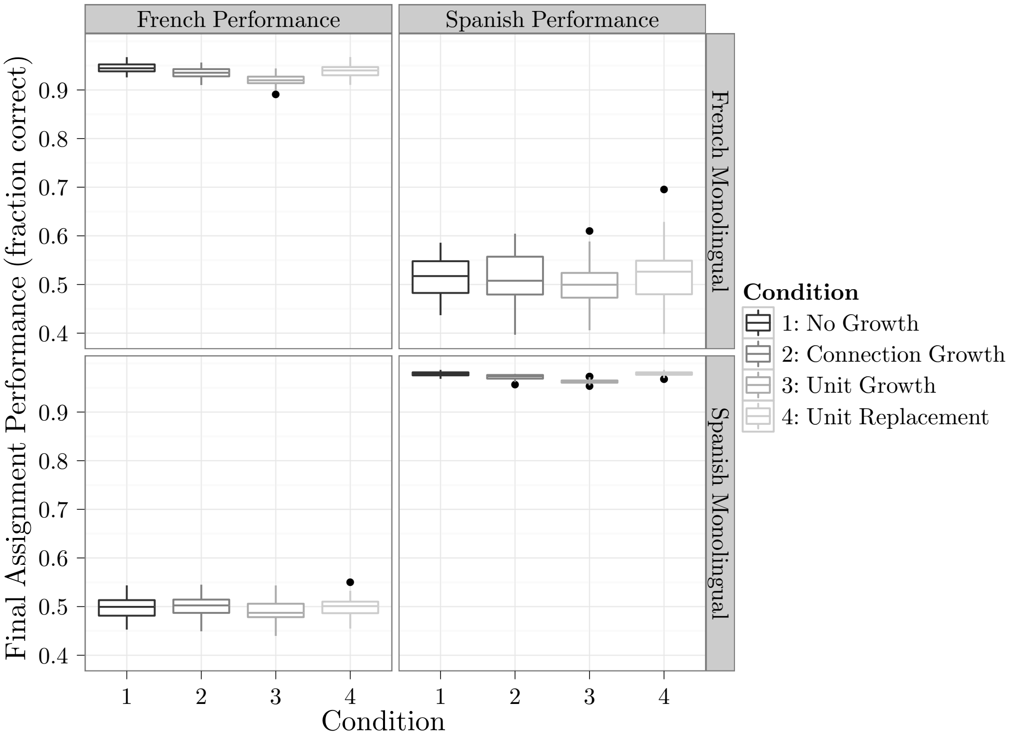


Figure 3.6: Results for monolingual networks, of all four developmental varieties, on the gender assignment task. Thirty separate networks learned the task in each developmental condition for each language. Each network was trained for 2 million trials in one language, and then evaluated on both languages.

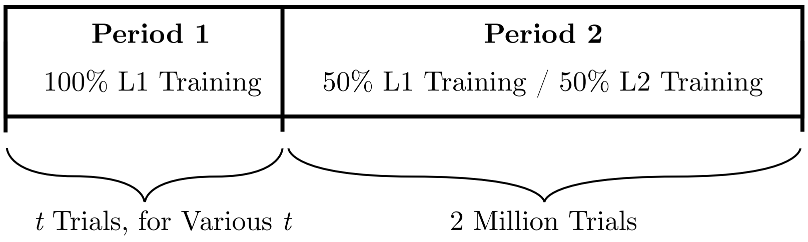


Figure 3.7: Diagram of the training schedule, consisting of two periods, for a generalized condition parametrized by a single variable t. In period 1, the network is trained on t trials in L1 exclusively. Training then proceeds to period 2 with a mixed training schedule of both L1 and L2 trials where the language of each trial is chosen with uniform probability. A true bilingual network, for which neither language is ever prioritized, can be simulated with $t = 0$; larger values of t correspond to later learners of their L2.

designed to assess the role of L1 entrenchment. Each condition varies the length of time t that the network spends learning the task on L1 alone before L2 is introduced. Each condition has two periods, with the first period varying in duration (see Figure 3.7). Period 1 consists of training only in L1 for t trials, where t varies widely across conditions. Immediately following this is period 2, in which L1 and L2 trials are mixed with equal probability. The duration of period 2 is always 2 million trials in an effort to ensure that the networks have time to reach peak performance on both languages.

A network whose training regimen has $t = 0$ is a native bilingual in the sense that L1 and L2 are presented at precisely the same time, and in the same proportions. Thus, such a network should exhibit no L1 entrenchment. Networks trained with higher values of t, having had a longer time with exposure only to L1, should exhibit more entrenchment. The prevailing ideas about L1 entrenchment offer a number of predictions about the final, peak L1 and L2 performance of the networks:

1. Networks should maintain relatively consistent final L1 performance regardless of the value of t;

2. Networks trained with $t = 0$, as native bilinguals, should not exhibit impairment in either language with respect to the other; and,

3. Networks should show increasing degradation of final L2 performance as t increases, at least until the networks have mastered L1 to a point at which the effect of entrenchment saturates.

These predictions can be investigated by plotting the final L1 and L2 performance of fully trained networks on the gender assignment task versus the value of t with which they were trained. Thirty separate networks train with each of 15 values of t (see Figure 3.8), as well as each of the four maturation conditions and each of two languages; thus, a total of 3,600 networks are trained to produce the following figures. For conditions in which the network matures during training, each of these networks begins training in its most immature state and develops over the course of the first 400,000 trials, at which point it reaches maturity—that is, architectural parity with the networks in the no growth condition. Thus, some networks in the connection growth and unit growth conditions (i.e., those with $t = 0$) are first exposed to L2 in their most immature state, while others (i.e., $t \geq 400{,}000$) are not exposed to L2 until after reaching maturity.

After both training periods are complete, one can record the fraction of inputs that each network assigns the correct gender, for both L1 and L2.

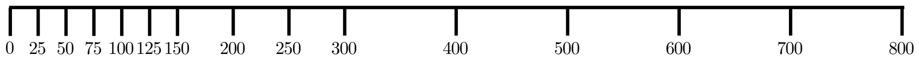


Figure 3.8: Values of t used in the assignment experiments, in thousands of trials. The values are spaced closer together at the beginning of the spectrum because that is where the large performance changes tend to happen.

The graphs shown in this section and the next depict the final performance of the networks on the y-axis versus the value of t—that is, the duration of the L1-only training period and thus the delay before L2 onset relative to L1—on the x-axis. Thus, the expected L1 entrenchment increases from negligible to maximal as one moves from left to right in each figure; another way of saying this is that the networks towards the left of the x-axis are closer to true bilinguals, whereas the networks closer to the right edge are late L2 learners. The y-axis values always depict final performance after the conclusion of training. These graphs show fitted curves for each of the different network maturation conditions, and for each such curve, the shaded area behind it represents the 95% confidence interval.

Investigating the first prediction requires examining the performance of the various networks in their native language (Figure 3.9). The prediction of relatively consistent performance across all values of t (i.e., levels of L1 entrenchment) appears to be largely borne out, as none of the individual curves differ by more that a couple percentage points as they move from left to right. As with the monolingual networks, French appears to be a bit harder to learn than Spanish, again likely due to the more regular nature of Spanish's phonetic gender cues. Differences in Spanish performance between

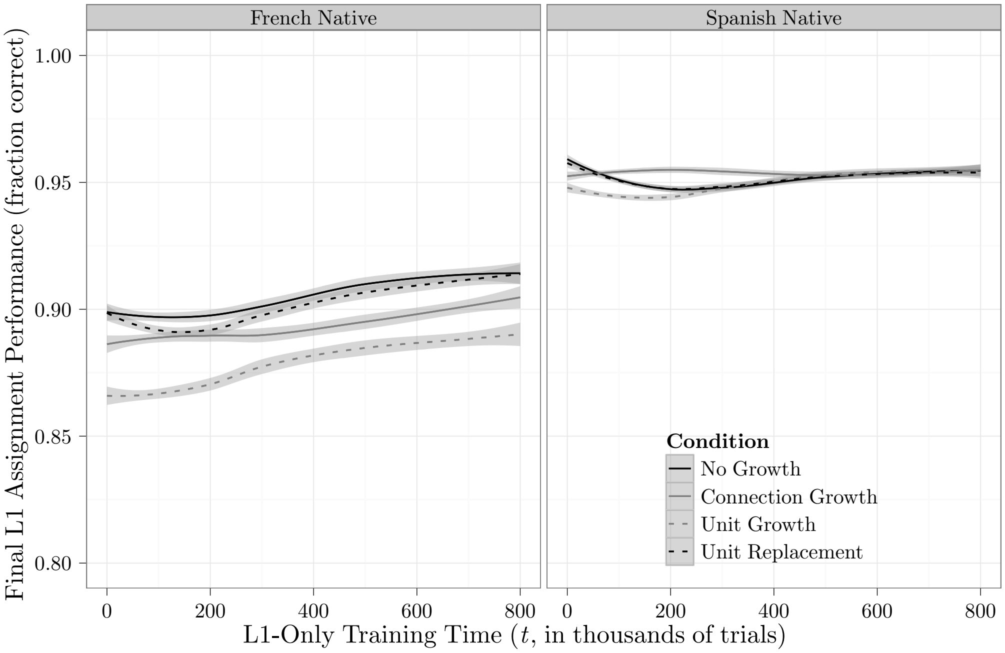


Figure 3.9: Final performance (fraction of test trials correctly answered) for bilingual networks tested on the gender assignment task in their native language after training was complete. The x-axis varies the time each network spent with its native language in isolation before the second language was introduced.

the different maturational variants of Spanish-native networks were minimal, while the French-native networks that grew their working memory capacity during training showed a slight disadvantage but the same general pattern of nearly flat performance across all values of t.

Figure 3.10 depicts each network's performance on its second language. A comparison of the performance of networks with $t = 0$ on the x-axis to the performance levels from the native languages in Figure 3.9 shows that, for most of the maturational conditions, the true bilingual networks (those with $t = 0$) perform as well as any tested network in both languages, lending support to the second prediction.

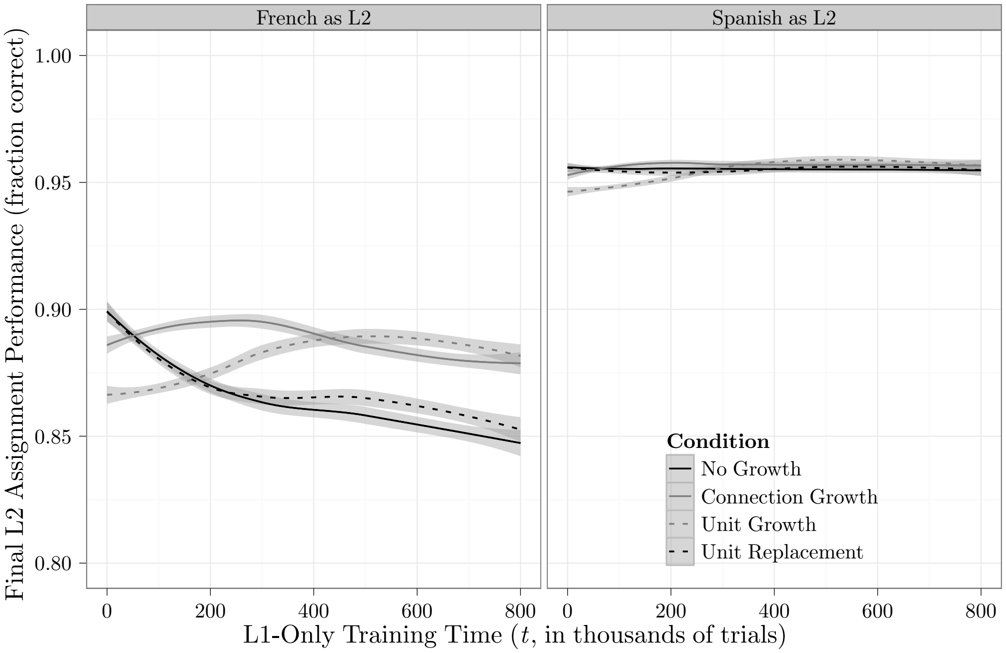


Figure 3.10: Final performance (fraction of test trials correctly answered) for bilingual networks tested on the gender assignment task in their second language after training was complete. The x-axis varies the time each network spent with its native language in isolation before the second language was introduced.

Moving on to the third prediction, Figure 3.10 reveals a t-related performance deficit in the no growth condition for L2 French; increasing Spanish exposure before French is introduced causes the final French performance of the network to decrease at a rate that is at first rapid but eventually slows for larger delays. The maturational properties in play for the connection growth and unit growth conditions, however, appear to have helped these networks compensate for the expected declines in French performance due to Spanish entrenchment. Networks in the unit replacement condition tended to perform at levels comparable to the no growth networks, suggesting that introduction of new working-memory resources without starting small may not be sufficient to gain a significant reprieve from the deleterious effects of increasing L1 entrenchment. On the right-hand graph, where French was the L1 and Spanish the L2, no appreciable t-related performance decreases were observed. This appears to be due again to the relative ease of the task for Spanish as compared to French.

At least in the case of French as an L2, the data shown in Figure 3.10 support both the predictions of performance declining due to increased entrenchment and of maturation during learning helping to overcome these difficulties.

3.4.2 Gender agreement task

The second set of experiments explores how a neural network model performs on a gender agreement task. During a trial, the **phoneme-prediction model** receives a noun phrase (e.g., *el mecanismo interno* in Spanish) presented as an unsegmented sequence of phonemes (e.g., [elmekanismointerno]) as input. The model's job at every point in this phoneme sequence is to predict the next few phonemes that it will hear. As such, the model could use everyday speech as both the input and the training signal. After training, the network's gender agreement performance is evaluated using noun phrases of the form determiner-noun-adjective—common constructions in the target languages of Spanish and French. To determine gender agreement, one gives the network the determiner and noun as input, followed by the portion of the adjective that is gender-neutral, and asks the network to predict the correct ending for the adjective. If the network predicts the gender-appropriate ending more strongly than the gender-inappropriate ending, its answer counts as correct.

The noun phrases serving as training data for the gender agreement task were extracted from the complete text of the French and Spanish versions of Wikipedia (2011). TreeTagger (Schmid, 1994) applied a part-of-speech tag to each word in either corpus. Noun phrases—of the forms determiner-noun, determiner-noun-adjective, and the less frequent determiner-adjective-noun, where the determiner is one from Figure 3.11—were extracted from the

French:

la, le, l', un, une, ce, cette, cet, aucun, aucune, chaque, tel, telle, sa, son, ma, mon, ta, ton, notre, votre, leur

Spanish:

el, la, un, una, este, esta, ese, esa, aquel, aquella, ningún, ninguno, ninguna, cualquier, cualquiera, cada, su, tu, mi, nuestra, nuestro, vuestra, vuestro

Figure 3.11: Determiners used in the gender agreement task.

resulting output. This list of noun phrases was filtered, removing any phrases containing words that were not in the appropriate language dictionaries—Lexique 3 for French (New, 2006) and CUMBRE for Spanish (2010). Finally, the lists were truncated to the most frequent 100,000 noun phrases for each language. These phrases comprise the training data. Each training trial chooses a phrase probabilistically, based on the phrases' corpus frequencies, to use as the input—and training signal—for the network on that trial.

Figure 3.12 depicts the architecture of the networks trained on the gender agreement task. The input layer is the same as it was for the gender assignment experiments, with each input unit corresponding to a binary auditory feature of a phoneme. This model, however, has two hidden layers of memory cells instead of one. This is because the gender agreement task involves two separate levels of segmentation of the input. To perform the task effectively, a learner would generally need to divide the phoneme sequence first into morphemes and words and, at a higher level, into noun phrases in which gender agreement must be maintained. Previous experiments with these types of networks on language tasks (Monner and Reggia, in press a)

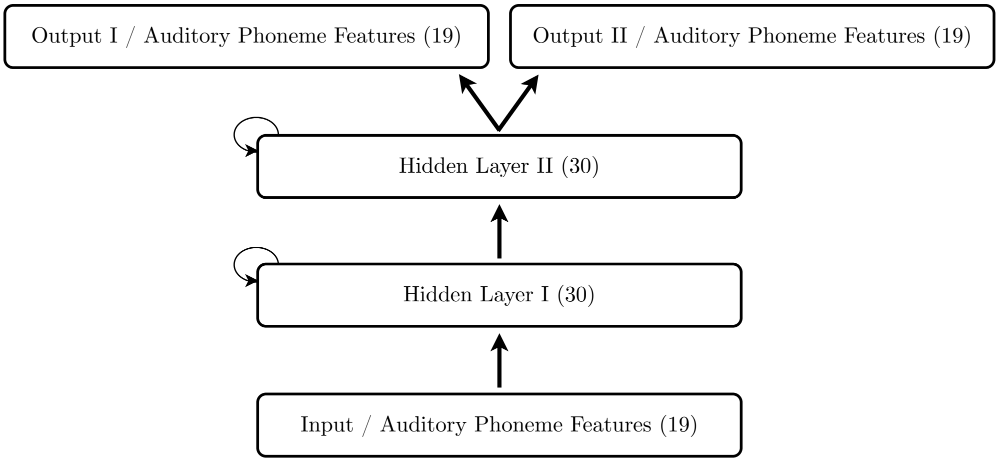

Output I / Auditory Phoneme Features (19)
Output II / Auditory Phoneme Features (19)
Hidden Layer II (30)
Hidden Layer I (30)
Input / Auditory Phoneme Features (19)

Figure 3.12: The network architecture of the phoneme-prediction model used in the gender agreement task. The network takes auditory phoneme features as input and passes them through a series of two hidden layers of memory cells. After processing each input phoneme, the network uses its two output layers to predict the next two phonemes that it will be given as input. The network has approximately 10,000 trainable connection weights.

have shown a network with two hidden layers to be more effective in this case than networks with a single hidden layer.

The network's output layers are each identical to the input layer because the network is predicting upcoming phonemes. There are two such output layers because the network must predict not only the next phoneme that will occur in the input, but the phoneme after that as well. The network makes predictions of two future phonemes because some of the gendered adjective endings consist of two phonemes. For example, the French adjective for the English word *particular* is *particulier* [paʀtikylje] in the masculine and *particuliére* [paʀtikyljɛʀ] in the feminine; the phonetic spellings make plain that the gendered endings of these adjectives differ across two phonemes, with [-e] ending the masculine form and [-ɛʀ] ending the feminine form. Since the network can only be allowed to see the gender-neutral portion of the phoneme sequence—[paʀtikylj-]—without giving away the gendered form intended by the speaker, the network must predict two subsequent phonemes (either of which may be null if subsequent phonemes do not exist) to capture gendered endings with two phonemes such as [-ɛʀ].

The performance evaluation on the gender agreement task after training uses only phrases of the determiner-noun-adjective form because it is the only form that is adjective-final. The testing paradigm requires an adjective-final form because the network must predict the gender-appropriate ending of the last word, and only adjectives generally have two distinct gendered endings. Gender-neutral adjectives and adjectives where the two gendered forms are orthographically distinct but phonetically identical (e.g., in French, the masculine *architectural* and the feminine *architecturale* are both pronounced [aʀʃitɛktyʀal]) are present during gender agreement training but ignored during the performance evaluation.

To determine a baseline level of performance on the gender agreement task, sets of networks trained on either French or Spanish only. Their performance results are shown in Figure 3.13. As was the case with the gender assignment task from the previous section, networks trained on French do well on French and perform at chance on Spanish. Networks trained on Spanish perform as expected on that language and do significantly worse on French.

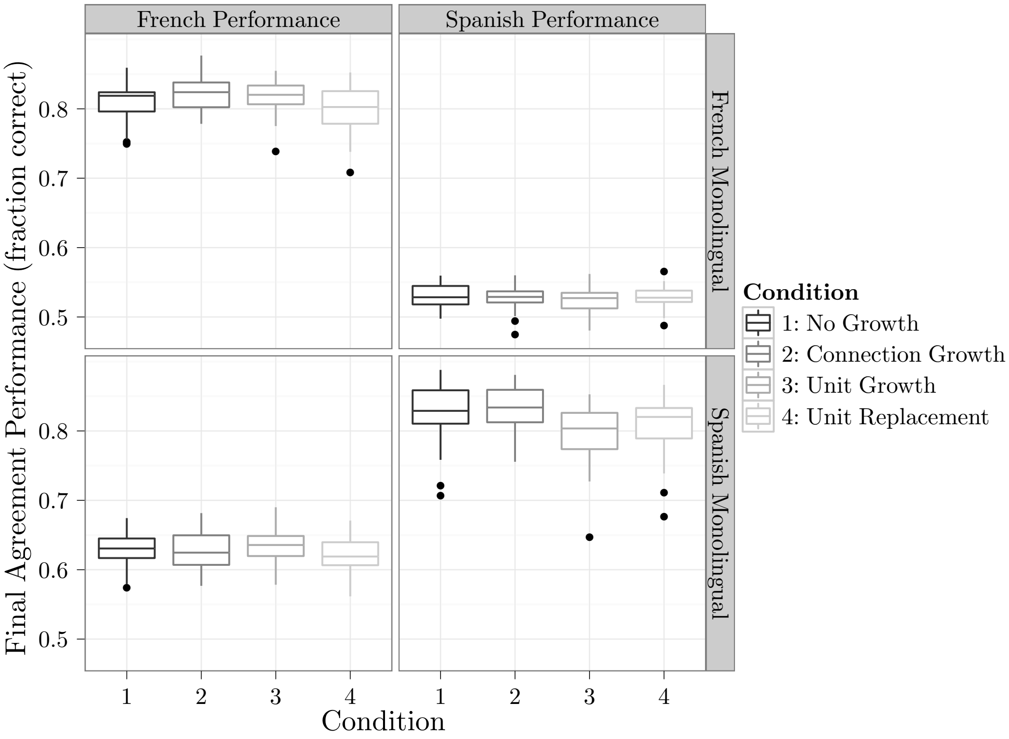


Figure 3.13: Results for monolingual networks, of all three developmental varieties, on the gender agreement task. Thirty separate networks learned the task in each developmental condition for each language. Each network was trained for 2 million trials in one language, and then evaluated on both languages.

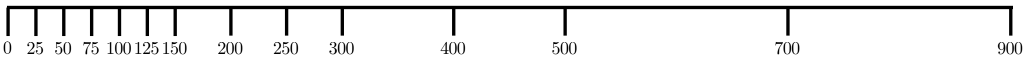


Figure 3.14: Values of t used in the agreement experiments (in thousands of trials). Since little change was observed for large values of t, the interval was expanded from the assignment experiments to allow the use of one less value of t and thereby save on simulation time.

The investigation of bilingual networks involves the same experimental setup as on the gender assignment task to investigate the effects of L1 entrenchment alone (i.e., the no growth condition) and together with network maturation (i.e., the unit growth, unit replacement, and connection growth conditions) on the gender agreement task. As before, training consists of two periods, the first consisting of t trials in which inputs come exclusively from the designated L1, and the second consisting of 2 million trials where inputs may be drawn from either language. Thirty networks learned the task in each maturation condition and for each value of t—the duration of the initial L1-only training period—in Figure 3.14.

Figure 3.15 shows the networks' performance on their first languages, broken out by language and maturation condition as before. As expected, performance is approximately flat with respect to changing t across all the conditions.

Figure 3.16 shows the final performance scores on L2 for networks in each condition of the gender agreement task as a function of t on the x-axis. The results for both languages here are similar to that observed in the gender assignment task for the case of L2 French. The mature networks in the no

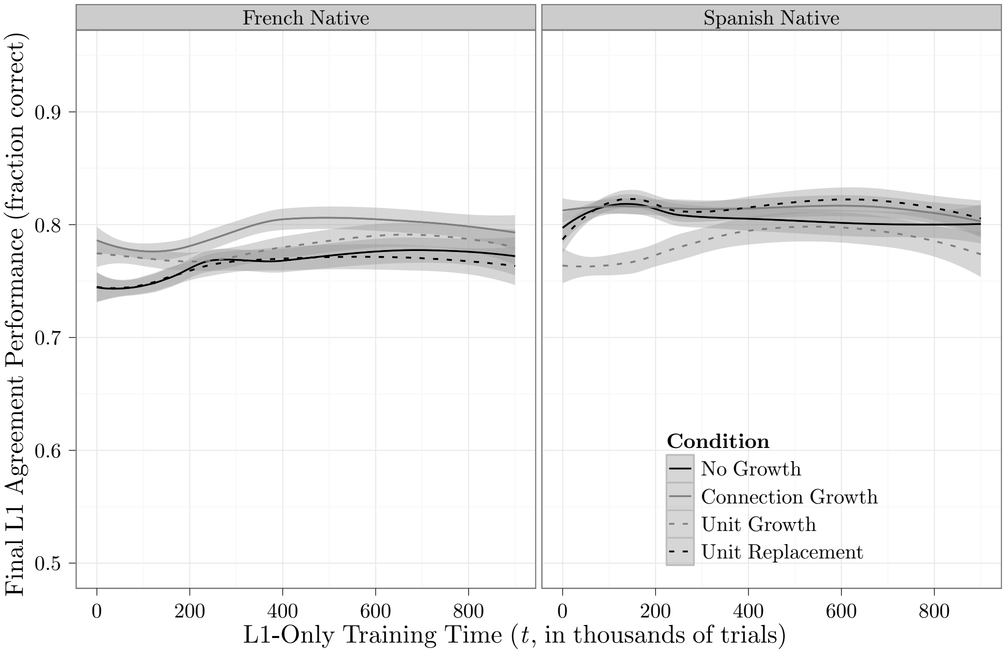


Figure 3.15: Final performance (fraction of test trials correctly answered) for bilingual networks tested on the gender agreement task in their native language after training was complete. The x-axis varies the time each network spent with its native language in isolation before the second language was introduced.

growth condition show a marked susceptibility to L1 entrenchment, with L2 performance decreasing by as much as 17% as t is increased, delaying the onset of L2 relative to L1. However, the networks in the unit growth condition were largely able to mitigate this performance decrease by introducing new units and connections during learning. Performance of networks in the connection growth condition fall between these two. The addition of new connections to the networks appears to successfully stave off entrenchment effects when the level of entrenchment is small, but for values of $t > 200{,}000$ the entrenchment effects again start to become apparent. This suggests that addition of new units and new connections both help to counteract deficits due to entrenchment. Viewed from the cognitive perspective, growth in long-term memory capacity—in the connection growth condition—during training helped to mitigate the effects of L1 entrenchment, as did growth in working memory capacity, as evidenced by the superior performance of the unit growth condition over the connection growth condition for higher values of t. However, as shown by the unit replacement networks again tending to track the performance of the no growth networks, the addition of fresh neural resources is not all that is required to reap a performance benefit. Instead, it seems that starting small, either in terms of working memory capacity or long-term memory capacity, or both, is an essential factor leading to the performance increase.

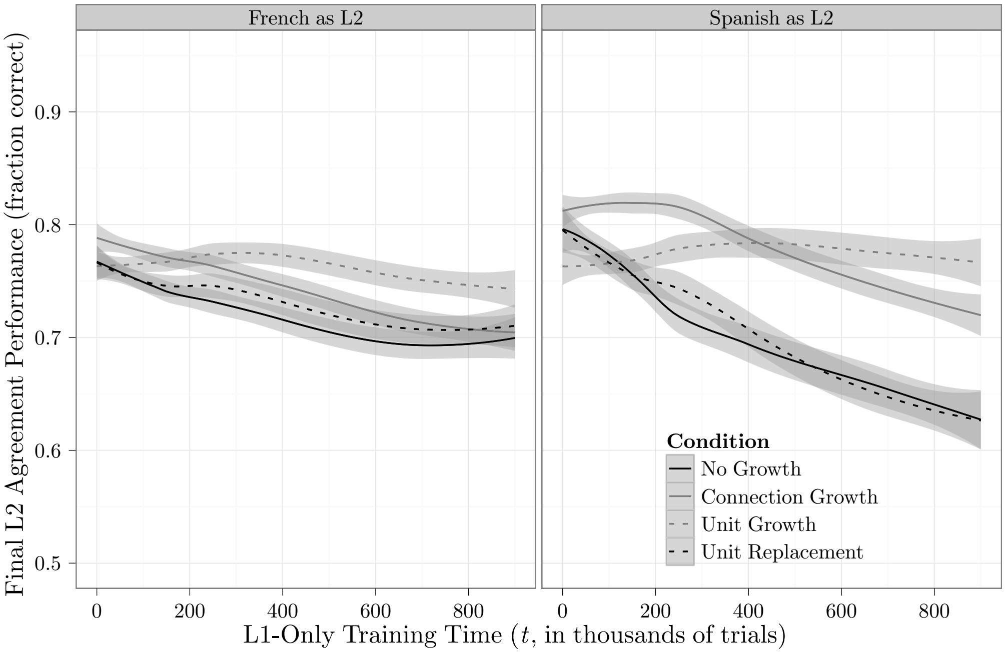


Figure 3.16: Final performance (fraction of test trials correctly answered) for bilingual networks tested on the gender agreement task in their second language after training was complete. The x-axis varies the time each network spent with its native language in isolation before the second language was introduced.

3.5 Discussion

With the exception of gender assignment in Spanish as an L2—by far the easiest task—the data presented in Section 3.4 support the predictions of established ideas of L1 entrenchment. The most dramatic of these can be seen clearly in the no growth conditions, which show an initially steep decline in learnability of the L2 task as time spent on the L1 task increases.

While this modeling approach does not directly implement cognitive constructs such as working memory capacity, Section 3.3.2 argued that the connection growth condition could be reasonably conceived as representing growth from an initially small long-term memory capacity, and the unit growth condition as growth of both long-term and working memory capacities from small beginning states. Allowing the networks to mature in either of these conditions helped to mitigate the negative impacts of L1 entrenchment, especially for longer delays in L2 onset. The fact that the connection growth condition generally improved upon the no growth condition suggests that growth of long-term memory capacity may be a key maturational factor during language learning. For the longest delays, the unit growth condition appears to have had the greatest positive impact, which suggests that growth of working memory capacity also has a positive influence in combating entrenchment effects.

The unit replacement condition, on the other hand, demonstrated the effects of adding fresh long-term and working memory resources to the network without starting small and without changing the network's overall size. The fact that the networks in this condition did not do substantially better than those in the no growth condition implies that the only thing lacking in the unit replacement condition—beginning from resources of modest capacity or starting small—is an essential factor underlying the performance gains made by the unit growth and connection growth networks. This lends support to the "less is more" hypothesis and clarifies that both the initial size and resource acquisition during training are crucial.

The "less is more" hypothesis is usually presented at the cognitive level, suggesting that a system with limited cognitive resources will latch on to the low-hanging organizational fruit, learning representations efficient enough to accommodate its small memory capacity. This can serve as a boon later on, when new memory capacity is added and the system can tackle more complex stimuli. This proposal also makes intuitive sense at the level of neural information processing for a variety of reasons. A network that has its full complement of resources when learning begins naturally learns to use all the resources at its disposal to widely distribute its learned interpretations of its L1 experiences. If an L2 is introduced later, the distributed L1 experience cannot be easily or quickly consolidated to make room for independent L2 learning. Instead, the L2 and L1 experiences intermix and interact, creating L1 entrenchment effects seen as performance deficits in L2. On the other hand, a network that begins training with more modest resources will be forced to attempt to encode the L1 using only the limited resources available.

Though these may initially be insufficient for a full understanding of L1, the limitations will force the network to adopt more efficient and less widely distributed encodings of the L1. This may entail segmenting the input into smaller generative chunks, like phonemes and morphemes. This consolidation of L1 knowledge in the resources that were added early leaves the later-added neural resources free to adapt to novel data such as that presented by an L2.

If this reasoning is correct, starting with fewer resources and building them up during language learning are key strategies to developing more modular representations for each language, which help to avoid the deleterious effects of L1 entrenchment. In short, entrenchment contributes to observed critical period effects, while childhood memory development helps to counteract them. Of course, this should not be interpreted as an assertion that the other purported causes of critical period effects, mentioned in Section 3.2, are spurious; on the contrary, there exists credible evidence for each of them. Rather, the purpose of this work is to help disentangle the confounded effects of entrenchment and memory development, demonstrating how each might contribute to the larger picture of age-correlated declines in second-language attainment.

These results also support the notion that the LSTM-g algorithm trains neural networks in a way that captures important artifacts of human learning, such as L1 entrenchment effects and the mollifying influence of memory development. While LSTM-g is not the first neural network training algorithm to reproduce some evidence of critical period effects, this confirmation helps to justify the use of LSTM-g for training neural networks that aim to model human learning, such as those presented in subsequent chapters.

Chapter 4

Grounding Language in Vision

4.1 Introduction

With the generalized neural network training algorithm from Chapter 2 in hand, and with the confirmation of the approach's resemblance to human learning provided by Chapter 3, this chapter begins to present the sentence-level language modeling that is the main focus of this dissertation. The model presented here completes the first half of the larger language acquisition task. Through observation, it gains an understanding of spoken natural language sentences that is grounded in terms of a visual micro-world. In this case, the problem of **grounding** is that of forming one's representation of language in terms of other, nonlinguistic types of experience. Simple examples of grounding include associating the word *red* with visual images of red things or physically understanding the shape intended when hearing about a *block*. Cognitive philosophers (e.g., Harnad, 1990) have argued convincingly that grounding is essential to forming a true understanding of linguistic meanings, as opposed to a mere understanding of how to manipulate linguistic symbols. Most symbolic and statistical models of natural language processing do not directly address this problem.

As in the previous chapter, the discussion presented here compares the model's learning to human learning. In this case, the analogy is tied directly to theories of cognition, which expect human mental representations to have the combinatorial power of symbol systems. More than twenty years ago, Fodor and Pylyshyn famously rebuffed proponents of the burgeoning connectionist movement, stating that "when construed as a cognitive theory, [...] Connectionism appears to have fatal limitations" (1988, p. 48). Central to their argument was the idea that classical theories of cognition, based on combinatorial symbol systems, possessed a key component—systematicity—that connectionist models seemed to lack. Systematicity, here, is the notion that comprehension of a given situation is intrinsically connected to the comprehension of related ones. Put another way, the ability to process novel situations relies directly on skills learned from similar, previously processed situations. Systematicity is routinely observed in human behavior, particularly in language. For example, if one understands the statement *the girl loves the boy*, one certainly can also process *the boy loves the girl.* Fodor and Pylyshyn argued that, practically by definition, a combinatorial symbol system—one in which entities can be combined and composed according to rules—is the only way to tractably describe systematicity as humans exhibit it. Faced with the work of the connectionists of their time, Fodor and Pylyshyn interpreted the movement as an attempt to replace classical cognitive theory with neural networks that, on the face of it, lacked the ability to operate on symbols systematically. They concluded that the only way neural networks could behave systematically was to directly implement a combinatorial symbol system. This would relegate connectionism to an explanatory level below—and in their view, unlikely to be relevant to—cognition.

Many of the connectionists responded to Fodor and Pylyshyn's assertions with neural network models demonstrating systematic behavior. However, most of these efforts were susceptible to the argument that they were not systematic enough to contradict Fodor and Pylyshyn's claims. This disconnect was possible because Fodor and Pylyshyn had given no straightforward means to measure or otherwise evaluate systematic behavior. To clarify matters, Hadley (1994) operationalized systematicity as it pertains to the use of natural language. Hadley defined systematicity in terms of a learner's ability to generalize from experience to correctly process novel sentences. To satisfy each new level of linguistic systematicity, a learner must make ever-larger leaps from the examples on which it has been trained. This concrete definition of systematicity is useful in that it includes only those forms of systematic behavior in which humans are known to engage. At the time, no connectionist model could achieve the highest level, *strong semantic systematicity*, which bolstered the arguments of Fodor and Pylyshyn.

Shortly after these systematicity requirements were codified, however, Hadley and Hayward (1997) and Hadley and Cardei (1999) advanced related models that met them. These models took simple sentences as input and produced something like a propositional representation of each sentence's meaning as output. The models were largely connectionist in form, consisting primarily of simple processing elements and trainable weighted connections between them, and in training utilizing Hebbian learning methods as well as self-organizing maps (Kohonen, 1990). However, the models' output layers possessed a "primitive, but combinatorially adequate conceptual scheme prior to verbal training" (Hadley and Hayward, 1997, p. 33). This built-in conceptual scheme consisted of single-node representations of each atomic symbol as well as nodes for thematic roles and the binding of the two into semantic compounds. Because all of the atomic symbols and all potential bindings were innate, these models in fact subsumed the combinatorial symbol systems whose mappings they were attempting to learn, imparting all the related systematicity for free. Indeed, the stated goal of these models was to "demonstrate that semantic systematicity is theoretically possible when connectionist systems reflect *some* classical insights" (Hadley and Hayward, 1997, pp. 10–11). Seeing as they implemented classical symbol systems in the most literal way possible—as part of the neural architecture itself—these systems did not serve as counterexamples to the claims of Fodor and Pylyshyn.

Recently, however, Frank et al. (2009) advanced a much more convincing neural network that they claimed was capable of demonstrating strong semantic systematicity. Their model consisted of a simple recurrent network (SRN; Elman, 1990) that learned distributed internal representations via error back-propagation (Rumelhart et al., 1986). Thus, the model was purely connectionist—precisely the type of model that Fodor and Pylyshyn seemed to argue could not be systematic. The Frank et al. model learned to map an input sentence to a **situation vector**—an analogical representation of the network's beliefs about possible states of the world. The situation vector that the model produced in response to a given sentence was not directly interpretable, as a symbolic output would be. However, by manipulating these vectors, Frank et al. were able to quantify the network's level of belief in any particular state of the world. To explain their model's systematic behavior, Frank et al. said that it "comes to display systematicity by capitalizing on structure present in the world, in language, and in the mapping from language to events in the world" (Frank et al., 2009, p. 2). They claimed further that their model counters Fodor and Pylyshyn's assertion, since it does not contain an innate embedded symbol system.

The Frank et al. model, though completely distributed and purely connectionist in nature, appears to possess the same combinatorial properties as a classical symbol system. But these combinatorial properties are largely what defines such a symbol system, leading directly to the suspicion that the Frank et al. model must in fact function as a symbol system, even though, on the surface, it is far from obvious how it might do this. One might hypothesize that the Frank et al. model implements a combinatorial symbol system in a much more interesting way than previous attempts. Instead of an innate, hard-wired neural symbol system like the one present in Hadley and Hayward's model, it seems much more likely that the Frank et al. model instantiates, through learning, a latent neural symbol system. The term **latent** here means that the symbol system is not visible at the level of the neural architecture; rather, it exists entirely as part of the learned internal representations of the network. This would require the network to learn distributed representations that it can manipulate in combinatorial ways, thus gleaning the systematicity benefits of classical cognitive symbol systems while preserving the distributed, incremental nature and graceful degradation of connectionist representations. Frank et al. have not publicly analyzed their model's internal representations in ways that would reveal a latent symbol system, complicating the evaluation of this hypothesis.

To examine the possibility of the emergence of a latent symbol system through training in a neural network, the remainder of this chapter presents a novel neural model that learns to ground a micro-language in a micro-world (Monner and Reggia, 2011, under review). Section 4.3.1 first demonstrates the model's systematic behavior, and Section 4.3.2 subsequently presents evidence that the trained network has created a latent symbol system inside itself.

This model, termed the **grounded-meaning model**, has two parallel, temporal input streams representing a visual scene and an auditory sentence. It is tasked with interpreting the sentence it hears in the context of the visual scene, producing as output a corresponding propositional representation of the meaning of the sentence. Put differently, the network needs to identify the intended referents and relations described in a given sentence and construct the sentence's grounded meaning with respect to those referents. During training, the network learns to segment the input sentence—presented at the phoneme level—into morphemes, words, and phrases. It develops efficient working memory representations of the collection of objects in the visual scene. It learns to map both singular and plural noun phrases onto a referent or referents in the scene. Crucially, the model behaves systematically, generalizing not only to novel sentences and scenes but to objects and descriptions never before encountered. By way of explaining this systematic behavior, Section 4.3.2 provides a detailed analysis of the internal representations learned by the network, showing how it manipulates these representations combinatorially, consistent with a learned implementation of a symbol system.

4.2 Methods

4.2.1 Task description

The grounded-meaning model learns to interpret a small subset of English in terms of a micro-world. Given input streams representing a visual scene and an auditory sentence, the network must combine these streams to create an output representation of the intended meaning of the speaker. By way of explaining the task, this section describes each of the streams of information that the network receives and produces: the input scene represented by the visual stream, the input sentence represented by the auditory stream, and the output grounded meaning represented by the intention stream. Figure 4.1 depicts the general structure of the inputs and outputs of the task, and the following sections explain each input and output sequence in detail.

4.2.1.1 Visual input stream

Each trial in the grounding task involves a randomly generated visual scene. A **scene**, in this context, is an abstract representation of a collection of up to four objects, each of which has three properties: a shape, a color, and a size. Each of these properties is described by discrete values:

- `Shape:` `block`, `pyramid`, or `cylinder`
- `Color:` `red`, `green`, or `blue`
- `Size:` `small`, `medium`, or `large`

Thus, 27 unique objects are possible. In the text, scene objects are denoted in a fixed-width font enclosed in square brackets, as, for example, `[small blue pyramid]`.

The model receives input corresponding to a list of objects in the scene, each of which is identified by its property values and also by a unique, identifying number. The latter allows the model to discriminate between objects that otherwise have identical attributes, allowing a given scene to contain `[large red block 1]` and `[large red block 2]` simultaneously while allowing the model to transparently refer to either. These objects are presented

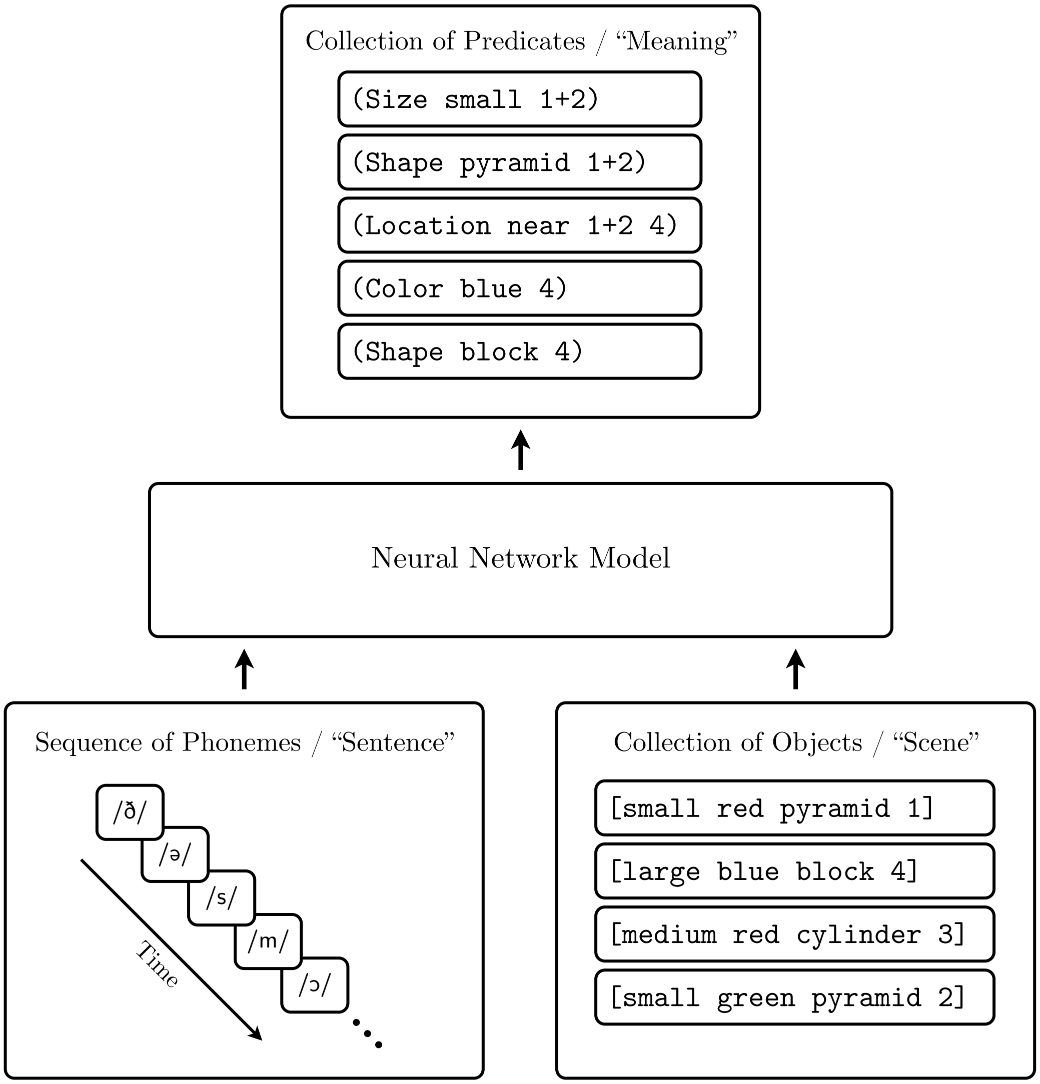
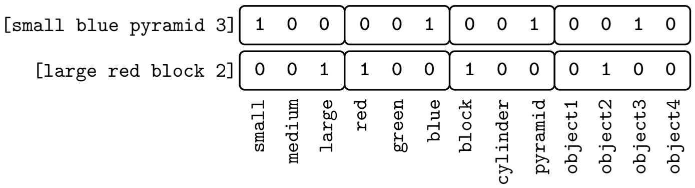


Figure 4.1: An overview of the inputs and outputs that comprise the task. The auditory input is depicted on the lower left and is comprised of a sentence represented as a temporally ordered sequence of phonemes. The visual input on the lower right consists of a collection of objects comprising a scene; in practice, these objects are presented to the model in a random order, one at a time. The output that the model is expected to produce is a meaning for the input sentence comprised of predicates involving objects from the input scene. In practice, the model must produce the predicates as a temporal sequence, one predicate at a time, in the order in which their corresponding phrases appear in the input sentence.



	small	medium	large	red	green	blue	block	cylinder	pyramid	object1	object2	object3	object4
[small blue pyramid 3]	1	0	0	0	0	1	0	0	1	0	0	1	0
[large red block 2]	0	0	1	1	0	0	1	0	0	0	1	0	0

Figure 4.2: Two examples of objects together with their associated neural activity patterns. The first three groups of bits denote the size, color, and shape of a given object, respectively. The final group of bits provides a single-unit identifier for each object, which remains unique in a given scene; this allows the network to transparently refer to any object, even in the presence of objects with duplicate attributes.

to the model as neural activity patterns that combine single active units representing each property value with a single active unit representing the unique identifier. Figure 4.2 depicts two example objects and their associated neural activity patterns as labeled binary vectors, where a 0 represents an inactive neural unit and a 1 represents an active unit.

These neural activity patterns—each representing an object in the scene—are presented to the visual input layer of the network in a temporal sequence called the **visual stream**. During training, the network's visual pathway must learn to create distributed representations that can simultaneously encode several objects, maintaining the bindings between individual objects and their (likely overlapping) attributes.

Clearly, the model could utilize more biologically faithful representations, such as a retinotopic depiction of the visual input. The grounded-meaning model could be extended to work with such an input representation. However, since the main concern here is language acquisition, it makes sense to use the more computationally expedient representation just described, which is closer to symbolic than sensory in nature. Something akin to this abstract object representation could be constructed by higher-level visual regions in response to sensory input. The model receives a scene's objects as a temporal sequence, rather than a single aggregate neural activity pattern, in part because this allows variation in the number of objects presented, which is important in light of research on the capacity limitations of human working memory (see Cowan et al., 2005, for a review). Working memory research shows that, while there are practical limits on the number of object representations an individual can maintain simultaneously, these limits are not hard and fast. A static array of object input units would provide this capacity limit *a priori*, while with a temporal representation, network dynamics determine the capacity limitations for the model's equivalent of working memory.

The visual stream is presented in its entirety before the presentation of the auditory stream begins. This way, the model needs to build a full working memory representation of a scene before hearing a sentence about that scene. This forces the model to consult its memory to ground the sentence, as opposed to consulting the external scene directly. Thus, to perform well on the task, the model must learn to maintain the important attributes of the scene in working memory, using an efficient scene representation—one that the network can later deconstruct and match against the auditory input.

4.2.1.2 Auditory input stream

After experiencing the visual stream, the model hears a sentence involving some of the objects in the scene. Sentences are generated from a simple, mildly context-sensitive grammar (Figure 4.3) that describes objects from the scene and relations between them. Crucially for systematicity testing, the grammar allows many ways to refer to any particular object. For example, a `[small blue pyramid]` could be described as a *small blue pyramid*, a *blue pyramid*, a *small pyramid*, or simply a *pyramid*. Notably, the grammar allows plural references to groups of objects, such that the pyramid from above might be paired with a `[large blue pyramid]` to be collectively referred to as the *blue pyramids*; or the original pyramid could be grouped with a `[small green cylinder]` to be collectively described as the *small things* because of their one common attribute. Examples of complete sentences that the grammar can generate are *the large pyramids are green*, *the blue things are under the small red blocks*, and *the small green pyramid is on the large red cylinder*.

Each word in an input sentence is transcribed into a phoneme sequence; these are then concatenated to create a single unsegmented sequence of phonemes representing the entire sentence. For example, the sentence *the large cylinder is green* is encoded as the phoneme sequence [ðəlɑɹʤsɪləndɚɪzgɹin]. Each phoneme in such a sequence is input to the network as a neural activity pattern representing some of the acoustic features

S	→	**NP VP**
NP	→	the [**Size**] [**Color**] **Shape**
VP	→	**Is Where** | **Is Color** | **Is Size**
Is[1]	→	is | are
Where[2]	→	on **NP** | under **NP** | near **NP**
Size	→	small | medium | large
Color	→	red | blue | green
Shape	→	things | pyramid[s] | block[s] | cylinder[s]

Figure 4.3: The grammar used to generate the sentences. Terminals begin with a lowercase letter, while nonterminals are in boldface and begin with an uppercase letter. The symbol | separates alternative derivations, and terms in brackets are optional.

[1] The evaluation chosen for the **Is** nonterminal must agree in number with its subject **NP**.

[2] The object **NP** of *on* cannot describe a pyramid since, in the micro-world, pyramids cannot support other objects due to their pointed tops. Similarly, the subject **NP** preceding *under* cannot describe a pyramid. In other words: Nothing can be *on* a pyramid, and a pyramid cannot be *under* anything. This restriction on pyramids will help to illustrate how the network internalizes micro-world systematicity in Section 4.3.2.

of the phoneme that are known to be used for identification by human learners. Such features include voicing, affrication, and formant frequencies, and are modeled after previous feature sets used in similar applications (Weems and Reggia, 2006; Schulz and Reggia, 2004). The full list of phonemes and features is given in Table 4.1. The features used here are more obviously acoustic and auditory than the set used in Chapter 3 and provide a useful contrast with the motor features that are used to describe the same set of phonemes in Chapter 5. The model's direct use of such features perhaps pushes it a step away from the human auditory sensory system, which at its basest level uses frequency-tuned neurons. However, these features have been identified as playing crucial roles in phoneme recognition (Jakobson et al., 1951; Singh, 1976; Paget, 1976), and the low-level auditory system is thought to contain similar representations (Phillips, 2001). The model developed in this chapter adopts this more computationally tractable method of representing speech sounds. Importantly, this set of features—or more likely a superset containing a few features that were not necessary here—represents a language-agnostic basis in which speech sounds can be described. Since the features can describe any phoneme, and the model can handle any sequence of such phonemes, a model based on these features could potentially represent words and sentences from any spoken human language.

These patterns—representing the phonemes in the sentence—are presented at the auditory input layer of the network as a temporal sequence termed the **auditory stream**. During training, the auditory pathway must simultaneously learn to segment the auditory stream into morphemes and words (for which boundaries are not included), pay attention to the syntactic relations between these elements, and discover the cues that identify objects and relations.

4.2.1.3 Intention output stream

After receiving both the visual and auditory streams, the network is tasked with constructing the sentence's meaning in the context of the scene. To do this, the network must combine its auditory and visual working memory representations to generate a sequence of **predicates**—as activity patterns over its output layer—called the **intention stream**. Each predicate

Table 4.1: Binary Acoustic Features of Heard Phonemes

	Phoneme																																			
Feature	b	d	e	f	g	h	i	j	k	l	m	n	o	p	s	t	u	v	w	z	æ	ð	ɳ	ɑ	ɔ	ə	ɚ	ɛ	ɪ	ɹ	ʃ	ʊ	ʌ	ʒ	ʤ	ʧ	θ
Consonantal	+	+	−	+	+	+	−	+	+	+	+	+	−	+	+	+	−	+	+	+	−	+	+	−	−	−	−	−	−	+	+	−	−	+	+	+	+
Vocalic	−	−	+	−	−	−	+	−	−	+	−	−	+	−	−	−	+	−	−	−	+	−	−	+	+	+	+	+	+	+	−	+	+	−	−	−	−
Compact	−	−	−	−	+	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	+	−	−	+	+	+	−
Diffuse	+	+	−	+	−	−	−	−	−	−	+	+	−	+	+	+	−	+	−	+	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+
Grave	+	−	−	+	−	−	−	−	−	−	+	−	−	+	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
Acute	−	+	−	−	−	−	−	−	−	−	−	+	−	−	+	+	−	−	−	+	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+
Nasal	−	−	−	−	−	−	−	−	−	−	+	+	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−
Oral	+	+	−	+	+	+	−	+	+	+	−	−	−	+	+	+	−	+	+	+	−	+	−	−	−	−	−	−	−	+	+	−	−	+	+	+	+
Tense	−	−	+	+	−	+	+	−	+	−	−	−	+	+	+	+	+	−	−	−	−	−	−	+	−	−	+	−	−	−	+	−	−	−	−	+	+
Lax	+	+	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	+	−	+	+	+	−	−	+	+	−	+	+	−	−	+	+	+	+	−	−
Continuant	−	−	−	+	−	−	−	−	−	−	−	−	−	−	+	−	−	+	+	+	−	+	−	−	−	−	−	−	−	−	+	−	−	+	−	−	+
Interrupted	+	+	−	−	+	−	−	−	+	−	−	−	−	+	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	+	−
Strident	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	+	−
Mellow	−	−	−	−	+	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+
+Voicing	+	+	+	−	+	−	+	+	−	+	+	+	+	−	−	−	+	+	+	+	+	+	+	+	+	+	+	+	+	+	−	+	+	+	+	−	−
−Voicing	−	−	−	+	−	+	−	−	+	−	−	−	−	+	+	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	+	+
+Duration	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	+	−	−	+	−	−	−
−Duration	+	+	−	+	+	+	−	+	+	+	+	+	−	+	−	+	−	+	+	−	−	+	+	−	−	−	−	−	−	+	−	−	−	−	+	+	+
+Frication	−	−	−	+	−	+	−	−	−	−	−	−	−	−	+	−	−	+	−	+	−	+	−	−	−	−	−	−	−	−	+	−	−	+	+	+	+
−Frication	+	+	−	−	+	−	−	+	+	+	+	+	−	+	−	+	−	−	+	−	−	−	+	−	−	−	−	−	−	+	−	−	−	−	−	−	−
Liquid	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−
Glide	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
Retroflex	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	+	−	−	−	−	−	−	−
$F_{2,VH}$	−	−	+	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−
$F_{2,H}$	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−
$F_{2,HM}$	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	+	−	−	−	−	−	+	−	−	−	−
$F_{2,LM}$	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−
$F_{2,L}$	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	+	−	−	−	−	−
$F_{2,VL}/F_{1,VH}$	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−
$F_{1,H}$	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	+	−	+	−	−	−	−	−	−	−	−	−	−	−
$F_{1,HM}$	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	+	−	−	−	−	−	−	−	−	−
$F_{1,LM}$	−	−	+	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−
$F_{1,L}$	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−
$F_{1,VL}$	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−

in the intention stream corresponds to an attribute or relation mentioned in the sentence. Predicates are represented in the model as neural activity patterns, but they are denoted in the text using a fixed-width font enclosed in parentheses, distinguishing them from the square-bracketed visual objects. If a sentence refers to the visual object `[small red cylinder 2]` as *small cylinder*, the network should produce the predicates `(Size small 2)` and `(Shape cylinder 2)`, but not `(Color red 2)`, since this attribute was not mentioned. If a sentence states that a *blue block* (referring to visual object 3) is *near* the small cylinder, the network must output the predicate `(near 3 2)`. Figure 4.4 shows two examples of predicates from the intention stream along with their associated neural activity patterns.

In general, the order of the predicates in the intention stream follows the order inherent in the auditory stream. That is, the intention stream for *the blue block is near the green block* will begin with predicates describing *the blue block*, followed by predicates describing the relation *near*, and ending with predicates for *the green block*. It may be that some objects in the scene, or even most of them, are not referenced in the sentence that accompanies it. In this case, these objects can be considered distractor stimuli, and while they are present in the visual stream input, they are not included in the target intention stream.

After a training trial, the network is shown the target intention stream. Comparing this behavior to that of a human language learner requires the

	Size	Color	Shape	Location	near	on	under	small	medium	large	red	green	blue	block	cylinder	pyramid	object1	object2	object3	object4	object1	object2	object3	object4
(Color blue 2)	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
(Location on 1+2 3)	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0

Figure 4.4: Two examples of predicates together with their neural activity patterns. The first group of units denotes the type of the predicate. For the attribute-based predicate types `Size`, `Color`, and `Shape`, the arguments consist of an activated attribute value unit and one or more object identifier units, all active in the first group of such units. For the `Location` predicate, the arguments are a relational term like `near` or `on` and two object identifiers—or groups of identifiers—with each argument activating specific units in its own group of identifier units. Above, the first example conveys the blueness of object 2, while the second example states that objects numbered 1 and 2 are both located on object 3.

assumption that the learner can, at least sometimes, derive the speaker's meaning from other sources—a task at which language learners seem to excel (Tomasello, 2003)—and that this meaning is available in something resembling a propositional form. This is perhaps not the most plausible of assumptions, and the final model presented in Chapter 5 does not require it.

4.2.1.4 Example trial

To concretize the input and output descriptions, this section presents a full trial of the language grounding task—that is, a complete specification of related visual and auditory input streams as well as the desired intention stream output. Figure 4.5 describes an input scene, consisting of four objects, and an input phoneme sequence for the sentence *the small pyramids are near the blue block*. A correct intention stream for these inputs must contain predicates denoting the objects numbered 1 and 2 as the *small pyramids*.

Input Visual Stream	Input Auditory Stream	Intention Stream
`[small red pyramid 1]`	[ðəsmɔlpɹəmədzɑɹnɪɹðəblʊblɑk]	`(Size small 1+2)`
`[large blue block 4]`	(gloss: *the small pyramids are near the blue block*)	`(Shape pyramid 1+2)`
`[medium red cylinder 3]`		`(Location near 1+2 4)`
`[small green pyramid 2]`		`(Color blue 4)`
		`(Shape block 4)`

Figure 4.5: An example trial of the language grounding task. Stream elements are depicted in human-readable form, but are presented to the network as sequences of neural activity patterns representing objects, phonemes, and predicates, respectively.

The intention stream should indicate object 4 as the referent of *the blue block*, containing predicates at the end of the sequence matching these two attributes with the appropriate object identifier. For the relation between the objects, the intention stream must contain a predicate representing the `Location` as `near`, indicating that objects 1 and 2, the pyramids, are close to object 4, the block.

4.2.2 Model description

The neural network that learns the grounding task utilizes the long short-term memory architecture (LSTM; Hochreiter and Schmidhuber, 1997; Gers and Cummins, 2000; Gers and Schmidhuber, 2001). LSTM uses stateful self-connected neural units called memory cells, which are allowed to have multiplicative input, output, and forget gates. The biologically motivated generalized long short-term memory (LSTM-g) algorithm, derived in Chapter 2, serves to train the network. LSTM-g is a reformulation of LSTM's original gradient descent training algorithm that gains the ability to accommodate arbitrary multi-level network architectures, while retaining the spatial and temporal locality observed in biological neural networks.

The grounded-meaning model utilizes an LSTM network instead of a more traditional recurrent network such as the SRN, in part because experiments show LSTM to provide both faster convergence and better generalization in complex multi-input, multi-output tasks that require the network to hold extensive information in working memory. Despite many attempts, pilot experiments were unsuccessful in finding an SRN configuration that could learn this language grounding task.

4.2.2.1 Network architecture

The specific network architecture of the grounded-meaning model is depicted in Figure 4.6. The network has a visual processing pathway that begins on the bottom right of the figure, comprised of a visual-stream input layer followed by a visual accumulation layer. The latter is a collection of memory cells that self-organize during training, gaining the capability to integrate the temporal visual stream into a flexible representation of an entire scene.

The network's auditory pathway is set up much the same way and begins on the bottom left of the figure with an auditory-stream input layer followed by two auditory accumulation layers in series. Previous experiments on learning ungrounded language representations, such as those in

Section 2.7.3, showed that a two-layer pathway outperforms a single-layer pathway on the task of integrating a temporal phoneme sequence into a sentence representation. This appears to be due to the multi-level segmentation inherent in the auditory part of the task. When using two layers, the first auditory layer is allowed to specialize in the aggregation of morphemes and words from phoneme sequences, while the second layer chiefly processes relationships between words. It should be noted, however, that performance on the grounding task does not depend crucially upon this two-layer network architecture choice.

To recover the intention stream, the model integrates the sentence representation and the scene representation using another layer of memory cells called the integration layer. This layer's representation is then used to generate an appropriate temporal sequence of predicates on the intention-stream output layer.

The three layers of memory cells in the auditory and visual pathways have input gates and forget gates, but not output gates, as these cells feed exclusively into other, input-gated memory cells. The memory cells in the integration layer, in contrast, have all three types of gates. To assist in the production of output sequences, the integration layer has a recurrent connection from the previous time-step's intention-stream output.

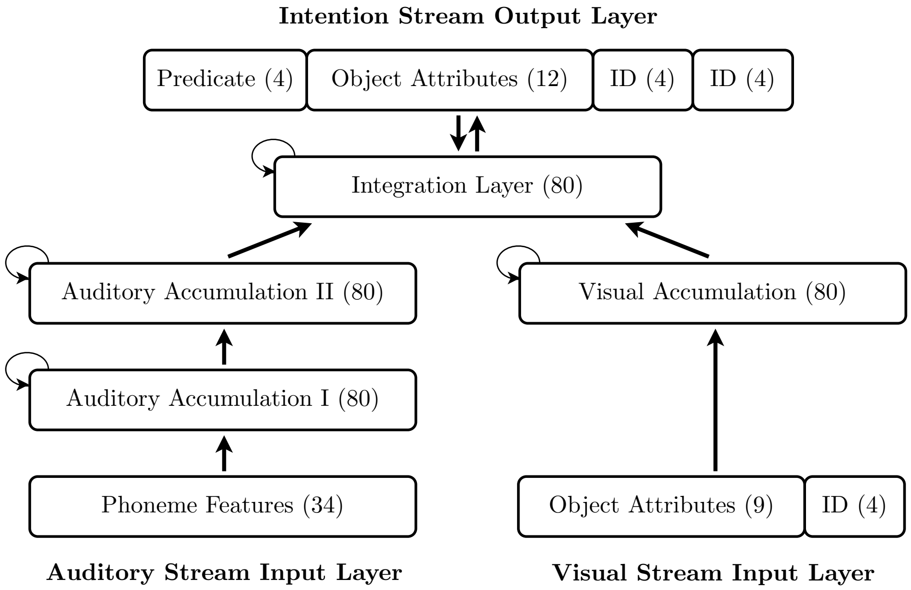


Figure 4.6: The architecture of the network. Boxes represent layers of units (with number of units in parentheses), and straight arrows represent banks of trainable connection weights between units of the sending layer and units of the receiving layer, though generally not all possible connections are present. Layers with a curved self-connecting arrow are composed of memory blocks (collections of memory cells and associated gates as described in Chapter 2; following the convention of Chapter 3, the memory cell layers and gate layers are collapsed into a single box for visual simplicity).

4.2.3 Levels of systematicity

As was the case with Frank et al. (2009), it is difficult to map Hadley's (1994) levels of systematicity onto observable performance attributes of the language grounding task. For example, Hadley's notion of strong semantic systematicity requires the learner to understand the meanings of known words encountered in novel syntactic positions, even in embedded sentences—a demonstration which is not possible with the modest training grammar employed here. To provide a benchmark against which to measure the model, this section defines four levels of systematicity as it relates to the task used here. These levels of systematic grounding correspond roughly in difficulty to those defined by Frank et al.—though not necessarily to those defined by Hadley—while lending themselves more directly to use with this task:

1. **Weak** systematic grounding: The learner can label familiar objects in novel scenes using familiar object descriptions. That is, having seen a `[medium blue block]` before, and referred to it as a *medium blue block* in a particular scene, the learner can generalize this ability to new scenes and new sentences.

2. **Categorical** systematic grounding: The learner can label novel objects in a scene using familiar descriptions. For example, if the learner has never previously seen a `[small red block]`, but *has* encountered objects referred to as *red blocks*, it can apply the label *red block* to this

novel object. This is “categorical” in the sense that applying familiar labels to novel objects is tantamount to lumping them into existing categories.

3. **Descriptive** systematic grounding: The learner can use novel descriptions to label familiar objects in a scene. In this situation, the learner has previously encountered, for example, a `[small blue pyramid]`, but has only ever heard it referred to as a *small pyramid* or a *blue pyramid* or just a *pyramid*. If the learner can successfully apply the label *small blue pyramid*, which it has never heard before, to this object, then it exhibits descriptive systematic grounding.

4. **Strong** systematic grounding: The learner can use novel descriptions to label objects it has never previously encountered. That is, having never encountered, for example, a `[small green cylinder]`, *and* having never heard the label *small green cylinder*, the learner can nonetheless recognize the novel object and refer to it using this novel label.

Section 4.3.1 uses these four criteria to demonstrate that the model presented in this chapter exhibits strong systematic grounding of the language it learns.

4.2.4 Experimental evaluation

This experiment trains copies of the grounded-meaning model in four different ways, evaluating it on sets of test sentences that probe the different levels of grounding systematicity from Section 4.2.3. It is important to note that in each of these four conditions, the test inputs serving as the basis for the performance scores are always novel—the model has never seen them during training. In what follows, an object or description is considered novel if it consists of a combination of features (e.g., `[large red pyramid]`) or words (e.g., *large red pyramid*) that does not occur in the training set. The four training conditions are:

1. **Weak** condition: The set of all possible scene-sentence pairs is partitioned at random with 10% reserved exclusively for testing. While test pairs are completely novel to the model, the individual objects and descriptions are likely to be familiar.

2. **Categorical** condition: One specific type of object—for example, a `[small green block]`—is never present in scenes during training. The model is tested in situations where this novel object is given a familiar description—as a *small block*, a *green block*, or simply a *block*, all labels that the network has previously applied to other types of objects.

3. **Descriptive** condition: One specific type of object—for example, a `[medium blue pyramid]`—while allowed to be present in the scenes, is never described fully; that is, the object can be referred to as, for example, a *blue pyramid* or a *medium pyramid*, but never as a *medium blue pyramid.* The model is scored on scenes containing this famil-

iar object paired with sentences containing the full, novel description, *medium blue pyramid*.

4. **Strong** condition: One type of object—a `[large red pyramid]`, for example—is never described *and* never appears in scenes. The model is scored in situations where this novel object appears and is referenced using the novel description, which in this case is *large red pyramid*.

Ten distinct, randomized copies of the grounded-meaning model learn the task set forth in each of the four conditions. For pairs of layers shown as connected in Figure 4.6, individual pairs of units are connected with a probability of 0.7, leading to networks with 320 memory blocks and approximately 60 thousand trainable weights. The learning rate parameter is set to 0.01. Each network is allowed to train on 3 million randomly selected scene-sentence pairs from its training set.

Each training trial consists of a random scene comprising two, three, or four distinct objects, with uniform probability. The grammar (Figure 4.3) then generates a random sentence that describes some aspect of the scene. Over half a million distinct scenes are possible, each giving rise to, on average, 36 possible grammatical sentences, for a total of approximately 18 million distinct scene-sentence pairs. Since inputs, especially simple ones, are often repeated, the network sees a very small fraction of the input space during training, though the network, like most neural models, still appears to require far more linguistic input than human learners, at least in terms of number of repetitions. For each pair of test inputs, the network must produce the correct intention stream, consisting of a temporal sequence of 2 to 7 predicates.

Before training, each network was evaluated on the set of test sentences for its condition. In each case, the pre-training networks activated the correct output units at levels appropriate for chance performance. No network, during any pre-training run, produced even a single correct output predicate, let alone a correct sequence of predicates corresponding to a sentence. This is consistent with chance performance, given the size of the space of potential outputs.

4.3 Results

4.3.1 Systematicity

Figure 4.7 compares the accuracy of fully trained networks across the four conditions. The ten networks in the weak condition produced all intention stream predicates correctly for, on average, 95% of novel scene-sentence pairs, while those in the categorical, descriptive, and strong conditions were 93%, 93%, and 97% accurate, respectively. The networks clearly pass all of these systematicity tests on the grounding task.

Comparing the conditions reveals a significant difference in performance only between the descriptive condition and the strong condition on a Welch two-sample t-test ($t \approx -3.2$, $df \approx 17.5$, $p < 0.01$). This likely has to do with

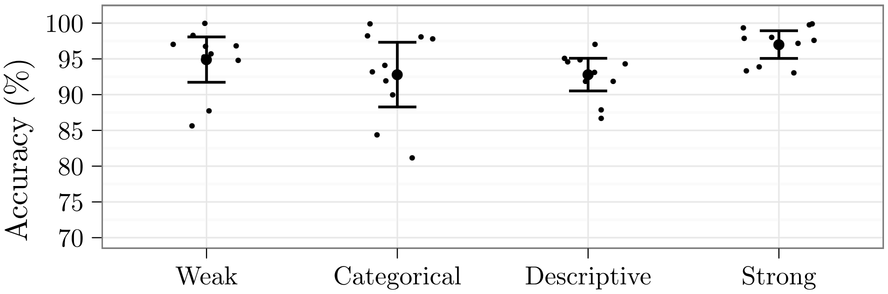


Figure 4.7: The percentage of completely correct intention streams recovered from random samples of 100 test-set sentences, averaged over 10 trials in each of the conditions. The small dots represent the performance of individual networks in a condition, and the large dots represent overall condition means. The error bars denote the 95% confidence intervals on the condition means.

an (intentional) asymmetry in the descriptive condition's training set. The network, observing 27 different visual objects but only 26 complete auditory object descriptions, is slightly impaired by this structural asymmetry. By contrast, in the strong condition, the scenes and sentences maintain their structural symmetry, with 26 visual objects corresponding to 26 complete auditory descriptions.

The network had far more trouble with the grounding part of the task—that is, selecting the referents for the various object descriptions—than it had with parsing the linguistic descriptions themselves. These two subtasks can be scored separately by observing how often the network produces the correct predicate sequence and attribute values—indicating that it processed the linguistic description correctly—and how often it identifies the unique identifier of the correct referent for that description. When scoring on accurate recognition of linguistic descriptions and ignoring referents, trained networks produced, on average, less than one error per 1,000 novel sentences.

While trained networks produced the correct referent for 98% of noun phrases—with their accuracy varying inversely with the number of objects in the scene and the number of referents in the sentence, as one might expect—it also took them much longer to reach this accuracy level, as can be seen in Figure 4.8. A typical network required only the first quarter of its training time to achieve near-perfect accuracy when recognizing linguistic descriptions, at which point it was identifying referents correctly only 80% of the time, a figure that slowly improved for the duration of training. That referents are so much more difficult to identify than object attributes and relations only underscores the difficulty of the language grounding task.

4.3.2 Analysis

The results in the previous section demonstrate that a trained network is capable of using grounded language systematically. To provide insight into how this occurs, this section presents an analysis of the learned internal representations of the network that enable this behavior. The analysis reveals that the network's linguistic representations capture important syntactic and semantic features of the sentences; that the learned representations capture latent micro-world regularities; and that the network's grounded representations over the integration layer appear to be compositional with respect to the

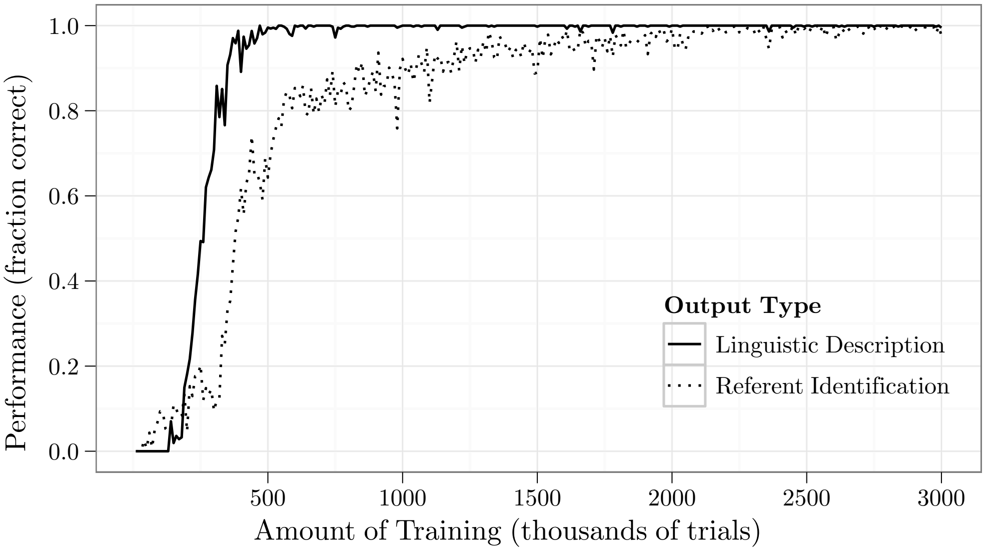


Figure 4.8: Depiction of performance versus training time for a randomly selected network on both linguistic descriptions and referent identifications. The network learns linguistic descriptions quickly, reaching maximal accuracy after a small fraction of the allowed training time. Achieving maximal accuracy on referent identification requires much more training.

components of the desired output predicates, suggesting that the network's learned representations instantiate a combinatorial symbol system.

The data analyzed in this section come from a single, randomly selected trained network from the weak systematicity condition. The choice of a network from the weak condition ensures the absence of systematic biases in the training set that the network experienced. That said, the representations learned by networks in the other conditions were not fundamentally different than the results presented.

The first avenue of investigation involves the information present in learned representations distributed over the network's second auditory accumulation layer. During processing of the auditory streams of each of 100 test sentences, the testing program takes snapshots of this layer's activation values at the end of each heard word. Averaging over all occurrences of a given word provides a canonical representation for that word. These representations are then hierarchically clustered to capture their similarities and differences, resulting in Figure 4.9. The top cluster corresponds to the adjectives present in the training grammar, and the nouns cluster individually below, with the singular and plural versions of the same noun clustering together. The meta-cluster of adjectives breaks down neatly into two clusters based on the function of the adjectives as color or size descriptors, though this may be due to a syntactic difference present in the grammar: Size descriptors always precede color descriptors when the two are present together, as is common in English. Given the amount of structure in these clusters, it is fair to say that the representations in the second auditory accumulation layer capture a basic form of syntactic category information regarding heard words.

Using the same methodology, one can examine the representations of heard words in the next layer downstream, the integration layer. Looking specifically at the words involved in number agreement—the singular and plural nouns and the corresponding verbs *is* and *are*—Figure 4.10 reveals that this layer's representations differ significantly from those of the previous layer. The representations here differ based in part on their number agreement class, with the singular nouns and *is* forming a single cluster, whereas the plural verb *are* is much more closely related to the plural nouns. One might initially

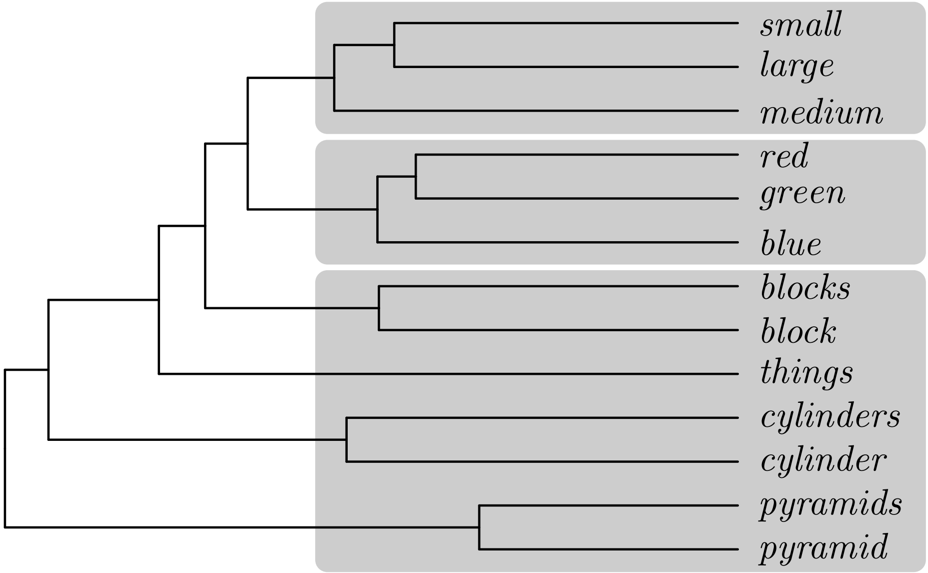


Figure 4.9: A hierarchical clustering of heard word representations from the network's second auditory accumulation layer. The hierarchy shows a tight cluster of adjectives (top), subdivided by type, while the nouns (bottom) cluster by word-form, with singular and plural representations forming individual clusters.

guess that this clustering is simply based on the similar phonemes that end the plural nouns, with all words ending in [s] or [z] forming one cluster to the exclusion of the other words. By this logic, one would expect *is* to cluster with the former group. However, the opposite is observed, suggesting instead that the representational divide reflects number agreement information.

Drilling down further into the data involves moving away from averages over all occurrences of a predicate type to averages over all occurrences of a specific combination of a predicate and its arguments. So instead of averaging over all `Shape` predicates, say, one needs to compute a separate average over all the occurrences of `(Shape block 1)`, `(Shape cylinder 2)`, and so on. A hierarchical clustering of these averages reveals the pattern shown in

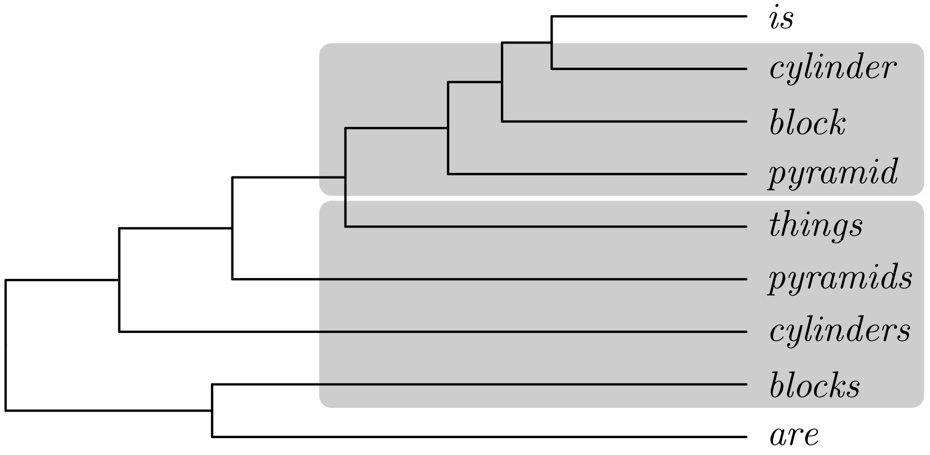


Figure 4.10: A hierarchical clustering of representations of heard words from the network's integration layer. The clustering shows a tight cluster (top) of singular nouns which also includes the singular verb *is*, which *are* is much more closely related to the plural nouns (bottom).

Figure 4.11. One can immediately identify four major clusters corresponding to the four unique object identifiers, suggesting that the representations in this layer systematically separate predicates based on which objects they involve. This hierarchical clustering pattern is also observed for the `Color` and `Size` predicates. This provides a glimpse of how the network performs grounding: It systematically varies its internal representations of predicates based on which micro-world objects those predicates involve.

Interestingly, each of the four main clusters in Figure 4.11 has a similar internal structure, with the `cylinder` and `block` attributes being consistently more similar to each other than to the `pyramid` attribute. This internal structure is due to subtle constraints that the micro-world places on the various shapes: Blocks and cylinders are allowed to support other objects, whereas pyramids are not, due, of course, to their pointed tops. No

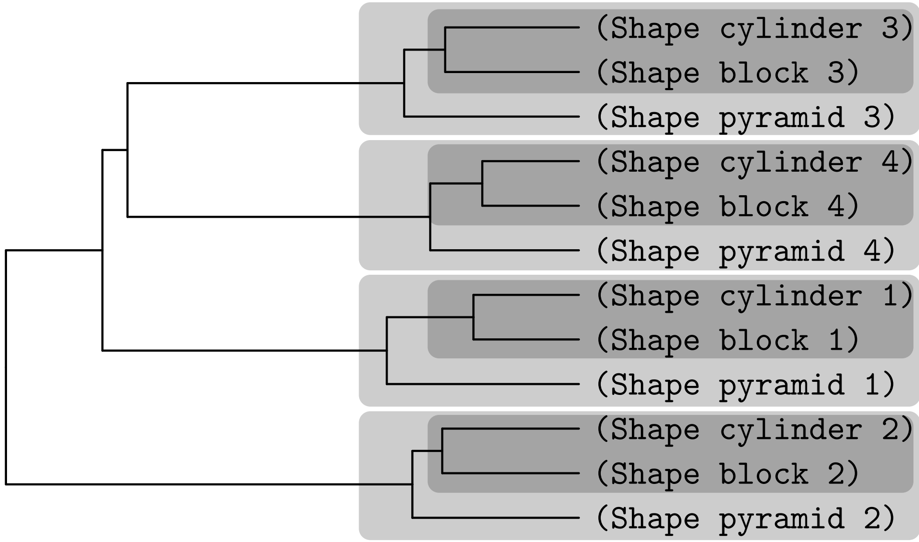


Figure 4.11: A hierarchical clustering of representations from the network's integration layer that co-occur with output predicates. Canonical predicate representations were obtained by averaging over all occurrences of a specific predicate/argument combination. For readability, this hierarchy involves only `Shape` predicates. The representations break down systematically based on the object involved and then by the specific shape involved.

information about the differing functions of the various shapes is ever presented explicitly to the network. The only indication of a difference between shapes is the absence of pyramid objects in certain argument positions of `on` and `under` predicates specifying a `Location`. Nevertheless, the cluster substructure in Figure 4.11 serves as evidence that the network encoded this latent difference in the operational regularities of the micro-world.

To further interpret the integration layer's internal representations, it would be nice to be able to graph them directly. Unfortunately, these representations exist in an 80-dimensional space–one dimension per unit in the layer—rendering them difficult to visualize. An illuminating way to reduce this complexity is via principal component analysis (PCA; Jolliffe, 2002), which creates a new 80-dimensional space by taking linear combinations of the original 80 dimensions in such a way as to account for as much of the variance as possible with each new dimension. Thus, most of the interesting variations in the data will appear in the first few dimensions, or principal components (PCs), of the transformed space. This has the side effect of centralizing redundant parts of the representation into single PCs, allowing a glimpse beyond the distributedness of the representations and making them more amenable to analysis and visualization. This allows the depiction of representations from the integration layer in several 2-dimensional spaces defined by pairs of the first few PCs. That said, the meaning underlying some of the PCs is still likely to be difficult to interpret, so this section only contains graphs for those PCs where the underlying pattern is clear. These graphs offer insights into the symbolic nature of the model's learned internal representations.

Snapshots of integration-layer representations that precede predicate outputs in the intention stream provide the data for Figure 4.12, which graphs the average integration-layer representations for predicates involving each separate object identifier. These average representations, denoted by large numbers representing each identifier, are very clearly separated when viewed in the space defined by PC1 and PC4, indicating that the network has no trouble determining which identifier belongs in a given predicate. This figure also breaks averages down further, showing sub-averages over each predicate type for a given object identifier, denoted by smaller text combining each

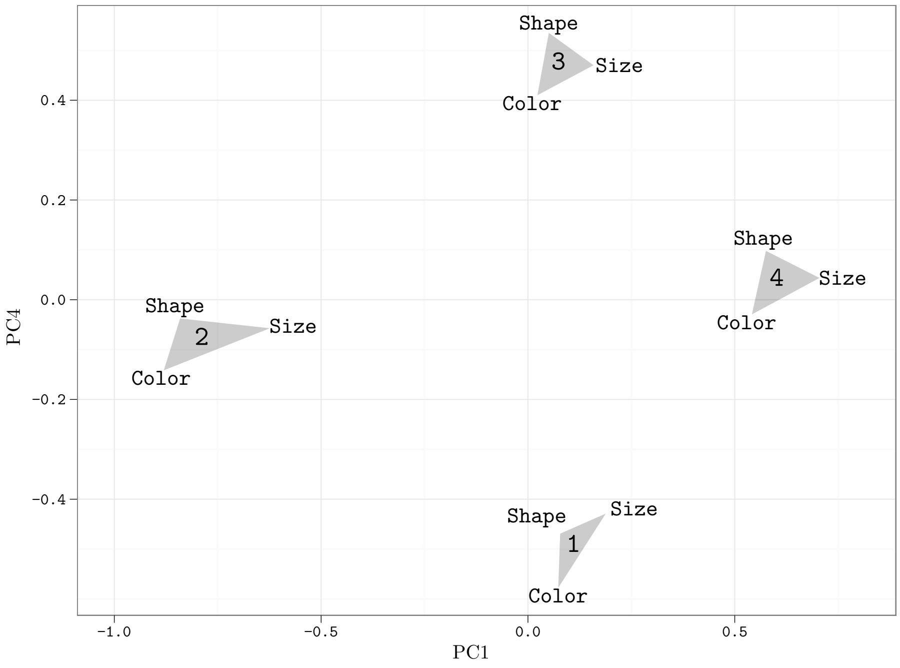


Figure 4.12: A PCA-space view of the integration layer's representational clusters (highlighted in gray polygons) for each of the four possible object identifiers (the large numbers). The sub-structure labels of each cluster show representational averages over predicate type for each identifier. There is systematic variation of each predicate type from the cluster center: `Shape` predicates above, `Color` predicates below and to the left, and `Size` predicates to the right and slightly above. This systematic variation is evidence of combinatorial symbolic representations.

predicate name with the appropriate identifier. One notes immediately the similarity in the sub-structural organization of these clusters: The `Shape` predicates always appear above the cluster center, while the `Color` and `Size` predicates appear, respectively, below and to the right.

A complementary pattern emerges in Figure 4.13, which visualizes averages over all instances of `Color`, `Size`, and `Shape` predicates, which are well-separated when viewing PCs 2 and 6. The sub-structure of these clusters depicts sub-averages over occurrences of each predicate type involving

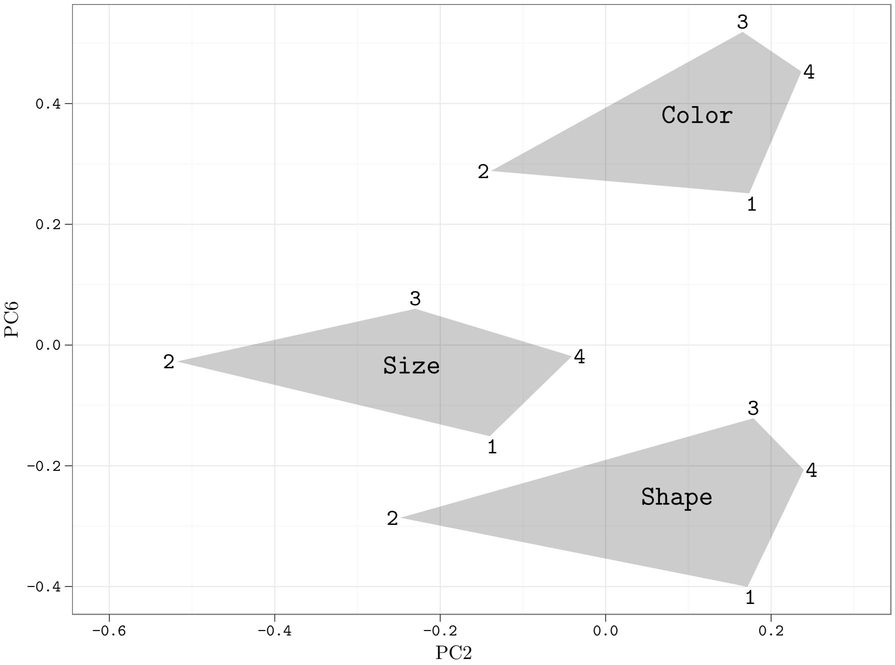


Figure 4.13: A PCA-space depiction of the integration layer's representational clusters (highlighted in gray polygons) of the predicate types `Color`, `Size`, and `Shape`. The sub-structure of these clusters shows averages over predicates containing specific object identifiers. Again, the systematic variation is evident, where predicates involving identifier `1` are below the cluster average; `2` are to the left; `3` are above; and `4` are to the right. This is further evidence of combinatorial representations of predicates and object identifiers.

a specific object identifier. Again, one can locate a given identifier in each cluster in a systematic way: The average of identifier `1` is below the cluster average; `2` is to the left; `3` is above; and `4` is to the right.

These are important observations. They demonstrate that the network's learned representations for object identifier and predicate type are essentially independent. For example, if the network is attempting to create a representation of a `Color` predicate, it does not need to know which object identifier is involved in the predicate; instead, it can simply decrease the value of the representation along PC4 (as can be seen in Figure 4.12), as this transformation is guaranteed to produce a `Color` predicate no matter what the identifier is. Similarly, to produce a representation involving object identifier 2, one need only translate the representation in the negative direction along PC2 (as in Figure 4.13), regardless of which predicate is involved. Of course, translating the network's representation along a PC entails small changes to the activations of many or all of the involved neural units, such that it would not be obvious to an observer that such a distributed change was all part of a single symbolic operation.

This representational independence means that the distributed representations are compositional—though one would be hard-pressed to spot it without aids such as PCA. In this context, **compositionality** means that, given representations for a `block` and a `pyramid`, one can create representations for `blue block` and `blue pyramid` by applying the same blue-ifying transformation to each; additionally, one can apply the reverse transformation to a `blue cylinder` to yield a regular `cylinder`. Compositionality follows from the existence of independent structure in the representations and allows representations to be manipulated based on their structure. Compositionality is, of course, the key property of classical symbols that allows them to be manipulated combinatorially, giving rise to systematicity. The fact that the network demonstrates representational independence, and thus compositional distributed representations, explains its ability to perform systematically: It appears to have implemented, through learning, a latent combinatorial symbol system on top of its purely connectionist substrate.

4.4 Discussion

The evidence from the previous section suggests that the representations learned by the grounded-meaning model comprise an implementation of a latent combinatorial symbol system in a neural architecture that is not hard-wired or otherwise predisposed to the task. This type of representation naturally combines the advantages of cognitive and connectionist representations in several ways. It is compositional, thus easily giving rise to systematic behavior. It is fully distributed, lending the benefits of graceful degradation due to damage that neural networks exhibit. Perhaps most importantly, it emerges through learning as a consequence of systematicity in the environment.

This last point in particular requires elaboration. The assertion that the symbolic structure emerges, and is not included in the system, may at first seem to be subject to the same arguments advanced against Hadley and Hayward's claims about their model (1997). Their model's representation was such that the different roles and fillers were *a priori* delimited, and the output criterion was that the best match from each class was chosen. This resulted in a system that could only produce valid propositions as output. In this sense, the output representation already included the relevant symbol system. The grounded-meaning model's output representation is fundamentally different. While well-formed intention stream outputs, like Hadley and Hayward's, are propositional in nature, the grounded-meaning model's training regimen does not constrain the combinatorial possibilities, allowing the system to generate output sequences that do not form valid propositional representations at all. The system must learn which symbols interact combinatorially, which are mutually exclusive, and so on. Thus no symbol system can be said to be built in before training.

Additionally, both the Hadley and Hayward (1997) and Frank et al. (2009) models, and indeed a large number of past connectionist models involving language, built all the necessary words into the network as fundamental units. The approach used here differs by using unsegmented phoneme sequences to represent entire utterances, leaving the model to learn a proper segmentation that provides the best mapping to the regularities of the microworld. The model's phoneme-based input representation also leaves open the possibility of the model learning novel words without an experimenter modifying the network architecture.

Like the Frank et al. (2009) model, the LSTM network powering the grounded-meaning model is trained by the method of gradient descent, which utilizes back-propagated error signals throughout the neural network. Since there exists little neurobiological evidence for back-propagated error signals, one might argue that both models are unrealistic in this respect. However, there is evidence that the overall architecture of these models is not as far removed from biological plausibility as one might expect. Specifically, Xie and Seung (2003) have recently discovered that gradient descent by backpropagation is essentially a computationally expeditious implementation of contrastive Hebbian learning. Hebbian learning methods are widely considered to be neurobiologically plausible and have been widely used in connectionist modeling, including the models of Hadley and Hayward (1997) and Hadley and Cardei (1999). Contrastive Hebbian learning, specifically, is the main ingredient in the training regimen of more biologically oriented models such as Leabra (O'Reilly, 2001). The model's use of an LSTM network in particular confers a few more benefits in terms of biological feasibility. Since real neurons are stateful—that is, their current state is always dependent upon their previous states in addition to their current inputs—they bear a closer resemblance to LSTM's stateful memory cells than to the stateless neural units traditionally used in many connectionist models. Additionally, the multiplicative functions of gate units in LSTM have close neurobiological correlates, and similar mechanisms have been used in models of the prefrontal cortex and basal ganglia (O'Reilly and Frank, 2006).

The grounded-meaning model differs significantly from the Frank et al. (2009) model, particularly in that its output representations are directly interpretable predicates, whereas their model's situation vectors required additional processing to ascertain the belief state of their model. Despite this difference, the output representations in the Frank et al. model were, like those in the grounded-meaning model, designed to convey the systematicity present in the micro-world. Because of this, it seems likely that the Frank et al. model, and indeed any other connectionist model that manages anything resembling strong systematicity without hard-wired support for a combinatorial symbol system, may in fact be learning a latent symbol system representation similar to the one demonstrated by the grounded-meaning model. While Frank et al. may at first appear to directly deny that their network possesses combinatorial representations, they in fact only argue this for the situation vector representations of their model's output layer. In light of the evidence presented here, on the other hand, one might guess that their model's learned representations over the *hidden layer* may come to be compositional, and thus symbolic in the sense of Fodor and Pylyshyn (1988).

This prediction may be construed as a reinforcement of one of Fodor and Pylyshyn's most important points—namely, that connectionist networks, to be truly systematic, must implement a symbol system (1988). While it is unclear whether Fodor and Pylyshyn meant to include emergent latent symbol systems as a potential means of satisfying this requirement, their comment, viewed in this inclusive way, seems to be essentially accurate. That said, one can still disagree with Fodor and Pylyshyn on their corollary assertion that connectionist networks, as implementors of classical cognitive theories, have nothing new to provide to cognitive science. Smolensky (1988) argued convincingly that any approach must account for both the soft, statistical nature of some aspects of cognition as well as the seemingly hard, rule-driven nature of other parts. He suggested that the best way to do this was to start with the soft connectionist formalism and use that to build the hard symbolic framework. The model presented here works towards this goal, but in doing so, does not need to give up the inherent softness of its connectionist roots. Symbolic cognitive theories often entirely ignore the issues of how their symbolic representations are formed through learning, or how they break down under load or because of deterioration from age or injury. Connectionist approaches, many of which emphasize both learnability and deterioration, are poised to address these questions, and in doing so, provide valuable developmental insights to cognitive researchers.

While some (e.g., Smolensky, 1990; Plate, 1995) have already argued convincingly that a marriage between symbols and distributed representations is both mathematically possible and neurocognitively necessary, this work suggests a learnable route through which such a system may be realized. This frees cognitive researchers to pursue both symbolic and connectionist approaches to cognition, with each theory constraining the other. Thus, symbolic approaches will continue to provide an extremely useful level of description of cognitive phenomena, and connectionist approaches will provide a means of realizing the former through simple, general learning processes over distributed neural networks. Barring a fundamental shift in science's understanding of how the brain processes information, this unified view is a necessary step in relating the mind's structure to the brain's function.

Chapter 5

Question Answering and Language Production

5.1 Introduction

The grounded-meaning model from the previous chapter shows how a neural network can learn to ground linguistic knowledge in other senses—an essential ability underlying true understanding of any language. However, a complete language model has more goals to fulfill, such as formulating responses to heard language and then producing those responses. This chapter introduces two incremental descendants of the grounded-meaning model that possess each of these abilities, gained by training in the domain of question answering.

Most linguistic utterances can be roughly classified as requests for information or sources of information, although some are both or neither of these. In human discourse, it is often the case that speech acts of these two classes alternate, with one party requesting information and the other responding, followed by one of the parties expressing another request, and so on. There is a considerable literature on how one could construct a program capable of participating in the request/response paradigm. One of the most recent and well-known examples of such a question-answering system is IBM's Watson

(Ferrucci et al., 2010), which can often best experienced players in the quiz show *Jeopardy!*. While Watson and systems like it are undoubtedly impressive, their errors are often baffling and inscrutable to onlookers, suggesting that the strategies they employ differ greatly from those humans use.

While question-answering systems have been well-studied in natural language processing domains, little research has been done on how the question/answer style of interaction might influence the ways in which humans acquire language. In human language modeling, much interest has been paid to the study of language comprehension (the transformation of an auditory signal into a meaning) and of language production (the inverse problem of transforming a meaning into an auditory signal), but there is little human language modeling research that focuses specifically on learning to produce appropriate responses to questions. This is an interesting subject in light of the fact that, when listening to language, learners are constantly confronted with these request/response, question/answer pairs. Particularly interesting is the question of how the language faculties of a learner in this situation could be implemented solely by a complex neural network like the human brain.

This chapter investigates the extent to which a pure neural network model of a human learner can develop a grasp of a micro-language by listening to simple question/answer pairs (Monner and Reggia, in press b). The model is situated in a simulated micro-world along with two other speakers, referred to as Watson and Sherlock. Watson asks simple questions about the shared environment in a subset of English, and Sherlock responds to these questions with the information Watson seeks. The model's task is to learn to emulate Sherlock. To do this effectively, the model must listen to the speech sounds of Watson's questions and learn to segment them into morphemes, words, and phrases, and then interpret these components with respect to the common surroundings, thereby grounding them in visual experience. The model must then recognize what information Watson is asking for and provide that information in a form that aligns with an answer that Sherlock would give.

This chapter examines two related models that differ in how the answers are provided. The first model learns to provide the answer to a question from Watson as a raw meaning—a series of predicates describing properties of, and relations between, objects in the micro-world. This style of response naturally resembles that of the grounded-meaning model in Chapter 4, since grounded understanding of the question is a clear requirement of finding the answer. As such, it is called the **meaning-answer model**. The answers this model provides are meant to be analogous to the learner's internal representations of meaning, though the representational form the model uses is determined *a priori* instead of learned. Teaching a model to answer this way is useful because it demonstrates explicitly that such a model can ground its answers by referring to concrete objects in the micro-world, rather than simply rearranging the question and guessing a plausible answer. However, for this to be a reasonable model of human language learning, it would need to learn entirely based on data that are readily available to any language learner. Thus, the model would need to have direct access to Sherlock's internal meaning representations, which is, of course, not generally possible in human language learning situations (though there is some evidence that the listener may often be able to infer meanings from context; see Tomasello, 2003). However, examining this limited model can still be useful, as its predicate-based outputs provide direct evidence that neural models can learn to produce a fully grounded representation of an answer to a question.

A second model addresses this limitation of the meaning-answer model by providing its answers much like the input questions—as sequences of speech sounds. This second model is termed the **spoken-answer model**. The representation of an answer in this case is unambiguously observable whenever Sherlock responds to Watson, placing the spoken-answer model a step closer to the reality of human language learning. The problem of learning to answer questions is more difficult in this case, since the network is trained using only the observable linguistic stimuli produced by Sherlock, which are not only more verbose than Sherlock's intended meanings but also potentially lossy translations thereof. Nonetheless, analysis of this model provides evidence that this approach to training offers a tractable path to both language comprehension and production.

5.2 Background

Previous efforts at neurocognitive modeling involving question answering have focused primarily on story understanding. Diederich and Long (1991) presented a hybrid connectionist model that provides plausible answers to open-class questions ("how," "why," "when," and so on) about a particular story. The model was a connectionist adaptation of a symbolic question-answering system called QUEST (Graesser and Franklin, 1990) and came equipped with pre-made spreading-activation modules containing both general world knowledge and knowledge about the story in question. Whereas this model used a vast database of general knowledge to answer open-class questions, the models in this chapter are concerned with learning to answer simpler, closed-class questions from first principles—that is, without building any knowledge into the system.

Miikkulainen (1993, 1998) presented DISCERN, a cognitively inspired connectionist model of story understanding that, unlike the Diederich and Long (1991) model, learned by conventional incremental training methods, such as Hebbian learning and gradient descent. Comprised of several distinct neural modules, DISCERN learned to process textual stories as familiar scripts, assigning the roles in these scripts to the relevant elements in the current story. In addition to story-paraphrasing capabilities, DISCERN included a question-answering subsystem that would receive a simple textual question word by word and process it into a case-role representation, generally with one unbound role representing the information the questioner sought. The model matched this representation against known stories in episodic memory to fill the vacant role and generate a textual answer to the question. This system differs from the models presented here in several ways. While DISCERN operated on text resolved down to the word level, the models in this chapter are concerned with speech resolved down to the phoneme level, and as such, with learning to distinguish words as well as interpret sentences. Where the makers of DISCERN built much functional modularity into the network, the models presented here develop such modularity via training, as their design provides no *a priori* functions to the layers. Finally, where DISCERN answers questions based on memory of a particular story, the models in this chapter are concerned with answering questions grounded in the immediate context provided by a visual microworld. The related GLIDES model (Williams and Miikkulainen, 2006) dealt with language grounding in a similar way but was concerned with free-form descriptions of the world, rather than answering specific questions about it.

Besides direct questions and answers, one can consider other types of request/response paradigms for training a neural network in language comprehension and production. Markert et al. (2009) trained a hybrid model, composed of nonconnectionist sensory analysis systems in addition to neural networks, to recognize simple spoken commands and use them to direct a robot in a motor task. The robot was able to recognize visual objects, associate them with words gleaned from the speech signal, and interpret commands to perform actions such as moving an object. This model is a convincing demonstration of the applicability of neural modeling methods to the task of integrating disparate input sources and learning to respond to a verbal request. It was not, however, designed as a model of a human learner, as evidenced by its non-neural preprocessing modules and built-in knowledge of motor plans and sentence structures.

While not fitting neatly into the question-answering paradigm, the model of sentence comprehension by St. John and McClelland (1990) serves as inspiration for this work. This model used simple recurrent networks (Jordan, 1986; Elman, 1990) to process a temporal sequence of words into a spatial "sentence gestalt" representation that was refined by training the network to complete role/filler pairs describing the sentence. Other models (e.g., Rohde, 2002) have since used this technique to great effect. The models presented in this chapter develop similar gestalt representations of both the visual scene and the auditory sentence before combining them into a grounded whole that can be used to produce answers.

The spoken-answer model in particular takes inspiration from a number of other language production models. Dell (1993) advanced a simple recurrent network model for sequentially producing the speech sounds of words as collections of phonological features. The models presented here reuse this notion of representing phonemes as bundles of features and extend Dell's model by producing speech sequences from learned internal semantic representations of sentences instead of hand-designed lexical representations.

Plaut and Kello (1999) presented a connectionist model of word comprehension and production that learned to associate 400 words with random unique "semantic" representations. An interesting contribution of this model was the inclusion of a forward model from articulatory features—representing the vocal tract movements necessary to produce speech sounds—to acoustic features. This kind of forward model enables a learner to receive feedback on appropriate productions by listening to speech—its own or that of others—and propagating mismatched acoustic expectations backward to modify articulatory representations. Rather than re-implement their forward model, the models in this chapter presume, for computational expediency, that such a model exists and that it can translate the available acoustic feedback into articulatory feedback.

Another connectionist model of comprehension and production advanced by Chang et al. (2006) learned to produce word sequences from a prespecified message. The main mechanism behind this learning was listening and imitation—listening to the utterances of others and actively predicting upcoming words, using previous words and inferred partial meanings as clues. The models presented here take this same approach of learning by listening and imitation, while improving the potential scalability of the model by not relying on singleton word representations (which effectively place hard limits on the learnable vocabulary) and, in the spoken-answer model, completely foregoing hand-coded representations of structured meaning (which place similar limits on the set of learnable concepts) in favor of implicit, learned meaning representations.

5.3 Methods

Both the meaning-answer model and the spoken-answer model have the same general structure, shown in Figure 5.1. Both models receive visual input from the environment via a sequence of temporally presented predicates describing objects and relations between them. The models also hear each question from Watson as auditory input consisting of a temporal sequence of phonemes. The outputs of both models mimic Sherlock's answers, but they do so at different levels: The meaning-answer model produces an answer to Watson's question as a sequence of grounded predicates, much like the grounded-meaning model in Chapter 4, whereas the spoken-answer model gives each answer as a sequence of speech sounds. Both models are pure neural networks, though the structure of these networks differs slightly to accommodate the differences between their tasks.

The remainder of this section explores in detail the tasks performed by the models and the representations for their input and output signals before comparing and contrasting the neural network architectures of each.

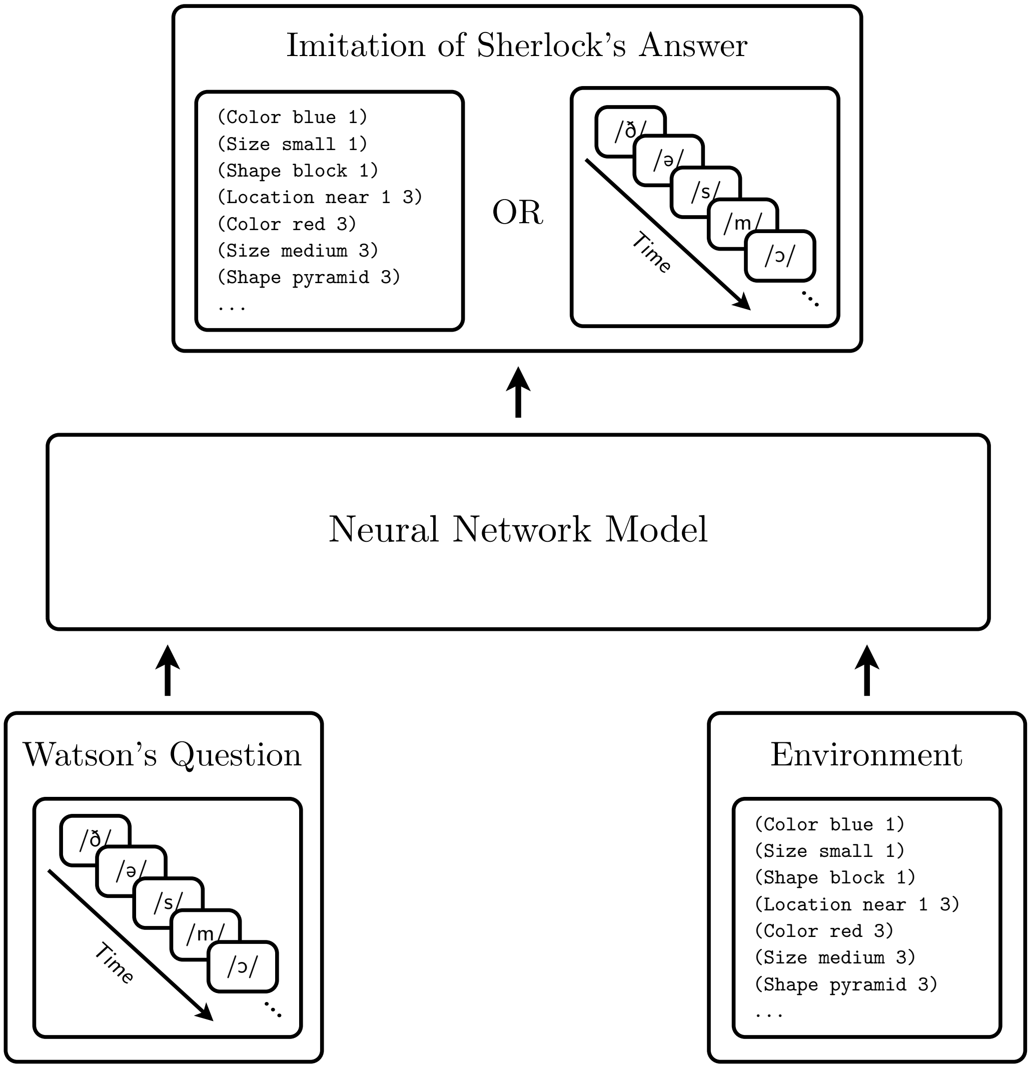


Figure 5.1: A high-level overview of the flow of information in the models.

5.3.1 Tasks

As mentioned in Section 5.1, the models are faced with the task of learning to comprehend and produce language in the context of answering questions. The overview of the task is simple: One of the models is placed in a shared micro-world environment with two other speakers, Watson and Sherlock. A training trial begins with Watson asking a question, the answer to which is directly observable in the environment. Sherlock surveys his surroundings and gives the answer, which the model observes and attempts to mimic. A testing trial, in contrast, is one in which Watson asks a similar question, but Sherlock is silent, relying on the model to produce the desired answer.

During a trial, the model first takes stock of the micro-world environment, receiving a stream of visual input and compressing it into a sort of visual gestalt representation for later reference. It then listens for Watson's question as a temporal sequence of speech sounds, processing these into an auditory gestalt of sorts. After hearing the totality of the question, the model attempts to combine the visual and auditory gestalts, internally mapping the references in the sentence to objects in the environment and subsequently figuring out what it is that Watson wants to know.

If the model in question is the meaning-answer model, it then provides the answer in the form of a sequence of grounded predicates that could serve as the meaning for a complete-sentence answer to Watson's question. The meaning-answer model must learn this behavior by imitating Sherlock, who always knows the answers to Watson's questions. In real human language learning, Sherlock's raw meaning would not necessarily be available for the model to imitate. The motivation for the meaning-answer model is to transparently demonstrate that a neural network model is capable of learning to produce this type of precise answer by grounding its auditory input in the visual scene.

In contrast to the meaning-answer model, the spoken-answer model answers questions by generating sequences of speech sounds that together form a complete sentence. The model again learns to produce these sounds by imitating Sherlock, and since these sounds are observable, this addresses the concern with the meaning-answer model and positions the spoken-answer model much closer to the reality of a human language learner. Because the model's internal meaning representations are learned, they are also much harder to interpret than the meaning-answer model's hand-designed predicates, and as such, it is more difficult to demonstrate robust grounding in the spoken-answer model than in the meaning-answer model. However, the spoken-answer model makes a different point: Mimicking ungrounded speech, which is generally a lossy translation of the internal meaning of the speaker, is sufficient to learn question-answering behavior.

The following subsections examine in detail the ways in which the auditory inputs, visual inputs, and both predicate- and speech-based model outputs are constructed, and also describe the vocabularies and grammars to which Watson and Sherlock restrict themselves.

5.3.2 Environment input

The micro-world environment shared by the three participants consists of a number of objects placed in relation to each other; an example environment configuration is depicted in Figure 5.2. Each object has values for the three attributes (`Size`, `Shape`, and `Color`), and each attribute has three possible values: `small`, `medium`, and `large` for `Size`; `block`, `cylinder`, and `pyramid` for `Shape`; and `red`, `green`, and `blue` for `Color`. Thus, 27 distinct objects are possible. In addition, each object has a number that identifies it uniquely in the environment, which is a useful handle for a specific object and is necessary in the event that the participants need to distinguish between two otherwise identical objects.

Each object is represented as three predicates, each of which binds an attribute value to an object identifier. For example, a small red block with unique identifier 2 is completely described by the predicates `(Size small 2)`, `(Color red 2)`, and `(Shape block 2)`, which are presented to the models in a temporal sequence. An environment consists of two to four randomly generated objects, and the predicates describing all of these objects are presented at the visual input layer of the model as a temporal sequence at the start of a trial. Each predicate is presented as a set of input unit activations.

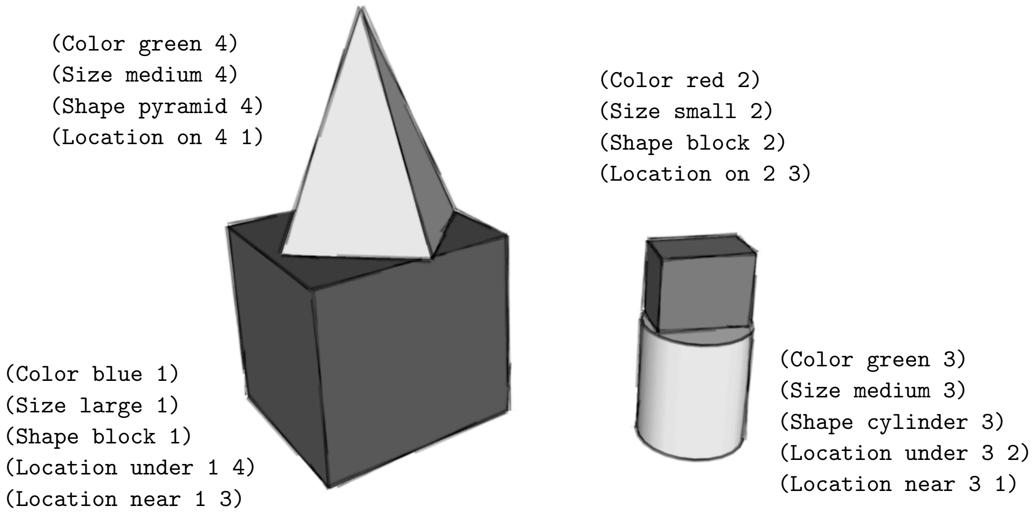


Figure 5.2: An example of a micro-world environment with four objects, along with the complete set of predicates that describe the environment. The predicates are the environment input to the model; the visual depiction of the objects here is simply for reference.

Input units are generally inactive, except for single active units corresponding to the type of attribute (e.g., `Color`), the value of that attribute (e.g., `blue`), and the unique identifier (e.g., `3`). See Figure 5.3 for a depiction of example predicates and their representations in terms of neural activity.

Additional predicates are used to describe spatial relations between the objects. One object may be near, on, or underneath another. For example, if the small red block (with identifier 2) is on top of a medium-sized green cylinder (identifier 3), that fact would be presented to the model as the predicate `(Location on 2 3)`. In the micro-world, the `on` and `under` relations are complementary (meaning that `(Location on 2 3)` implies `(Location under 3 2)`), and the `near` relation is symmetric (such that `(Location near 1 3)` implies `(Location near 3 1)`). The location predicates are presented along with the attribute predicates and at the same visual input layer.

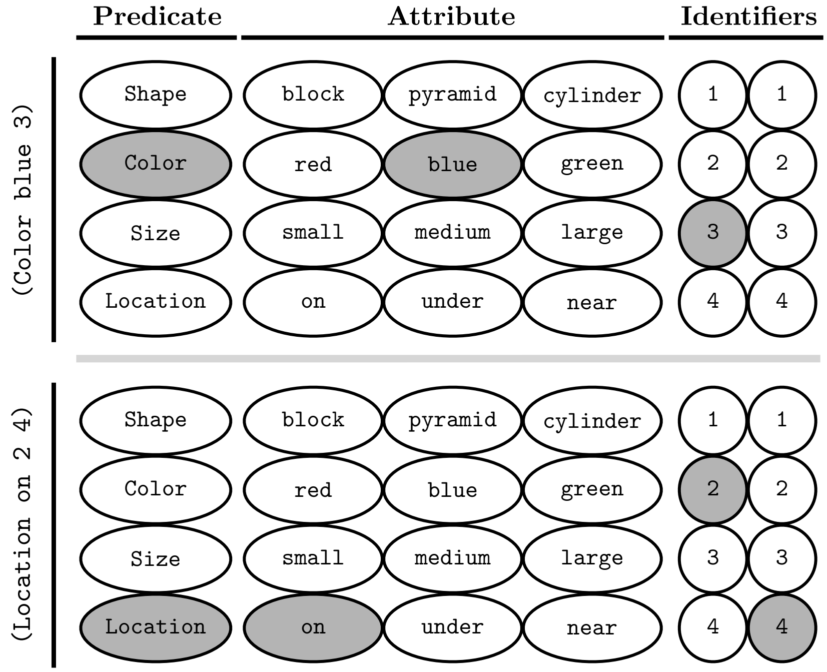


Figure 5.3: Neural representations of two example predicates used as visual input and meaning output. The left-most group of 4 cells in each case denotes the predicate type, of which only one cell will be active in a well-formed predicate. Cells in the middle group of 12 each stand for an attribute value; again, in well-formed predicates, only one of these is active at a time, and the attribute value will correspond to the correct predicate type. The right-most cells correspond to unique identifiers for environmental objects; in most cases, only an identifier from the first column is active, binding the active predicate-attribute pair to that identifier, as in the first example above representing `(Color blue 3)`. Location predicates, such as the second example `(Location on 2 4)`, activate two identifier nodes to establish a relationship between them.

Though this space of possible environments may seem small at first, the number of unique environmental configurations is quite large. Using only two, three, or four objects at a time provides approximately 2.48×10^{10} distinct possible micro-world configurations.[1]

The visual representation just described is a drastic simplification of real visual input that the models adopt for reasons of computational tractability. At the cost of moving away from sensory-level visual input, the models gain enough computational efficiency to study the language acquisition phenomenon of primary interest. This type of high-level visual representation can be viewed as the equivalent of a representation that might be produced by the later stages of the human visual system, though probably not in this precise form.

This environment representation differs from that used in Chapter 4, breaking each object up into single-attribute predicates. This ends up requiring more work from the model, which must now learn to bind together many attributes—based on the object identifier—to create a unified representation of each object, taking care not to confuse objects with similar attributes. Since each object is no longer presented to the network as a single coherent pattern, this representation is in some sense lower level than that utilized

[1] This is calculated as $\sum_{n=2}^{4} 27^n \; 2^{\binom{n}{2}} \; 3^{\binom{n}{2}}$ where n is the number of objects in an environment and there are 27 featurally distinct objects. The first term denotes the number of distinct combinations of objects possible. The second term denotes the independent presence or absence of `near` predicates for all combinations of two objects, counted as unordered because near is symmetric. The third term captures the three possibilities for `on` and `under` predicates for each combination of two objects—neither of these predicates exists, or both exist with two possible object orderings.

in Chapter 4, bringing the model one step—albeit a small one—closer to sensory-level (i.e., retinotopic) visual input. The other major deviation from the Chapter 4 visual representation is the inclusion of additional predicates describing the spatial relations between the objects in the environment. This addition allows answers to be conditioned on the locations of objects, so the models can now distinguish, for example, two otherwise identical small red blocks by the fact that one of them rests on top of a cylinder and the other does not.

5.3.3 Question input

Watson produces complete English sentences that ask questions about the current shared environment. There are many possible questions for each environment; for the example environment in Figure 5.2, Watson might ask: *what color block is the green pyramid on?*, *what thing is under the small block?*, *what color are the medium things?*, or *where is the pyramid?*.

At the start of each trial, Watson examines the environment and then begins deriving a question beginning at the **Question** nonterminal in the mildly context-sensitive grammar of Figure 5.4. The derivation is constrained not only by the grammar itself, but also by the environment. Specifically, any objects that Watson refers to must be present in the environment and, to a sophisticated observer, unambiguously identified by the full context of the question. For example, Watson could not ask *what color is the block?*

because it is not clear which block is being referred to, and thus any answer would be poorly defined. Watson can, however, ask questions about groups of objects that share a property, such as *what color are the medium things?*; in this case, the medium things are the cylinder and pyramid, which are both green, so the answer is well defined. Questions posed to the models are required to have well-defined answers to maintain the simplicity of evaluation and training; after all, if the answer is ambiguous, how can one tell whether the model produced the correct answer or not? An important future research task will be to relax this requirement and see if one can train a model such that, when it is given an ambiguous question, it produces either an answer that is plausible, or an answer indicating its uncertainty.

The words in Watson's question are phonetically transcribed, and the resulting phoneme sequences are appended to produce one uninterrupted temporal sequence corresponding to the spoken sentence. Word and morpheme boundaries are not marked, leaving the model to discover those on its own, just as with real-world speech signals. When the speech sequence for a question is presented as input to the model, individual phonemes are given one at a time. Phonemes are represented using the same bundles of binary acoustic features used in Chapter 4. These features include such cues as voicing, affrication, and formant frequency categories. A full listing of phonemes and their features can be found in Table 4.1.

Question	→	where **Is Object** | **What Is Property**
Answer	→	**Object Is Property**
Object	→	the [**Size**] [**Color**] **Shape**
What	→	what color [**Size**] **Shape** | what size [**Color**] **Shape** | what [**Size**] [**Color**] thing
Property[1]	→	**Location** | **Color** | **Size**
Location[2]	→	on **Object** | under **Object** | near **Object**
Is[3]	→	is | are
Size	→	small | medium | large
Color	→	red | blue | green
Shape	→	things | pyramid[s] | block[s] | cylinder[s]

Figure 5.4: The mildly context-sensitive grammar used to train the meaning-answer model on the question-answering task. Terminals begin with a lowercase letter, while nonterminals are in boldface and begin with an uppercase letter. The symbol | separates alternative derivations, and terms in brackets are optional. Watson begins derivations from the **Question** nonterminal to generate a question that can be given as input to the model, and Sherlock begins from the **Answer** nonterminal to generate an answer to such a question.

[1] When the parent nonterminal is **Question**, the evaluation chosen for **Property** is constrained so as not to reveal the missing property from the preceding **What** expansion. For example, questions such as *what color pyramid is red?* are disallowed.

[2] Pyramid objects are disallowed as subjects of *under* and objects of *on* because, in the micro-world, pyramids cannot support other objects due to their pointed tops.

[3] The evaluation chosen for **Is** must agree in number with the **Object** expansion that serves as its subject.

5.3.4 Answer output

After Watson has finished asking a question, the model attempts a response. On training trials, Sherlock then gives the correct response, and the model adjusts its connection weights to better imitate this response in the future. Training is supervised in the sense that complete training targets are always provided, but it is a type of self-supervision where the model need only observe the naturally occurring dialog between Watson and Sherlock and attempt—perhaps covertly—to imitate the latter.

To answer Watson's question, the model must examine the environment and determine the set of object attributes and relations Watson is referring to. If the question was *what color block is the green pyramid on?*, in the context of the environment of Figure 5.2, the model must first determine that `(Color green 4)` and `(Shape pyramid 4)` are of interest, and then find that the on-relation only makes sense with object number 1, as `(Location on 4 1)`. `(Shape block 1)` shows that object number 1 fits with the clue that Watson has a block in mind. Finally, the model must deduce that Watson asked about the color, and so must retrieve the predicate `(Color blue 1)` from its working memory representation of the environment.

When training the meaning-answer model, Sherlock gives precisely these five predicates as its answer, and the meaning-answer model must learn to do the same. On the other hand, when training the spoken-answer model, Sherlock gives the answer as a speech sequence. Specifically, Sherlock derives a complete English sentence, starting from the **Answer** nonterminal in the grammar of Figure 5.4. Sherlock constrains the derivation to conform to the contents of the predicate-based answer. For the question *what color block is the green pyramid on?* and the environment shown in Figure 5.2, Sherlock will invariably answer *the green pyramid is on the blue block*. The answer sentence is phonetically transcribed, just as Watson's questions are, to form an unsegmented temporal sequence of phonemes. The spoken-answer model uses Sherlock's speech stream as a temporal sequence of training targets to be produced in response to the question input.

Previous models (e.g., Plaut and Kello, 1999) have proposed that a learner run its internal representation of the answer's meaning forward to create a series of articulations for the sounds of the sentence. By feeding these representations through a forward model mapping articulatory features to auditory features, the learner could generate predictions about the speech stream. This would enable the learner to compare Sherlock's speech stream with its own predictions, working backward to turn auditory prediction errors into a training signal for articulatory features, and thus learning how to produce the desired speech. The spoken-answer model, for computational expediency, assumes that this process has already taken place, training directly on phonemes as bundles of binary articulatory features. The complete list of binary articulatory features and associated phonemes can be found in Table 5.1.

Table 5.1: Binary Articulatory Features of Spoken Phonemes

	Phoneme																																			
Feature	b	d	e	f	g	h	i	k	l	m	n	o	p	s	t	u	v	v	w	z	æ	ð	ɳ	ɑ	ɔ	ə	ɚ	ɛ	ɪ	ɹ	ʃ	ʊ	ʌ	ʒ	ʤ	ʧ	θ
Consonantal	+	+	−	+	+	+	−	+	+	+	+	−	+	+	+	−	+	+	+	+	−	+	+	−	−	−	−	−	−	+	+	−	−	+	+	+	+
Vocalic	−	−	+	−	−	−	+	−	+	−	−	+	−	−	−	+	−	−	−	−	+	−	−	+	+	+	+	+	+	+	−	+	+	−	−	−	−
Anterior	+	+	−	+	−	−	−	−	+	+	+	−	+	+	+	−	−	+	−	+	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+
Coronal	−	+	−	−	−	−	−	−	+	−	+	−	−	+	+	−	−	−	−	+	−	+	−	−	−	−	−	−	−	+	+	−	−	+	+	+	+
+Voicing	+	+	+	−	+	−	+	−	+	+	+	+	−	−	−	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	−	+	+	+	+	−	−
−Voicing	−	−	−	+	−	+	−	+	−	−	−	−	+	+	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	+	+
Continuant	−	−	+	+	−	+	+	−	+	−	−	+	−	+	−	+	+	+	+	+	+	+	−	+	+	+	+	+	+	+	+	+	+	+	−	−	+
Stop	+	+	−	−	+	−	−	+	−	+	+	−	+	−	+	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	+	+	−
Nasal	−	−	−	−	−	−	−	−	−	+	+	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−
Strident	−	−	−	+	−	−	−	−	−	−	−	−	−	+	−	−	−	+	−	+	−	−	−	−	−	−	−	−	−	−	+	−	−	+	+	+	−
Very High	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
High	−	−	−	−	+	−	−	+	−	−	−	−	−	−	−	−	+	−	+	−	−	−	+	−	−	−	−	−	+	−	+	+	−	+	+	+	−
Middle	+	+	+	+	−	−	−	−	+	+	+	+	+	+	+	−	−	+	−	+	−	+	−	−	−	+	+	−	−	+	−	−	−	−	−	−	+
Low	−	−	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	+	−	−	−	−	+	−	−	−	−
Very Low	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−
Front	−	−	+	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	+	+	−	−	−	−	−	−	−	−
Front Center	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	+	−	−	−	−	−	−	−	−	−	−
Center	+	+	−	+	−	+	−	−	+	+	+	−	+	+	+	−	+	+	−	+	−	+	−	−	−	−	+	−	−	+	+	−	−	+	+	+	+
Back Center	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−
Back	−	−	−	−	+	−	−	+	−	−	−	+	−	−	−	+	−	−	+	−	−	−	+	+	+	−	−	−	−	−	−	+	−	−	−	−	−

In both the predicate-output and speech-output scenarios, the answers to the questions differ, based on the micro-world environment, such that the mapping from questions to answers is never one-to-one. Indeed, both predicate-based and speech-based answers to Watson's question *what color block is the green pyramid on?* would change if the block under the green pyramid were red instead of blue; the predicate-based answer would also change (albeit slightly) if the scene were visually identical but the object numbering was altered. Though this example suggests that the number of possible answer variations is small, some questions admit much greater environment-based variation than others. Questions that are more open-ended, such as *where is the block?* could have literally hundreds of possible answers stemming from the diversity of environments where the question makes sense. Thus, neither the model nor Sherlock could answer questions reliably without integrating information from the visual environment.

5.3.5 Neural architectures

To learn the tasks described in the previous section, the neural networks that comprise both the meaning-answer model and the spoken-answer model need to accurately recognize and produce sequences that are as long as 20 predicates or 40 phonemes. They are built using the long short-term memory (LSTM; Hochreiter and Schmidhuber, 1997; Gers and Cummins, 2000; Gers and Schmidhuber, 2001) architecture, which has been previously shown to perform very well on long temporal sequences, and trained using the generalized long short-term memory (LSTM-g) training algorithm described in Chapter 2.

Although the basic components of the two models are the same, the network architectures differ slightly, based on the demands of each task. The network architecture of the meaning-answer model is shown in Figure 5.5. The network has an input layer for visual predicates (bottom right), which are presented temporally and accumulated into the visual gestalt in a visual accumulation layer. Internal layers such as this one are composed of the self-recurrent LSTM memory blocks described in Chapter 2 and are identified by the small recurrent arc on the top-left side. The network also has an

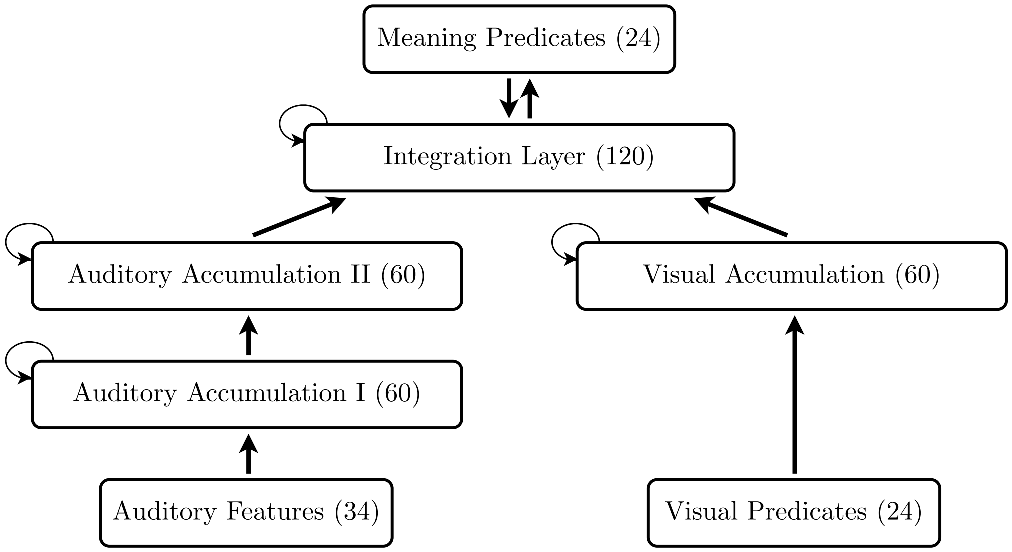


Figure 5.5: The neural network architecture of the meaning-answer model, which is closely related to that of the grounded-meaning model (see Figure 4.6 for comparison and explanation).

input layer for auditory features, which feed through two successive layers of memory cells, forming an auditory gestalt on the second such layer. The two pathways differ in serial length because pilot experiments showed that the visual and auditory gestalt representations are best formed with one and two layers of intermediary processing, respectively. After both input gestalts have formed, they are integrated by another layer of memory cells, which is then used to sequentially generate predicates that specify the grounded output that answers the input question. The network's previous output is fed back to the integration layer in the style of a simple recurrent network (SRN; Jordan, 1986) to provide a more specific context for sequential processing.

The network that implements the spoken-answer model, detailed in Figure 5.6, differs from that of the meaning-answer model in only a few key respects. The main change is that the meaning-answer model's output layer, which used to specify predicates, now consists of articulatory features used to create output speech sequences. The other change is the addition of another layer of memory cells between the integration layer and the output. Pilot experiments showed that network architectures lacking this additional layer had more trouble converting the intended speech stream into articulatory features. A key feature of the spoken-answer model architecture is that, unlike the meaning-answer model, it does not prespecify any semantic representations. This forces the network to learn its own internal meaning representations, which have the benefit of being distributed while supporting the strongly systematic generalization that language use requires (Fodor and Pylyshyn, 1988; Hadley, 1994).

5.4 Results

5.4.1 The meaning-answer model

Ten independent instances of the meaning-answer model, each with randomly chosen initial connection weights, train on the question-answering task for 5 million randomly generated trials. This duration of training may seem long, but it represents a mere fraction of a percent of the input space, leaving the model to generalize across the rest. An arrow between a pair of layers in Figure 5.5 indicates that a given pair of units from these layers possesses a trainable weighted connection with a probability of 0.7. The networks have 300 memory cells in all internal layers combined and about 60

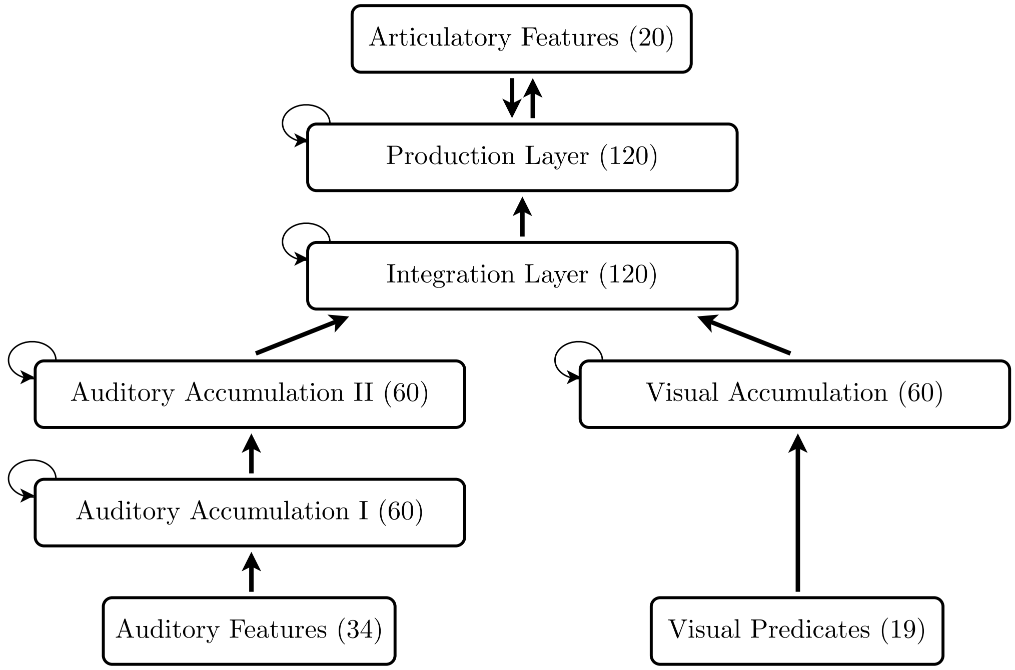


Figure 5.6: The neural network architecture of the spoken-answer model, depicted in the style described in Figure 4.6. This architecture has an additional production layer that helps the network learn to generate the output phonemes from the internal meaning representations of the integration layer. The number of units in the visual predicate input layer decreased from 24 to 19 to accommodate the slightly simplified grammar that the spoken-answer model learns (see Figure 5.11).

thousand trainable connection weights in total. The networks use a learning rate of 0.01.

Reported accuracy results are based only on trials that the model had never encountered before evaluation time, and as such, always reflect generalization and never memorization. For each such trial, the model must produce a sequence of predicates, where a predicate is counted as correct only if it occurs in the correct sequential position and has all unit activations on the correct side of 0.5. On average, the trained meaning-answer models were able to produce a perfect sequence of grounded predicates to answer a novel input question 92.5% of the time, strongly suggesting that this style of observe-and-imitate training is sufficient even for the difficult task of grounded question comprehension and answering.

To evaluate the relative difficulty of the components of the task, one can decompose the output predicates into three categories: attribute values corresponding to linguistic descriptions that are present in the question, referent identifications made by grounding the question in the environment, and attributes that are inferred from the environment to answer the question. Comparing the time-course of accuracy over these three categories, Figure 5.7 shows that the model learns to understand spoken descriptions fairly quickly, with accuracy leveling off above 95% after the first million trials. At this point, accuracy on referent identifications and question-answers is relatively weak, at 80% and 70%, respectively; however, accuracy on these

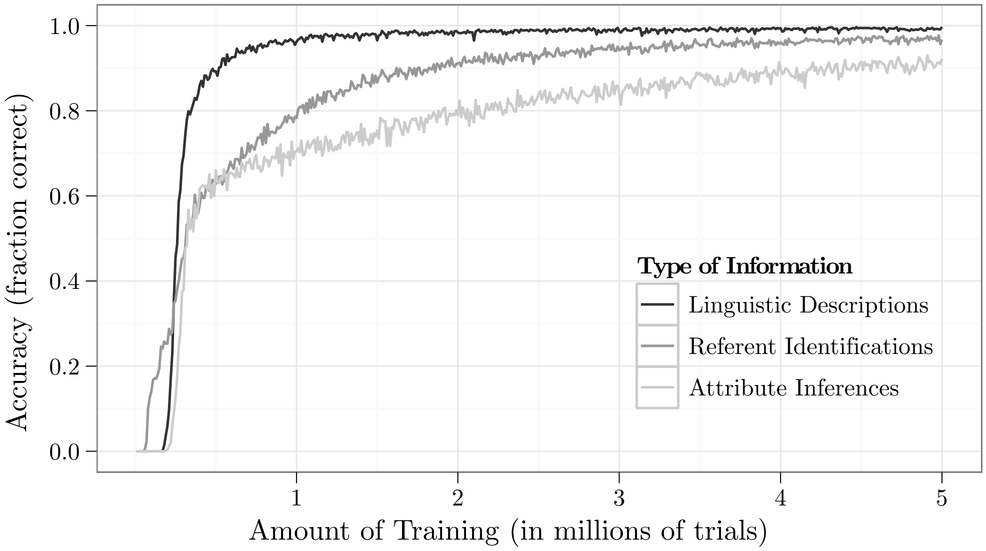


Figure 5.7: Accuracy versus training time for three separate components of the question-answering task, averaged over all ten training runs.

areas continues to slowly improve as training progresses, providing evidence that grounding the language in the environment is much more difficult than recognizing a sequence of words and the basic outline of the meaning it entails. It also shows that using the environment to infer the answers to questions is harder still.

To begin to gain an understanding of how the model acquires this systematic ability to answer novel questions in novel environments, one must examine the internal representations that the model learns during training. These are obtained from snapshots of one trained model's integration layer taken immediately after it finishes hearing Watson's question. For certain very simple questions, such as *what color is the block?*, there are enough instances in the test data such that each possible answer—in this case, *red*, *green*, or *blue*—is represented multiple times. Looking for systematic differences between related sets of questions—and between multiple instances of the same question where the different micro-world environments would suggest differing answers—reveals what the model knows at the instant immediately after it hears the complete question.

The representations of three related questions are examined first. Each question asks about the color of a different type of object—a *block*, a *pyramid*, or a *cylinder*—and can have any of the three possible colors as its answer, depending on the environment. Applying principal component analysis (PCA) to the representations aids in analysis by removing some of the redundancy and distributedness that is characteristic of learned representations in neural networks. While variation along the principal components (PCs) need not, by design, correspond to interpretable changes in the internal representation, this is in fact the case for many PCs. This indicates that the models are learning representations that can vary systematically in many dimensions. The following figures only involve those PCs for which the variation is easily interpretable. Though lower PC numbers represent larger amounts of variation in general, the PC number is essentially immaterial here, since the goal is merely to point out systematic variation in *some* PC.

Figure 5.8 graphs the representations in PCA-space, grouping them by question type, and subdividing those groups based on the expected answer. Here, each shaded polygon corresponds to a group of questions that are identical, though they may have different answers. For example, a polygon might represent the question *what color is the pyramid?*. Valid answers to this question involve a color attribute: `red`, `green`, or `blue`. Each vertex of such a polygon is labeled with the answer Watson is looking for and represents an average over all such questions that have this answer. So, to complete the example, the vertex labeled `blue` (the top-left-most vertex in Figure 5.8) represents the average representation over all instances where Watson asked *what color is the pyramid?* and had a blue pyramid in mind.

In this figure and those that follow, the representations for some of the shaded groups may seem to overlap, which one might expect to cause the model to mistake one group for another. These representations, however, are always well-separated by other principal components that are not shown, leaving the model with no such ambiguity.

The results show a remarkably systematic pattern. First, the model's internal representations differ consistently, based on the expected answer to the question, which was not present in the auditory input and has not yet been produced by the network as output predicates. This systematic difference implies that the model is aware of the expected answer to the question as soon as it is asked. While the model possessing the answer is not surprising, since all the information needed to deduce it is present in the visual input, the clarity of the answer's presence in the internal representation just after the question input suggests that the model processed the question online, deriving the answer immediately from its working-memory representation of

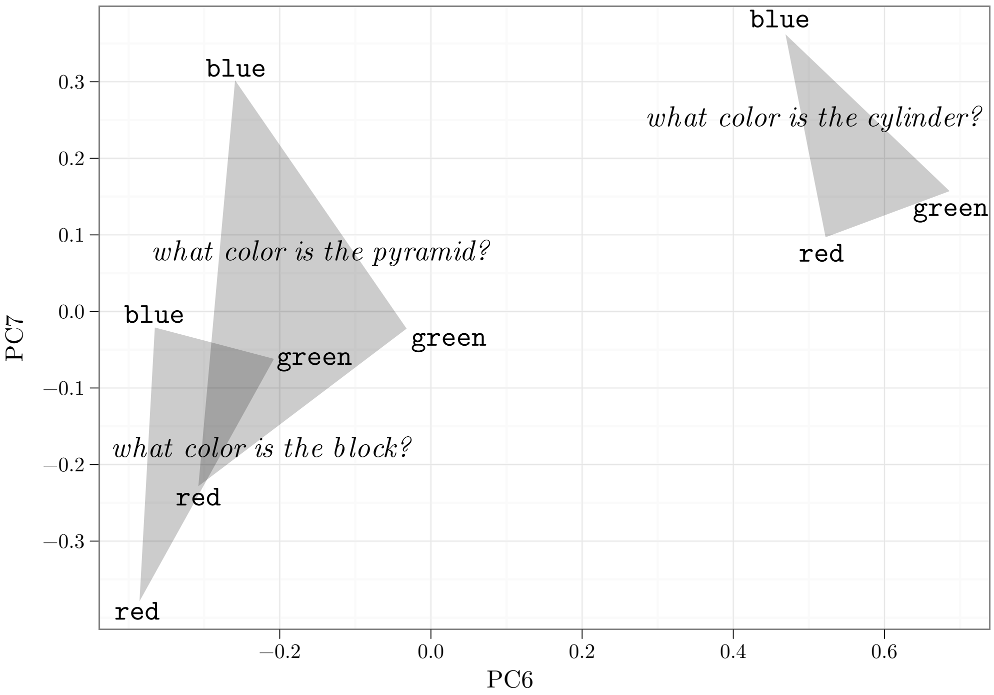


Figure 5.8: Learned internal representations generated by the model for a set of *what color?* questions, visualized in PCA-space and separated according to the expected answer to the question. Shaded polygons correspond to sets of questions that are identical but might nonetheless have different answers. Each possible answering attribute corresponds to a labeled vertex of the polygon.

the micro-world environment. Additional systematicity is apparent in the respective orientations of the answer-groups for each question. Regardless of which question was asked, a *blue* answer is always a positive shift along PC7 and a *red* answer is always a negative shift, with *green* answers falling in the middle and along a positive shift in PC6. This type of organization recalls the representations discovered in the model from Chapter 4, where they served as evidence of compositional, symbol-like representations for the various color concepts.

This analysis was repeated for a collection of similar *what size?* questions, with very similar results as shown in Figure 5.9. Again, the model displays clear representational differences that reveal its knowledge of the question's answer and further show it represening this information using systematic variations that are largely independent of the question being asked.

One might next inquire about the extent to which the model's internal representations reflect the environmental referents present in the question and answer. This can be gauged by analyzing the same set of *what color?* and *what size?* questions, again partitioning the trials into groups by question, but this time subdividing these groups based on the unique identifier that the micro-world assigned to the referent. These identifiers are never represented in auditory input. Thus, the network's reliable knowledge about identifying number is direct evidence that the model's internal sentence representations are grounded in the visual input. The results in Figure 5.10 show

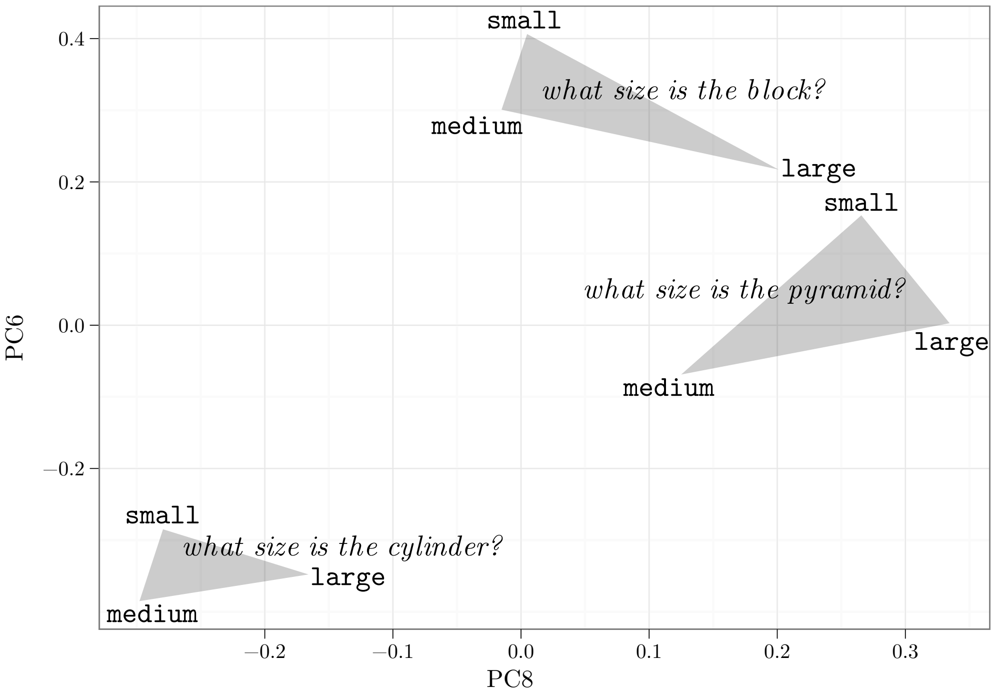


Figure 5.9: Learned internal representations generated by the model for a set of *what size?* questions, visualized in PCA-space and separated according to the expected answer to the question, in the style of Figure 5.8.

a remarkably systematic representation that makes clear several distinctions at once. First, the two question types *what color?* and *what size?* are separated along PC1. Second, the shape of the object in each question has an observable effect on its representation regardless of question type, with *pyramid* questions producing markedly smaller shaded regions in this projection than the other questions, while the *cylinder* and *block* questions produce regions of similar size, with the former shifted down along PC4 across both question types. Finally and most importantly, each question's representation reveals clear knowledge of the numerical identifier of the question's referent, with questions about objects 1 and 4 being distinguished along PC4, while questions about objects 2 and 3 differ along PC1. This figure definitively shows the systematic independence of the model's knowledge about the type of answer being sought (*size* or *color*), the shape of the object in question (*block*, *pyramid*, or *cylinder*), and the identifying number of the referent.

5.4.2 The spoken-answer model

Because of the additional size and complexity of the spoken-answer model, it trains on a slightly smaller grammar for computational expediency. This grammar is a subset of the one in Figure 5.4, arrived at by disallowing cylindrical objects and removing the size attribute altogether. For reference, the resulting grammar is shown in Figure 5.11. Pilot experiments indicate that the spoken-answer model has no trouble learning the full

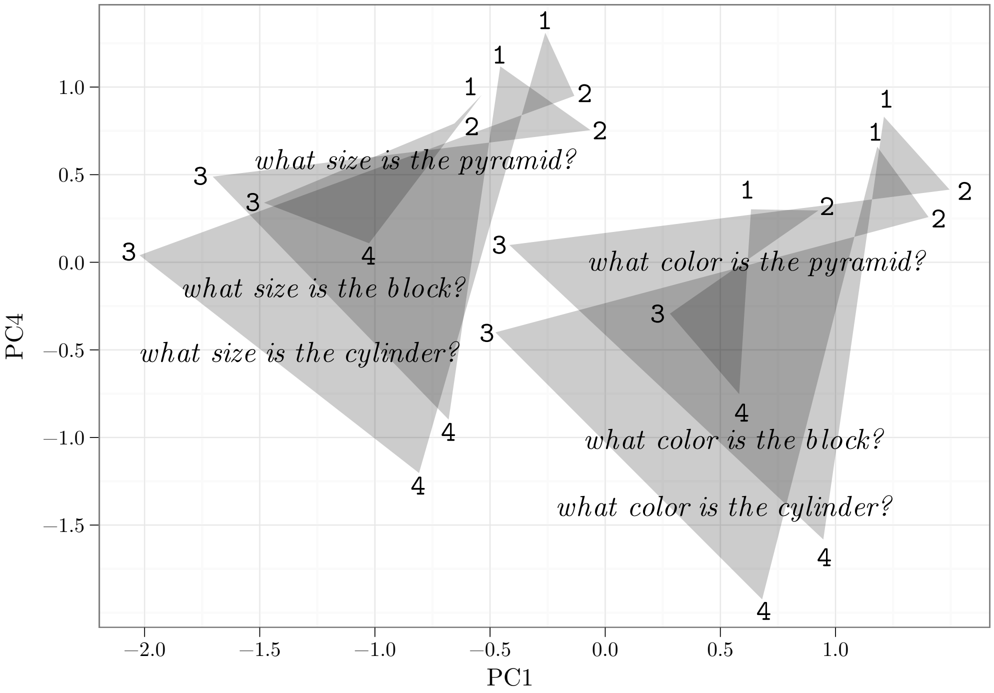


Figure 5.10: Learned internal representations generated by the model for a set of *what color?* and *what size?* questions, visualized in PCA-space and separated according to the object number of the sentence's referent, in the style of Figure 5.8.

Question	→	where **Is Object** | **What Is Property**
Answer	→	**Object Is Property**
Object	→	the [**Color**] **Shape**
What	→	what color **Shape** | what [**Color**] thing
Property[1]	→	**Location** | **Color**
Location[2]	→	on **Object** | under **Object** | near **Object**
Is[3]	→	is | are
Color	→	red | blue | green
Shape	→	things | pyramid[s] | block[s]

Figure 5.11: The slightly simpler grammar on which the spoken-answer model is trained. For comparison, and for the text of the footnotes, see the original grammar in Figure 5.4.

grammar, and scales from smaller grammars at approximately the same rate as the meaning-answer model. However, computational resource constraints impeded replicated experiments with the model at that size.

Ten individual instances of the spoken-answer model train for up to 5 million randomly generated trials to learn the question-answering task, using the grammar from Figure 5.11. The networks are connected as shown in Figure 5.6, with each pair of units in connecting layers having a probability of 0.7 of sharing a weighted connection. This probability, combined with the 420 total memory cells across all internal layers, results in networks with approximately 120 thousand connection weights. The learning rate is 0.002.

Trained models are assessed on their ability to produce a perfect sequence of phonemes comprising a full-sentence answer to an input question on a novel trial. Each phoneme in a sequence is considered correct if each feature unit has an activation on the correct side of 0.5. On average, the spoken-answer models are able to accomplish this for 96.9% of never-before-seen trials. For example, in a micro-world containing a blue block on top of a red block, as well as a red pyramid nearby, Watson asked the model *what red thing is the blue block on?*, to which the model responded with [ðəblʊblakɪzanðəɹɛdblak], which translates as *the blue block is on the red block.* This response was correct in the sense that the phoneme sequence the model chose represents the desired answer to the question; additionally, the model produced these phonemes in the correct sequence, and each had exactly the right articulatory features.

The occasional errors made by the networks fall into roughly three categories. The first and most common is a slight mispronunciation of an otherwise correct answer. For example, the model was asked *what color pyramid is on the red block?*, and it produced the answer [ðəbl?pɪɹəmədɪzanðəɹɛdblak], where the "?" indicates that the network produced a pattern of articulatory features that does not precisely correspond to one of the language's phonemes. However, it is clear from context that the model meant to say *the blue pyramid is on the red block*, which is the correct answer to the question in the environment provided during that trial. Despite the fact that the network clearly knew the correct answer and came extremely close to producing it, the reported statistics count every trial like this as an error.

The other type of error occurs in cases where the model seems unsure of the expected answer to the question. Sometimes, this takes the form of a direct and confident substitution of an incorrect answer, as was the case when the model was asked *what color is the block?* and confidently answered *the block is green* when the block was in fact blue. Other times, the model muddles two possible answers when speaking. For example, when asked *what color block is the blue pyramid under?*, the model responded with [ðəblʊpɹəmədɪzəndɚðəgl?blɑk], which glosses roughly as *the blue pyramid is under the glih block*. The correct answer for the malformed word here would have been *blue*, but the model was apparently confused by a preponderance of green objects present in the micro-world on that trial, producing this hybrid of the two words in its answer. In other instances, the model trails off or "mumbles" when it does not know which word to say in its answer. A trial where the model was asked *where is the blue pyramid?* provides an example of this behavior. The pyramid in question was on top of a green block, which required the model to produce three salient pieces of information that were not present in the question (i.e., *on*, *green*, and *block*) as part of its answer. The model came back with [ðəblʊpɹəmədɪzɑnðəgɹinb??], roughly *the blue pyramid is on the green buhhh...*. Though the model produced the first two components of the expected answer, it was apparently unsure enough about *block* that it trailed off into unrecognizable phonemes and, in fact, stopped producing phonemes short of where the utterance would normally end.

Looking at the spoken-answer model's learned distributed representations reveals the same sorts of patterns that were present in those of the meaning-answer model. Figures 5.12–5.14 examine the spoken-answer model's internal representations by analyzing snapshots of the integration layer activations immediately after a question is presented. PCA strips some of the redundancy out of the representations, identifying the main components for productive visualization, two at a time. This time, the investigation focuses on a more involved set of *where?* questions, which each require the network to produce three pieces of information that were not present in the question. In response to the example question *where is the red block?*, the model would need to respond by placing the red block in relation to a reference object, as in *the red block is under the blue pyramid.* Figures 5.12–5.14 test the internal representations for the presence of information about the location (*under*), color (*blue*), and shape (*pyramid*) of the reference object immediately after the model hears the question and before it begins its response.

Figure 5.12 shows a view of internal representations from PCs 2 and 3, depicting not only a clear separation of three variations of the *where?* question, but also systematic manipulations of PC3 to distinguish the *on* and *near* relationals, while PC2 separates these from *under*.

Figure 5.13 shows the color of the reference objects in PCs 4 and 8. While other PCs not depicted here demarcate representations of the different question types, this figure shows PC4 separating *red* from *green* and PC8 distinguishing *blue* from either of these.

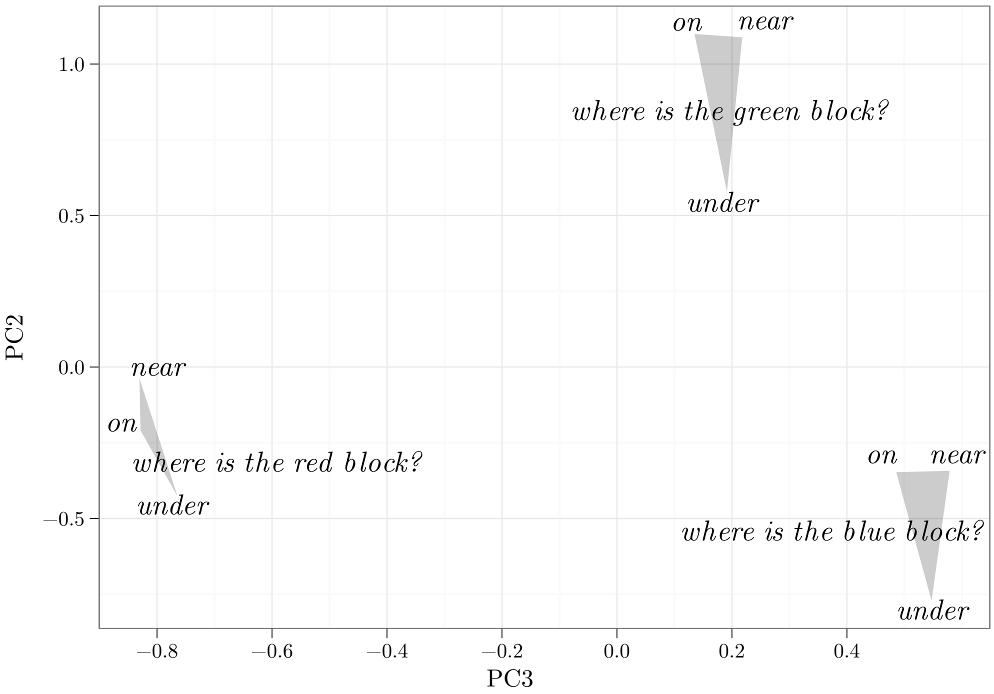


Figure 5.12: Learned internal representations generated by the spoken-answer model for a set of *where?* questions, visualized in PCA-space and separated according to the relational word that positions the reference object in relation to the question's subject. Shaded polygons correspond to sets of questions that are identical but might nonetheless have different answers. Each possible word representing a valid answer corresponds to a labeled vertex of the polygon.

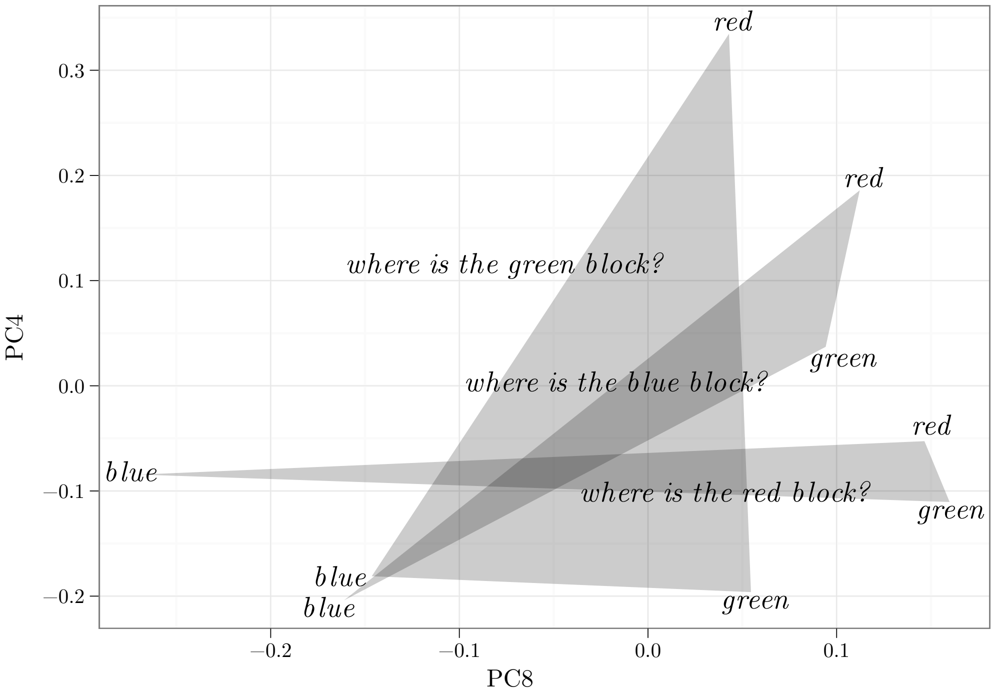


Figure 5.13: Learned internal representations generated by the model for a set of *where?* questions, visualized in PCA-space and separated according to the color of the reference object used to locate the question's subject, in the style of Figure 5.12.

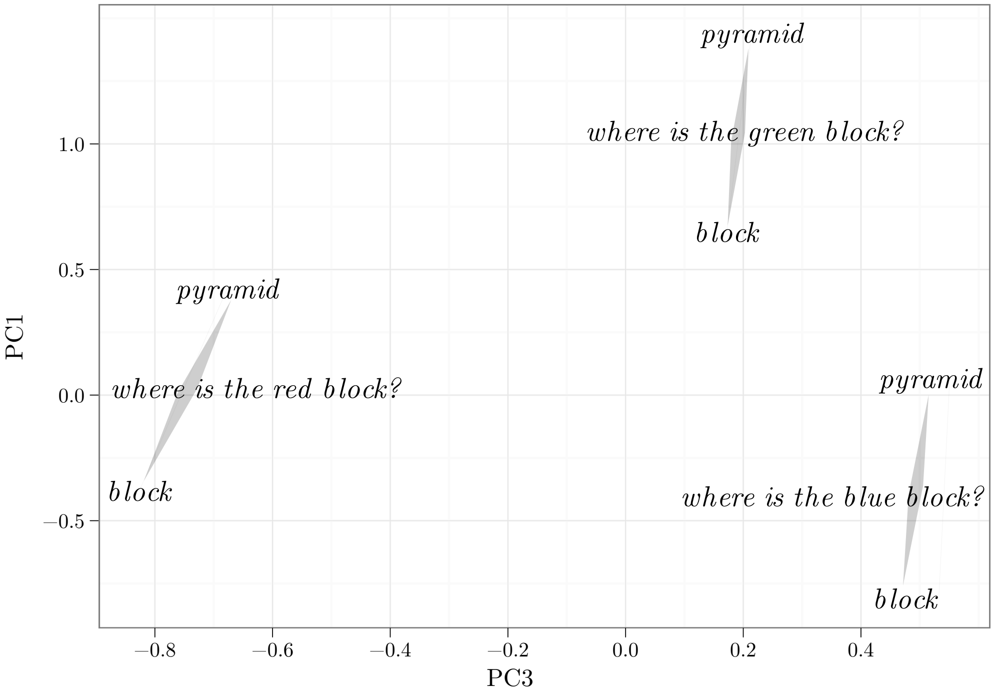


Figure 5.14: Learned internal representations generated by the model for a set of *where?* questions, visualized in PCA-space and separated according to the shape of the reference object used to locate the question's subject, in the style of Figure 5.12.

Finally, Figure 5.14 breaks out the question representations in PCs 1 and 3, showing that the identity of the reference object as a *block* or a *pyramid* is primarily represented along the first principal component.

In total, Figures 5.12–5.14 present convincing evidence that the spoken-answer model learns internal representations much like those of the meaning-answer model. These representations quantize the input space and vary systematically along a number of principal dimensions to represent complex knowledge.

This section on the spoken-answer model, unlike the previous section on the meaning-answer model, does not contain a figure depicting information about the remaining property of the reference objects—the identifying number assigned to each. This is because an analysis of the representations showed no underlying systematicity to the representations when broken down by object number. However, this is not surprising. Being more speech-like, the responses that the spoken-answer model produces differ from those of the meaning-answer model in that they do not involve explicit specification of object numbers. Object numbers only exist in the visual input to the spoken-answer model, and to perform the question-answering task, the model must use them to bind attributes together to form coherent objects—that is, binding `(Color blue 2)` with `(Shape pyramid 2)` and `(Location near 2 3)` to produce the conception of a blue pyramid that is near some other object. Once this binding is complete for the entire environment, the model has no reason to retain the object number that was used to perform the binding; the number has served its function and is henceforth superfluous since it is not needed as part of the response. Therefore, one should not expect the high-level representations at the integration layer to involve object number at all, and indeed, the variation due to object number is small and unsystematic compared to the meaning-answer model.

5.5 Discussion

The results from the previous section suggest that for both the meaning-answer model and the spoken-answer model, observation and imitation are realistic methods by which a question-answering behavior can be learned. The meaning-answer model learned to answer questions by directly mimicking the intended meaning of another speaker, Sherlock. While such imitation may occasionally be possible in situations where the learner can readily infer the speaker's meaning, such situations might not be frequent enough to facilitate full language learning. In any event, such a model could only hope to explain how the answers are derived, but not how they are communicated. Besides serving as a simplified proof-of-concept and a stepping stone from the grounded-meaning model to the spoken-answer model, the meaning-answer model provides clear evidence that a purely neural network model is capable of integrating two separate sensory streams to produce a coherent whole. The meaning-answer model is the only known neural network model able to successfully learn to map sentence-level questions, represented at sub-lexical resolution, onto complex, composable symbolic meanings representing their answers.

The spoken-answer model improves on the meaning-answer model in two ways. First, it learns to perform the question-answering task not by mimicking Sherlock's meaning, but by mimicking Sherlock's speech. Even though speech is often a lossy translation of the meaning, it has the virtue of always being observable—a property that places this model closer to real-world plausibility. Second, the spoken-answer model's performance encompasses not only answer derivation but also response generation, giving it an extra level of explanatory power over the meaning-answer model. This final model is the only known neural network able to map sub-lexical representations of natural language questions to natural language answers, devising its own semantic representations along the way.

As was the case with the grounded-meaning model in Chapter 4, both of the models presented here appear to devise internal representations that are approximately compositional, imbuing them with the generative power of symbolic approaches to cognition. At the same time, these learned representations are distributed, conferring beneficial properties like redundancy and graceful degradation.

These models suggest that simple observation and imitation might be sufficient tools for learning to solve question-answering tasks. As discussed at the outset of this chapter, question answering—or more generally, the request/response mode of communication—is fundamental to language. Since the principles in question are so elementary, computational models that observe and imitate seem to be ideal for application to complex language learning tasks.

Chapter 6

Conclusion

The work presented in this thesis makes a variety of original contributions to multiple fields, including computer science, second-language acquisition, and cognitive neuroscience. It also has a number of limitations as well as possibilities for future work.

6.1 Contributions

The first major contribution is the generalized long short-term memory (LSTM-g), a variant of the original training algorithm for networks of the long short-term memory architecture. The original LSTM training algorithm has the desirable and biologically realistic trait that it depends only on information that is spatially and temporally local to the memory cell being trained; it also has a serious limitation in that it is applicable only to shallow networks with no more than a single hidden layer in series. Deeper recurrent networks are trainable by other means, but all previously known algorithms make use of information that is nonlocal in either time or space, seriously impairing their biological plausibility while simultaneously harming their efficiency. The new LSTM-g algorithm from Chapter 2 begins with the basic template of gradient descent on the error function and, by reorganizing the formalism and truncating error signals in a novel fashion, produces a general training algorithm for second-order networks with arbitrarily deep architectures while utilizing only local information. Combined with other recent results demonstrating the biological plausibility of gradient descent (if not specifically error back-propagating implementations thereof; Xie and Seung, 2003), LSTM-g provides an approach to neural network training that is simultaneously powerful, efficient, and biologically inspired.

The neural models of the acquisition of grammatical gender presented in Chapter 3 address the suitability of LSTM-g to model human learning artifacts, which has bearing on its potential to create more human-like natural language processing models. These models, in addition to their role of validating the use of LSTM-g in human language models in general, are major contributions of this research in their own right because they bear on the long-standing debate over critical period effects observed in the second-language acquisition literature. The models use diverse real-world input to examine how a language learner with no preconceived notions about grammatical gender—or even knowledge of its existence—can develop a functional knowledge of gender assignment and agreement by merely predicting the inputs it will see next. The models show that grammatical gender is highly learnable under these conditions by approaching human-level performance on gender-related tasks in two languages. Examinations of bilingual models show that final attainment is absolutely affected by entrenchment—that is, the longer the learner is exposed to the native language before a second language is introduced, the lower the learner's peak performance on tasks in the second language. This major effect is mediated in part by memory development, which lends support to the "less is more" hypothesis because model comparisons show that starting with a small working memory capacity and augmenting it during language learning contributes to better overall performance. Since these two contributors to critical period effects are usually irreparably confounded in human learners, the ability of the models to vary them independently is key to their importance as contributions to the second-language acquisition literature.

The greatest contributions of this dissertation, however, come from the neurocomputational models of grounded language comprehension, question answering, and language production introduced in Chapters 4 and 5. Of these, the spoken-answer model from Chapter 5 is the most advanced and serves as the culmination of the language modeling work in this dissertation. This model learns using fine-grained auditory input at the level of individual phonemes, segmenting the auditory stream into morphemes, words, and phrases grounded in its visual knowledge of its surroundings. The grounded-meaning model and the meaning-answer model learn to comprehend language and answer questions, respectively, by mapping them to an output representation consisting of sets of predicates that describe the intended meaning. In contrast, the spoken-answer model learns to understand speech by devising its own distributed representations of meaning. From these learned meaning representations, the spoken-answer model learns to speak, generating sequences of articulatory movements that specify speech streams forming complete-sentence answers to questions. The model does all this using the language-agnostic media of acoustics and articulation for its inputs and outputs, allowing it to, in theory, repeat the experiments presented here using virtually any spoken language, without modification. This feature of the model is fundamental because the human language-learning substrate is clearly just as general, allowing children to learn any language to which they are exposed.

All told, the spoken-answer model captures many important facets of the language learning process, including segmentation of unbroken phoneme streams, grounding of words and phrases in meanings derived from other senses, word selection and ordering for production, and the inferences required to answer questions. Though previous models have addressed some of these themes, no other known neural network models to date have combined them all, especially with so much care paid to input and output realism and biological plausibility of the underlying neural network methods. This puts the spoken-answer model in a unique position to serve as the basis for more human-relatable natural language processing models.

One important point that these models make clear is the compatibility of compositional, classically symbolic representations with distributed neural representations. Compositionality is key in achieving both efficient encoding of, and computation over, large multi-dimensional data. The assertions of cognitive scientists that mental representations must be compositional to explain human behavior (Fodor and Pylyshyn, 1988) need not be in conflict with assertions by connectionists that the brain's representations must be distributed. While it is difficult to intuit, such compatibility has been shown possible in theory (Smolensky, 1990; Plate, 1995), though little attention has been paid to how a learner might gradually modify its distributed representations to achieve compositionality. Chapters 4 and 5 present evidence that the models learn such compositional representations using gradient descent, constituting perhaps the strongest evidence to date that such learning is possible. However, this work does not shed light on how or why the models developed compositionality, leaving that question as a promising topic for future research.

6.2 Limitations and future work

Though the gender models of Chapter 3 serve both as an excellent justification for the human-like learning properties of LSTM-g and as a useful contribution to discussions about critical period effects, they have a number of limitations. The largest of these is that the models learn solely about the acoustic properties of words. This may be sufficient to study phonological, morphological, and possibly even syntactic properties of grammatical gender learning, but it completely ignores semantic aspects that are known to play a role in the assignment of gender to words. A more complete model would take semantic knowledge into account, although doing so with anything approaching realistic input would be a daunting challenge due to a lack of large semantically labeled corpora. A second limitation of the gender models is their restriction to two languages in the work presented—specifically two siblings in the lingustic family tree, French and Spanish, that have very similar gender systems. Since the models can, because of the phonological nature of their input and output representations, learn about gender systems of varying complexity in any spoken language, it would be enlightening to test them with further languages—in particular those having very different gender systems. It would also be useful to examine a variety of language pairs to discover whether similarity between the first- and second-language gender systems is a benefit or a detriment to bilingual learning. Finally, the models neglect a number of other documented causes of critical period effects, including social and motivational factors, which would be interesting to study in tandem with entrenchment and development.

The final spoken-answer model of Chapter 5 has limitations as well. For example, it could be improved to have stronger fidelity to human-like sensory inputs and motor outputs. The current auditory input, at the level of sequences of acoustic features, is essentially a single step removed from the frequency-sensitive neurons of the human cochlea. The model's visual inputs are predicates, on the other hand, which provide a built-in understanding of the environment that is much more abstract than a biologically motivated retinotopic map. The abstractness of these representations certainly aids the model, since the objects, attributes, and even relational predicates already correspond to words in heard sentences, rather than having to be built up in conjunction with the representations for the words themselves. Lowering the abstraction level of these inputs will be an important avenue for future extension of the spoken-answer model, once considerations of computational resources permit. Additionally, the spoken-answer model in its current form relies on direct error feedback at the articulatory level, which is not plausible for human learners. Current theories posit a forward model from articulation to audition, allowing a listener to predict the auditory signals that will arise from articulatory outputs. The learner can then compare these predictions with the heard signal to generate auditory prediction errors which can be transformed into articulatory error information (e.g., Kello and Plaut, 2004). A future version of the model could learn this forward model via a babbling stage and propagate errors backward across it to train speech production in a more natural manner.

A further limitation of the spoken-answer model involves the tasks that the model learned to perform. All of the open-class words used in the language learning tasks were concrete in the sense that they were tied directly to environmental stimuli. An important avenue for future work will be to include abstract notions, such as ownership, in the task.

Finally, the spoken-answer model is limited in its immediate applicability to the human brain because no attempt has been made to map its layers to specific brain regions. As mentioned in Chapter 1, one of the main goals of this modeling effort was to develop a sentence-level successor to the word-level Wernicke-Lichtheim-Geschwind (WLG) model developed by Weems and Reggia (2006). As such, a productive future avenue for investigation could include an attempt to extend the spoken-answer model by relating it to the brain regions present in the WLG model. Such work would also include a detailed lesioning study of the model, which could now focus on several aphasia metrics, such as fluency and grammaticality, that had no analogues in the original WLG model because it was limited to single words. Such a study could be invaluable in predicting deficits due to lesions in specific areas and might even be able to shed light on patient-specific therapies best suited to stimulate recovery.

In addition to its future extension as a brain model, the spoken-answer model, or even the simpler grounded-meaning model, has potential applicability in studies of psycholinguistic phenomena. Since a major focus of the model is grounding, one could easily teach the model to perform anaphor resolution by training it on an appropriate grammar. This would provide a fine-grained sense of the sorts of input necessary to produce a model that matches human behavior on the task. Listener interpretation of quantifier scope would provide another fertile test bed, as the model's interpretations of quantifiers can be observed via the grounded predicates it produces. Such a model could be studied during development to see if it makes the same interpretive missteps that children make when learning any particular language.

Many psycholinguistic tasks that explore listener interpretations would be ripe for investigation with this type of model.

This dissertation showcases important insights gained by starting from a low level in the hierarchy of the study of the brain and mind. From collections of simple interacting neurons, a wealth of data has emerged concerning the genesis and composition of cognitive representations, as well as the origin of behaviors as complex as the inference necessary to answer a question. This data can lead not only to more accurate models of cognition but also to more sophisticated natural language processing systems that more closely emulate a mind. The human brain is the most complex natural machine known. To fully understand the wonders that spring from it, we must study its organization at every level, from molecule to mind.

Bibliography

Abrahamsson, N. and K. Hyltenstam (2008). The robustness of aptitude effects in near-native second language acquisition. *Studies in Second Language Acquisition 30*(4), 481–509.

Abrahamsson, N. and K. Hyltenstam (2009). Age of onset and nativelikeness in a second language: Listener perception versus linguistic scrutiny. *Language Learning 59*(2), 249–306.

Baddeley, A. (2003). Working memory and language: An overview. *Journal of Communication Disorders 36*, 189–208.

Bayer, J., D. Wierstra, J. Togelius, and J. Schmidhuber (2009). Evolving memory cell structures for sequence learning. In *Proceedings of the International Conference on Artificial Neural Networks*, pp. 755–764.

Bley-Vroman, R. (1988). The fundamental character of foreign language learning. In W. Rutherford and M. S. Smith (Eds.), *Grammar and second language teaching: A book of readings*, pp. 19–30. New York: Newbury House.

Broca, P. (1861). Loss of speech, chronic softening and partial destruction of the anterior left lobe of the brain. *Bulletin de la Societe Anthropologique 2*, 235–238.

Carroll, S. E. (1995). The hidden dangers of computer modelling: Remarks on Sokolik and Smith's connectionist learning model of French gender. *Second Language Research 11*(3), 193–205.

Chang, F., G. Dell, and K. Bock (2006). Becoming syntactic. *Psychological Review 113*(2), 234–272.

Cochran, B. P., J. L. McDonald, and S. J. Parault (1999). Too smart for their own good: The disadvantage of a superior processing capacity for adult language learners. *Journal of Memory and Language 41*, 30–58.

Corbett, G. (1991). *Gender.* New York: Cambridge University Press.

Cowan, N., E. M. Elliott, J. Scott Saults, C. C. Morey, S. Mattox, A. Hismjatullina, and A. R. A. Conway (2005). On the capacity of attention: Its estimation and its role in working memory and cognitive aptitudes. *Cognitive Psychology 51*(1), 42–100.

CUMBRE (2010). *Corpus del Español Contemporáneo de España e Hispanoamérica.* Madrid: SGEL.

DeKeyser, R. M. (2000). The robustness of critical period effects in second language acquisition. *Studies in Second Language Acquisition 22*(4), 499–533.

DeKeyser, R. M. (in press). Age effects in second language learning. In S. Gass and A. Mackey (Eds.), *Handbook of second language acquisition.* London: Routledge.

DeKeyser, R. M., I. Alfi-Shabtay, and D. Ravid (2010). Cross-linguistic evidence for the nature of age effects in second language acquisition. *Applied Psycholinguistics 31*(3), 413–438.

DeKeyser, R. M. and J. Larson-Hall (2005). What does the critical period really mean? In J. F. Kroll and A. M. B. de Groot (Eds.), *Handbook of bilingualism: Psycholinguistic approaches*, pp. 89–108. Oxford: Oxford University Press.

Dell, G. (1993). Structure and content in language production: A theory of frame constraints in phonological speech errors. *Cognitive Science 17*(2), 149–195.

Diederich, J. and D. L. Long (1991). Efficient question answering in a hybrid system. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 479–484.

Duncan, J., R. J. Seitz, J. Kolodny, D. Bor, H. Herzog, and A. Ahmed (2000). A neural basis for general intelligence. *Science 289*(5478), 399–401.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science 14*, 179–211.

Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition 48*(1), 71–99.

Ferrucci, D., E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty (2010). Building Watson: An overview of the DeepQA project. *AI Magazine 31*(3), 59–79.

Fodor, J. A. and Z. W. Pylyshyn (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition 28*(1-2), 3–71.

Frank, S. L., W. F. G. Haselager, and I. van Rooij (2009). Connectionist semantic systematicity. *Cognition 110*(3), 358–379.

Gathercole, S. E. (1999). Cognitive approaches to the development of short-term memory. *Trends in Cognitive Sciences 3*(11), 410–419.

Gers, F. A. and F. Cummins (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation 12*(10), 2451–2471.

Gers, F. A., J. A. Pérez-Ortiz, D. Eck, and J. Schmidhuber (2003). Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets. *Neural Networks 16*(2), 241–250.

Gers, F. A. and J. Schmidhuber (2000). Recurrent nets that time and count. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 189–194.

Gers, F. A. and J. Schmidhuber (2001). LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks 12*(6), 1333–1340.

Geschwind, N. (1965). Disconnexion syndromes in animals and man. *Brain 88*, 585–644.

Giles, C. L. and T. Maxwell (1987). Learning, invariance, and generalization in high-order neural networks. *Applied Optics 26*(23), 4972–4978.

Goldowsky, B. N. and E. L. Newport (1993). Modeling the effects of processing limitations on the acquisition of morphology: The less is more hypothesis. In E. Clark (Ed.), *Proceedings of the 24th Annual Child Language Research Forum*, pp. 124–138. Stanford, CA: Center for the Study of Language and Information.

Graesser, A. C. and S. P. Franklin (1990). QUEST: A cognitive model of question answering. *Discourse Processes 13*(3), 279–303.

Graves, A., D. Eck, N. Beringer, and J. Schmidhuber (2004). Biologically plausible speech recognition with LSTM neural nets. In *Proceedings of the International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, pp. 127–136.

Graves, A. and J. Schmidhuber (2008). Offline handwriting recognition with multidimensional recurrent neural networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Eds.), *Neural Information Processing Systems 21*, pp. 545–552. Vancouver: MIT Press.

Guillelmon, D. and F. Grosjean (2001). The gender marking effect in spoken word recognition: The case of bilinguals. *Memory & Cognition 29*, 503–511.

Hadley, R. (1994). Systematicity in connectionist language learning. *Mind & Language 9*(3), 247–272.

Hadley, R. F. and V. C. Cardei (1999). Language acquisition from sparse input without error feedback. *Neural Networks 12*(2), 217–235.

Hadley, R. F. and M. Hayward (1997). Strong semantic systematicity from Hebbian connectionist learning. *Minds and Machines 7*(1), 1–37.

Hakuta, K., E. Bialystok, and E. Wiley (2003). Critical evidence: A test of the critical-period hypothesis for second-language acquisition. *Psychological Science 14*(1), 31–38.

Harley, B. (1979). French gender "rules" in the speech of English-dominant, French-dominant and monolingual French-speaking children. *Working Papers in Bilingualism 19*, 129–156.

Harnad, S. (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena 42*(1-3), 335–346.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence 40*(1-3), 185–234.

Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural Computation 9*(8), 1735–1780.

Holmes, V. and B. de la Bâtie (1999). Assignment of grammatical gender by native speakers and foreign learners of French. *Applied Psycholinguistics 20*, 479–506.

Hyltenstam, K. and N. Abrahamsson (2003). Maturational constraints in second language acquisition. In C. J. Doughty and M. H. Long (Eds.), *Handbook of second language acquisition*, pp. 539–588. Oxford, UK: Blackwell.

Iverson, P., P. K. Kuhl, R. Akahane-Yamada, E. Diesch, Y. Tohkura, and A. Kettermann (2003). A perceptual interference account of acquisition difficulties for non-native phonemes. *Cognition 87*, B47–B57.

Jaeger, H. (2001). *The "echo state" approach to analysing and training recurrent neural networks.* GMD Report 148, German National Research Center for Information Technology.

Jaeger, H. (2002). *Short term memory in echo state networks.* GMD Report 152, German National Research Center for Information Technology.

Jakobson, R., G. Fant, and M. Halle (1951). *Preliminaries to speech analysis: The distinctive features and their correlates.* Boston: MIT Press.

Jia, G. and D. Aaronson (2003). A longitudinal study of Chinese children and adolescents learning English in the United States. *Applied Psycholinguistics 24*(1), 131–161.

Jia, G., D. Aaronson, and Y. Wu (2002). Long-term language attainment of bilingual immigrants: Predictive variables and language group differences. *Applied Psycholinguistics 23*(4), 599–621.

Johnson, J. S. and E. L. Newport (1989). Critical period effects in second language learning: The influence of maturational state on the acquisition of English as a second language. *Cognitive Psychology 21*, 60–99.

Jolliffe, I. T. (2002). *Principal component analysis* (2nd ed.). New York: Springer-Verlag.

Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Conference of the Cognitive Science Society*, pp. 531–546.

Kareev, Y., I. Lieberman, and M. Lev (1997). Through a narrow window: Sample size and the perception of correlation. *Journal of Experimental Psychology: General 126*(3), 278–287.

Kello, C. T. and D. C. Plaut (2004). A neural network model of the articulatory-acoustic forward mapping trained on recordings of articulatory parameters. *Journal of the Acoustical Society of America 116*(4), 2354–2364.

Kersten, A. W. and J. L. Earles (2001). Less really is more for adults learning a miniature artificial language. *Journal of Memory and Language 44*, 250–273.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE 78*, 1464–1480.

Lapkin, S. and M. Swain (1977). The use of English and French cloze tests in a bilingual education program evaluation: Validity and error analysis. *Language Learning 27*, 279–314.

Lenneberg, E. H. (1967). *Biological foundations of language.* New York: Wiley.

Lew-Williams, C. and A. Fernald (2010). Real-time processing of gender-marked articles by native and non-native Spanish speakers. *Journal of Memory and Language 63*, 447–464.

Li, P., I. Farkas, and B. MacWhinney (2004). Early lexical development in a self-organizing neural network. *Neural Networks 17*(8-9), 1345–1362.

Li, P., X. Zhao, and B. MacWhinney (2007). Dynamic self-organization and early lexical development in children. *Cognitive Science 31*(4), 581–612.

Lichtheim, L. (1885). On aphasia. *Brain 7*, 433–484.

Long, M. (1990). Maturational constraints on language development. *Studies in Second Language Acquisition 12*(3), 251–285.

MacWhinney, B. (2006). Emergent fossilization. In Z. Han and T. Odlin (Eds.), *Studies of fossilization in second language acquisition*, pp. 134–156. Clevedon, UK: Multilingual Matters.

MacWhinney, B., J. Leinbach, R. Taraban, and J. McDonald (1989). Language learning: Cues or rules? *Journal of Memory and Language 28*(3), 255–277.

Markert, H., U. Kaufmann, Z. Kara Kayikci, and G. Palm (2009). Neural associative memories for the integration of language, vision and action in an autonomous agent. *Neural Networks 22*(2), 134–143.

Matthews, C. A. (1999). Connectionism and French gender attribution: Sokolik and Smith re-visited. *Second Language Research 15*, 412–427.

Mayberry, R. I., E. Lock, and H. Kazmi (2002). Linguistic ability and early language exposure. *Nature 417*(6884), 38.

Miikkulainen, R. (1993). *Subsymbolic natural language processing: An integrated model of scripts, lexicon, and memory.* Cambridge, MA: MIT Press.

Miikkulainen, R. (1998). Text and discourse understanding: The DISCERN system. In R. Dale, H. Moisl, and H. Somers (Eds.), *A handbook of natural language processing: Techniques and applications for the processing of language as text.* New York: Marcel Dekker.

Miller, C. B. and C. L. Giles (1993). Experimental comparison of the effect of order in recurrent neural networks. *Pattern Recognition 7*(4), 849–872.

Monner, D. and J. A. Reggia (2009). An unsupervised learning method for representing simple sentences. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 2133–2140.

Monner, D. and J. A. Reggia (2011). Systematically grounding language through vision in a deep, recurrent neural network. In *Proceedings of the Conference on Artificial General Intelligence*, Berlin. Springer-Verlag.

Monner, D. and J. A. Reggia (in press a). A generalized LSTM-like training algorithm for second-order recurrent neural networks. *Neural Networks*.

Monner, D. and J. A. Reggia (in press b). Towards a biologically inspired question-answering neural architecture. In *Proceedings of the Conference on Biologically Inspired Cognitive Architectures*, Amsterdam. IOS Press.

Monner, D. and J. A. Reggia (under review). Systematicity and emergent latent symbol systems in recurrent neural networks.

Monner, D., K. Vatz, G. Morini, S.-O. Hwang, and R. DeKeyser (under review). A neural network model of the effects of entrenchment and memory development on grammatical gender learning.

Movellan, J. R. (1990). Contrastive Hebbian learning in the continuous Hopfield model. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton (Eds.), *Connectionist Models: Proceedings of the 1990 Summer School*, pp. 10–17. San Mateo, CA: Morgan Kaufmann.

New, B. (2006). *Lexique 3: Une nouvelle base de données lexicales.* Louvain, Belgique: Actes de la Conférence Traitement Automatique des Langues Naturelles (TALN 2006).

Newport, E. (1988). Constraints on learning and their role in language acquisition: Studies of the acquisition of American Sign Language. *Language Sciences 10*, 147–172.

Newport, E. (1990). Maturational constraints on language learning. *Cognitive Science 14*(1), 11–28.

O'Reilly, R. C. (2001). Generalization in interactive networks: The benefits of inhibitory competition and Hebbian learning. *Neural Computation 13*(6), 1199–1241.

O'Reilly, R. C. and M. J. Frank (2006). Making working memory work: A computational model of learning in the prefrontal cortex and nasal ganglia. *Neural Computation 18*(2), 283–328.

Paget, R. (1976). Vowel resonances. In D. Fry (Ed.), *Acoustic phonetics*, pp. 95–103. Cambridge: Cambridge University Press.

Paradis, M. (2009). *Declarative and procedural determinants of second languages.* Amsterdam: Benjamins.

Phillips, C. (2001). Levels of representation in the electrophysiology of speech perception. *Cognitive Science 25*(5), 711–731.

Plate, T. (1995). Holographic reduced representations. *IEEE Transactions on Neural Networks 6*(3), 623–641.

Plaut, D. and C. Kello (1999). The emergence of phonology from the interplay of speech comprehension and production: A distributed connectionist approach. In B. MacWhinney (Ed.), *The emergence of language*, pp. 381–415. Mahwah, NJ: Lawrence Erlbaum Associates.

Prokhorov, D. (2005). Echo state networks: Appeal and challenges. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 1463–1466.

Psaltis, D., C. Park, and J. Hong (1988). Higher order associative memories and their optical implementations. *Neural Networks 1*(2), 149–163.

Puskorius, G. V. and L. A. Feldkamp (1994). Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks. *IEEE Transactions on Neural Networks 5*(2), 279–297.

Rohde, D. L. T. (2002). *A connectionist model of sentence comprehension and production*. Ph.D. dissertation, Carnegie Mellon University.

Rohde, D. L. T. and D. C. Plaut (1999). Language acquisition in the absence of explicit negative evidence: How important is starting small? *Cognition 72*(1), 67–109.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors. *Nature 323*(9), 533–536.

Scherag, A., L. Demuth, F. Rösler, H. J. Neville, and B. Röder (2004). The effects of late acquisition of L2 and the consequences of immigration on L1 for semantic and morphosyntactic language aspects. *Cognition 93*, B97–B108.

Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the International Conference on New Methods in Language Processing*, pp. 44–49.

Schmidhuber, J., D. Wierstra, M. Gagliolo, and F. Gomez (2007). Training recurrent networks by Evolino. *Neural Computation 19*(3), 757–779.

Schulz, R. and J. A. Reggia (2004). Temporally asymmetric learning supports sequence processing in multi-winner self-organizing maps. *Neural Computation 16*(3), 535–661.

Schulz, R. and J. A. Reggia (2005). Mirror symmetric topographic maps can arise from activity-dependent synaptic changes. *Neural Computation 17*(5), 1059–1083.

Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences 3*(03), 417–457.

Shin, Y. and J. Ghosh (1991). The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 13–18.

Singh, K. (1976). *Distinctive features: Theory and validation*. New York: University Park Press.

Smolensky, P. (1988). The constituent structure of connectionist mental states: A reply to Fodor and Pylyshyn. *Southern Journal of Philosophy 26*(S1), 137–161.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence 46*, 159–216.

Sokolik, M. E. and M. E. Smith (1992). Assignment of gender to French nouns in primary and secondary language: A connectionist model. *Second Language Research 8*, 39–58.

St. John, M. F. and J. L. McClelland (1990). Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence 46*(1-2), 217–257.

Surridge, M. E. (1989). Le facteur sémantique dans l'attribution du genre aux inanimés en français. *Revue Canadienne de Linguistique 34*, 19–44.

Surridge, M. E. (1993). Gender assignment in French: The hierarchy of rules and the chronology of acquisition. *International Review of Applied Linguistics in Language Teaching 31*, 77–95.

Surridge, M. E. (1995). *Le ou la? The gender of French nouns.* Philadelphia: Multilingual Matters.

Sutton, R. S. and A. G. Barto (1998). Temporal-difference learning. In *Reinforcement Learning: An Introduction*, pp. 167–200. Cambridge, MA: MIT Press.

Teschner, R. V. and W. M. Russell (1984). The gender patterns of Spanish nouns: An inverse dictionary-based analysis. *Hispanic Linguistics 1*, 115–132.

Tomasello, M. (2003). *Constructing a language: A usage-based theory of language acquisition.* Cambridge, MA: Harvard University Press.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind 59*(236), 433–460.

Ullman, M. T. (2004). Contributions of memory circuits to language: The declarative/procedural model. *Cognition 92*, 231–270.

Uylings, H. B. M. (2006). Development of the human cortex and the concept of "critical" or "sensitive" periods. *Language Learning 56*, 59–90.

von der Malsburg, C. (1973). Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik 14*(2), 85–100.

Weems, S. A. and J. A. Reggia (2006). Simulating single word processing in the classic aphasia syndromes based on the Wernicke-Lichtheim-Geschwind theory. *Brain and Language 98*(3), 291–309.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE 78*(10), 1550–1560.

Wernicke, C. (1874). Der aphasische symptomenkomplex eine psychologische studie auf anotomomischer basis. In G. Eggert (Ed.), *Wernicke's work on aphasia*, pp. 219–283. The Hague: Mouton.

Wikipedia (2011). Wikipedia, the free encyclopedia. [Online; accessed January 2011].

Williams, P. and R. Miikkulainen (2006). Grounding language in descriptions of scenes. In *Proceedings of the Conference of the Cognitive Science Society*, pp. 2381–2386.

Williams, R. J. and D. Zipser (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation 1*(2), 270–280.

Xie, X. and H. S. Seung (2003). Equivalence of backpropagation and contrastive Hebbian learning in a layered network. *Neural Computation 15*(2), 441–454.