Hierarchy and interpretability in neural models of language processing

ILLC Dissertation Series DS-2020-06

Institute for Logic, Language and Computation

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 107
1098 XG Amsterdam
phone: +31-20-525 6051
e-mail: illc@uva.nl
homepage: http://www.illc.uva.nl/

The investigations were supported by the Netherlands Organization for Scientific Research (NWO), through a Gravitation Grant 024.001.006 to the Language in Interaction Consortium.

Publisher: Boekengilde

Printed and bound by printenbind.nl

ISBN: 90–6402–222-1

Hierarchy and interpretability in neural models of language processing

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen
op woensdag 17 juni 2020, te 13 uur

door

Dieuwke Hupkes

geboren te Wageningen

Promotiecommisie

Promotores:	Dr. W.H. Zuidema	Universiteit van Amsterdam
	Prof. Dr. L.W.M. Bod	Universiteit van Amsterdam
Overige leden:	Dr. A. Bisazza	Rijksuniversiteit Groningen
	Dr. R. Fernández Rovira	Universiteit van Amsterdam
	Prof. Dr. M. van Lambalgen	Universiteit van Amsterdam
	Prof. Dr. P. Monaghan	Lancaster University
	Prof. Dr. K. Sima'an	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

to

my parents Aukje and Michiel

Contents

Part One: Artificial languages

Part Two: Natural language

Part Three: Guiding models

Acknowledgments

I have frequently heard that the acknowledgement section is the most read section of a dissertation. It is also the only section in which I can write a bit erratically and exercise my (from Dutch stemming?) preference to make endlessly long sentences with multiple sidenotes in parentheses. So that is what I will do.

I want to start by thanking Jelle and Khalil, who both gave me a chance at a moment that many others might not have. To explain this to you, I have to go a bit back in time. Every since I was a little kid, I have always been extremely interested in language, but my main topic of study when I started at the university was physics. As a sidenote, while knowledge of electrodynamics, thermal physics and quantum mechanics are not frequently directly important to my current research, I do feel that the bachelor of physics programme at the UvA gave me an incredibly useful foundation to study any other topic after. They will most likely never read this, but I would like to express my gratitude to the people that are responsible for this programme as well as the teachers that taught me. Some of them (Auke Pieter, Erik, Stan) I still see regularly walking around at the Nikhef building and they remind me of a good time.

To go back to the main story: the physics programme provided me with an excellent foundation, but it of course left me with many gaps in knowledge when I started to study natural language processing. In particular, apart from implementing numerical solutions to differential equations in matematica, I did not know how to program. When I took my first natural language processing course, taught by Khalil (who, I want to mention, was an incredibly inspiring teacher!) this became a substantial hurdle, which I solved by partnering up with someone – Zé Pedro – who did know how to program. Zé programmed our parser, while I wrote the report. We got a high mark for it, but I felt bad about pretending to be able to program, so I went to Khalil and told him that I did not do any programming and wrote only the report. After asking me if I understood the underlying concepts (I did), he told me that if I promised him to program a parser at some later point in my life that he had no problem with me not being able to

program now. With this kind gesture, he allowed me to pass my first NLP course.

Of course, my programming problems did not suddenly end after one block, and when I arrived at my next NLP course, taught by Jelle, I was still struggling. I got stuck on the second assignment of Jelle's course (at that time, Jelle started his course with one lecture of command line tools using regular expressions I had no problem finishing that assignment, and I became a huge fan of hacking through text files using the things he taught me; now, I usually search online when I cannot figure out what to do myself, but for several years after his course I frequently referred back to the examples in his assignment!). Not being able to finish this second assignment, I decided to drop Jelle's course, upon which Jelle called me into his office and told me he would be sad to see me go, because it seemed from the seminar sessions I really enjoyed the course. I told him I did, but that I did not manage to finish the second assignment. He forgave me the assignment and, later, I passed my second NLP course.

Jelle and Khalil, while I still frequently think back of these two moments – that perhaps you do not even remember anymore – I am not sure if I ever really thanked you for giving me these opportunities, so I want to do so now: we will never know if I would have given up, but I might have, and I owe you many thanks for how you approached me.

Of course, keeping me on board at that course is far from the only thing that I would like to thank Jelle for. I learned many many things from him (not only the invaluable bash-hacking with regular expressions!) and I have always been amazed by his vision and his ability to provide useful feedback even on projects that he hardly knew anything about. Jelle, I am extremely grateful for the academical guidance that you have given me over the years, for that you stood up for me when I needed that, and for that you always gave me the freedom to develop myself and my own ideas, even in cases where that was not necessarily in your own best interest.

Academically speaking, there are a few more people that I would like to thank. First of all, I would like to thank Rens, who read my entire dissertation with a speed almost unimaginable. I really appreciate his usually extremely quick responses. Directly related to my dissertation, I would like to thank also my committee members, who took the time to read and assess my work.

A bit more broadly I would like to thank all the people that I collaborated with in the past couple of years. I was lucky to collaborate with many different people, researchers from other universities but also students at the University of Amsterdam. I will not mention them all by name, but I do want to thank them all for what they taught me and the students in particular for trusting me to supervise them. I learned a lot from all of you!

Also the rest of my colleagues and the ILLC in general I want to thank as a group, making an exception for my office mates Malvin and Jouke. It was great to share an office with you; thanks for helping me out with stupid problems (and not

laughing at me) and, importantly, thanks for following me with the sitting balls! The ILLC provided me a great environment to learn and work and I enjoyed the chats with everyone (including the support staff!) in the hallways. I genuinely hope I will get to return as a permanent staff member some day.

A colleague (who also became my partner) that I do want to mention specifically is Elia, to whom I owe a great deal of thanks. We did many projects together and I learned a lot from him. He not only was responsible for a huge boost in my productivity and even my career in general, through his connections and managing capabilities, but he also helped me refind my love for doing research. When I met Elia, I was unconvinced I wanted to continue my academic career after my PhD. I am not sure what exactly it was – his enthusiasm and drive, or the many interesting projects we did together – but after working with him for less than half a year I could not imagine anymore doing anything else than researching natural language processing. Thank you Elia, I hope that I may do many many more projects with you!

On the edge of personal and academic I want to thank also my younger brother Elte. He answered my endless and probably very naive questions about linux, terminals, dependency errors, git and all things technical. He saved me many times when I messed up my operating system or my github repository and thanks to him I know now that git $\neq$ github, that the ruby installation of the ubuntu repository is a mess, and I can install my own operating system. I am now also able to find the answers to most of my technical problems using Google, although incidentally I still ask him for help. Thank you my sweet brother.

Lastly, on a purely personal level, I would like to thank my friends for supporting me and distracting me when I was stressed. For all of them, I could write down something special they did during this period, but I will only mention two people in particular: Judith, who is one of my oldest friend and was always available to listen to my stories and Suzy, my pole dance partner. To Suzy I would like to say: thank you for everything. We started as just pole partners, but training with you was always such a delight – no matter how stressed out I was – that you became my best friend and spending time with you became invaluable for my mind.

Lastly, the very most important thanks go to my parents, to whom I dedicated this dissertation. Aukje en Michiel, over de beslissing om dit werk aan jullie op te dragen hoefde ik niet eens een fractie van een seconde na te denken. De lijst van dingen waar ik jullie voor zou willen bedanken is eindeloos; jullie vertrouwen, liefde en voorbeeld zijn van onschatbare waarde. Ik heb een diepe bewondering voor jullie beide en ik kan me geen betere ouders wensen of voorstellen ♡.

Chapter 1

Introduction

When I started this thesis project in 2015, the landscape of the field of computational linguistics looked substantially different from what it looks like now. Transformers did not yet exist, machine translation models were still commonly using n-grams as their target side language model, and the paper proposing the now widely used optimiser *Adam* (Kingma and Ba, 2015) had not yet been officially published. Recurrent neural networks – that until then had been of interest mostly to cognitive scientists – had just cautiously started to outperform symbolic models, in some domains.

1.1 My original plan

The ambitious plan of my four-year project was to build upon these preliminary successes to build a *neurally plausible semantic parser* that would combine symbolic knowledge of the structure of language and the cognitively interesting neural network models. My main motivation to embark on this project was my interest in the structure of language and my curiosity about how this structure could be represented in human brains. Connectionist architectures – inspired by the human brain and starting to be more successful on interesting natural language tasks – formed a perfect starting point for this enterprise. I was enthusiastic about incorporating existing (symbolic) knowledge about natural language within artificial neural networks to combine the best of both worlds.

My project plan described two steps. First, I would focus on understanding how and why different parts of the current – mostly symbolic, some neural – models of hierarchical sentence processing worked. Then, I would use these insights to *develop models in which we gradually eliminate assumptions such as discretisation of time, discrete, symbolic categories and pre-given tree structures.* The project plan furthermore mentioned that I would test this neurally plausible parser on *large* corpora such as the Penn Treebank (PTB) and the Stanford Sentiment Treebank. For the current reader, with knowledge of the current models, in 2019,

it might not come as a large surprise that I quickly deviated from this two-step plan.

Quickly after I started my project, I discovered that I could not tweak, adapt or understand neural models like I was used to with symbolic models. Therefore, integrating existing knowledge into such models was a difficult undertaking. How difficult exactly is exemplified by the main sources of the recent successes of neural models. While many researchers have tried to improve neural networks with domain knowledge, it has thus not (yet) lead to groundbreaking improvements of neural network models. The most prominent successes in what is now commonly called *deep learning* should instead be attributed to computational advances that allowed us to train models on larger data sets (the PTB does not qualify as a large corpus anymore), technical advances that optimised the training algorithms (e.g. Kingma and Ba, 2015) and engineering advances that resulted in new architectures (e.g. Bahdanau et al., 2015; Vaswani et al., 2017).

I sometimes find it disappointing that the main improvements did not stem from incorporating external knowledge about the phenomenon that is modelled and that the actual advances are also not particularly interesting for researchers interested in cognition or language. However, the resulting new generation of models that perform remarkably well on many natural language processing tasks did revive the relevance of considering connectionist models as alternative models of language processing. Contrary to the early days of connectionism, in which neural networks were typically trained to model small data sets of fewer than 100 sentences, we now have the opportunity to study neural network models that can capture many interesting phenomena. This observation forms the basis of the work presented in this dissertation, in which I will (re)consider how useful neural network models are to further our understanding of human language processing. Or, in other words, I examine whether such models can serve as *explanatory models* of language processing, where I focus in particular on the processing of the *hierarchical compositional structure* of natural language.

1.2 Neural networks as explanatory models

For a neural network to be useful as an explanatory model of human language processing, there are three important prerequisites:

i) The model should share some relevant aspects of processing with humans (**architectural similarity**);

ii) The model should be able to represent or approximate the behaviour of humans with respect to some phenomena that we would like to understand (**behavioural similarity**);

iii) We should be able to obtain insight into *how* it implements or processes these phenomena (**model interpretability**).

1.2.1 Architectural similarity

The first prerequisite for explanatory models is that they should have some properties relevant to the modeller. To give a simple example, if someone wants to build an explanatory model to understand how particular types of sequences can be modelled incrementally, it is important that the model used to do so also receives sequences incrementally. Similarly, for a model to be useful to understand how language is processed by humans, it should share some relevant properties with the human processing system. It is my conviction that neural networks do.

Why neural networks? When I started to look into the capabilities and internal dynamics of neural network models, I often had to defend why I was interested in them. Many linguists I spoke to were skeptical of leaving the insightful symbolic models for the 'black-box' neural network models. Neuroscience-oriented people, on the other hand, often considered artificial neural networks too far away from the real brain to be able to tell us something useful about humans.

Like the linguists that asked critical questions about this line of research, also I preferred symbolic models of language, because such models are easy to use as explanatory models that help explain the data they describe. There is, however, also an important concern that symbolic models cannot address: how can the symbolic view be reunited with the architecture of the human brain? The brain does not have any obvious means to represent rules or symbols, and a symbolic model – even if it perfectly describes language – can thus not tell us why language is structured like that, or how such a system can be implemented by a human brain. Artificial neural network models, provided they can adequately model these symbolic aspects of language that we care about, might.

While also neural networks are, indeed, in many aspects incomparable with real brains, they do share some – in my opinion very relevant – dimensions with the human processing system: they have no explicit means to represent rules and symbols and – in case of recurrent networks – have to process incoming data sequentially, incrementally and in linear time. As such, if we can understand them, they might in fact serve as explanatory models – of how the seemingly symbolic structure of language can be implemented in a system of interconnected units that sequentially process structured sequences.

Why *recurrent* neural networks? Researching the capabilities and internal dynamics of neural network models has now become part of an established and even relatively mainstream field and I rarely have to defend studying neural networks anymore. I do sometimes have to defend, however, why my primary focus is on recurrent models instead of the newer, currently very popular all-attention-based architectures that outperform recurrent networks in most applied domains. Concerning this dissertation, a first obvious argument is that these models did not exist for the larger part of my PhD project. For my second and

more important argument to study recurrent models, which I already touched before, I would like to cite the first sentence of the article in which Jeffrey Elman introduced the simple recurrent network:

> *Time is clearly important in cognition. It is inextricably bound up with many behaviors (such as language) which express themselves as temporal sequences.* (Elman, 1990, p1.)

All-attention architectures sacrifice this important temporal and incremental aspect of human cognition and language processing. I therefore argue that such models might be very useful to learn more about the structure of language from a static perspective, but can – like symbolic models – not directly provide insight in how the structures in language can be processed by humans. For that reason, in this dissertation, I focus almost exclusively on recurrent architectures, and make only little deviations to discuss attention-based models and convolutional models to validate results.

1.2.2 Behavioural similarity

The second prerequisite concerns a model's ability to represent the phenomena that the researcher is interested in. In my case, this refers to the ability to correctly process natural language and its hierarchical structure. When I started my PhD project, very little literature was to be found on this topic. I could find theoretical papers on what models could do in theory (e.g. Siegelmann and Sontag, 1995), mathematical descriptions of how variable binding can be hardwired in a network (e.g. Smolensky, 1990) and small scale studies that demonstrated that and how (sometimes hand-crafted) recurrent networks can implement small context-free or even context-sensitive languages (e.g. Gers and Schmidhuber, 2001; Rodriguez, 2001). I found very little on what models trained in an end-to-end fashion with back-propagation learn when facing natural data.

This gap of knowledge resulted in the first major aim of my dissertation: increasing our understanding of what neural networks learn when they are trained with backpropagation on quite large amounts of data and to what extent they can adequately deal with hierarchical structure. Most of the studies I present focus on uncovering whether a specific aspect of hierarchical compositionality or syntactic structure is learned under some particular circumstances. I consider artificial setups, in which the hierarchical compositional structure is very crisp and clear, but also more naturalistic scenarios, in which networks are trained on noisy naturalistic data. While conducting these studies, I also discovered that what exactly the requirements are that we would like models to fulfill is not well specified and agreed upon. In other words, it is not clear what we would like them to learn under which circumstances. In Chapter 4 of this dissertation, I address this issue specifically, by teasing apart different aspects of processing of

structure that may be important to different researchers, and testing for them independently.

1.2.3 Model interpretability

Lastly, for a model to be useful as an explanatory model of a particular phenomenon, it is important that we at least to some extent understand *how* it implements this phenomenon, which is formulated in the third prerequisite. For symbolic models, this requirement is easily fulfilled, because their parameters usually have clear meanings – e.g. *the probability that word X has more than one left dependent* (DMV, Klein and Manning, 2004). This makes them very suitable as explanatory models. The meaning of the parameters (or 'weights') of neural networks is usually unclear.

To use neural networks as explanatory models, it is thus important to develop techniques to understand their behaviour and internal dynamics. The second aim in almost all of my chapters is therefore to develop or test techniques to increase our understanding of neural networks. One of these techniques, that I develop and explore in different ways in many of the chapters, is *diagnostic classification*. I will also cover several techniques borrowed and adapted from other fields.

1.3 Summary and outline

Overall, this thesis describes a series of studies that encompass the usefulness of recurrent neural networks as explanatory models of human language processing. These studies include extensive analyses of what aspects of hierarchical compositionality and syntactic structure recurrent networks can learn, the development of a set of techniques that can be used to analyse also future networks and an in-depth analysis of what researchers may mean when they talk about compositionality in the context of neural networks. At the end of this dissertation, I explore how this knowledge could be exploited to improve the extent to which models show the behaviour we desire, contextualise this research done in this dissertation and elaborate on how I think this work has contributed to different fields.

Outline

This dissertation is divided in three parts, preceded by a chapter that describes the relevant background to this dissertation. In this background chapter, I explain the three main recurrent architectures I study and describe two strands of research that have considered the abilities of such models to represent hierarchical structure and compositionality.

Part 1 In part 1, consisting of Chapter 3 and Chapter 4, I consider artificial setups. In the first of these chapters, I focus on the extent to which different recurrent architectures can learn to process hierarchically compositional structures. I also introduce *diagnostic classification*, one of the main interpretability techniques proposed in this dissertation. In the second chapter, I present a study that takes a closer look at what it means for a neural model to be able to process compositional structure. I identify different components related to compositionality motivated by linguistics and philosophy and propose a series of tests that aims to shed some light on how they relate to each other.

Part 2 In part (Chapter 5, 6 and 7) I focus on *language models*, which are trained on naturalistic data (in this dissertation: english sentences). These three chapters all consider whether and how such language models can represent long-distance relationships that indicate an understanding of hierarchical structure, taking *subject-verb agreement* as a case study. To do so, they use different methods: in Chapter 5, I use different versions and extensions of diagnostic classification; in Chapter 6, I describe a study that uses *neuron ablation* and again diagnostic classification; in Chapter 7, I develop *generalised contextual decomposition*.

Part 3 In Chapter 8, the last content chapter of this dissertation, I investigate if the solution that a model finds can be changed by changing the learning signal of the model. This chapter tests a hypothesis about a potential cause that models' solutions sometimes deviate from desired solutions. Furthermore, in this chapter I investigate if it is possible to understand from a trained model's parameters if it has learned a solution that incorporates structure.

Conclusion and discussion Chapter 9 contains the conclusion and a discussion of this dissertation. In this chapter, I summarise and review the research I did, speak briefly about its societal impact and sketch some paths for future work.

Chapter 2

Recurrent neural networks and grammatical structure

In this dissertation, I explore the usefulness of artificial neural networks as explanatory models of language processing. To do so, I consider two different topics: I investigate whether artificial neural networks are able to learn the hierarchical compositional structures assumed to be important for natural language, and I develop a series of interpretability techniques that can be used to understand how these models represent and process this. In the current chapter, I provide some background information that helps the reader to understand and appreciate the rest of this thesis.

In particular, I first provide a detailed description of the main models of interest for this dissertation: *recurrent neural networks* (RNNs). For many purposes it might be sufficient to understand these models on a global level; in the context of this dissertation, it is useful to have a more detailed understanding of how they work. In the next section, I therefore discuss in some detail the three most prominent recurrent cells, providing both their mathematical description as well as a more intuitive account of the functions of their components. I also briefly explain how they are used to form different types of recurrent architectures.

Then, in Section 2.3 and 2.4, I describe two different approaches to understand the extent to which neural networks can capture grammatical structure. These approaches match the ones I take myself in part one and part two of this dissertation. For both approaches I give several examples, of which I more elaborately discuss one that I consider exemplary for this particular approach.

2.1 Recurrent neural networks

Recurrent neural networks (RNNs) are artificial neural networks that consist of one or multiple layers of interconnected units that together process temporal inputs. At any point in time, the state s_t of a recurrent neural network is computed based

on the current input and its *previous* state:

$$s_t = f(x_t, s_{t-1}) \tag{2.1}$$

Because of the *recurrent* connection of the network with its previous hidden state, it can remember information from earlier in the input sequence and connect it with the current input.

The difference between different recurrent cells resides in how the function to go from one step to the next is defined. In what follows, I explain this computational step for simple recurrent networks (SRNs), gated recurrent units (GRUs), and long short term memory networks (LSTMs). All the described computation steps concern *one layer* networks but can be easily extended to include more layers: the current state of one layer can then simply be seen as the input for the next one.

2.1.1 Simple recurrent networks

The most elementary form of recurrent neural network is the *Simple Recurrent Network* (SRN, Elman, 1990). The state of an SRN at time step t is defined by a single vector h_t. Mathematically, the network can be described as follows:

$$h_t = \tanh(Wx_t + Uh_{t-1} + b) \tag{2.2}$$

The weight matrix W defines the transformation from the input to the hidden state, the weight matrix U the transformation of the current hidden state. The bias term b a fixed constant that is added at every step; it can be used by the model to change the midpoint of the nonlinear activation function tanh which wraps the entire processing step. W, U, and b are all learnable parameters of the model.

2.1.2 Long short-term memory networks

While SRNs can in theory remember information from many time-steps back, in practice they are often unable to learn dependencies that span over long ranges. This inability is often associated with how recurrent neural networks are trained: with gradient descent and *backpropagation through time* (Rumelhart et al., 1986). For a long-distance dependency to be learned, there should be a flow of gradient between the two dependents. In SRN models, this is often not the case, because the gradient vanishes very quickly with distance.

In 1997, Hochreiter and Schmidhuber proposed the long short-term memory (LSTM) network, which is specifically designed to circumvent this 'vanishing gradient effect'. The state of such an LSTM model at any time step t is defined by not one but two vectors: the hidden state h_t and the memory cell c_t.

The introduction of *gates* The information flow from the current to the next state of an LSTM model is modulated by three *gates*, that are computed by the model itself, based on the current input and the previous hidden state of the network:

$$f_t = \sigma(W_f x_t + V_f h_{t-1} + b_f) \tag{2.3}$$
$$i_t = \sigma(W_i x_t + V_i h_{t-1} + b_i) \tag{2.4}$$
$$o_t = \sigma(W_o x_t + V_o h_{t-1} + b_o) \tag{2.5}$$

These gates are called the *forget*, *input* and *output* gate, respectively, and can take values between 0 and 1.

Memory cell To compute the state of the memory cell c_t at the next time step, first a 'candidate memory cell' activation is computed:

$$\tilde{c}_t = \tanh(W_{\tilde{c}} x_t + V_{\tilde{c}} h_{t-1} + b_{\tilde{c}}) \tag{2.6}$$

The next memory cell activation c_t of the network is a weighted sum between this candidate activation and the previous memory cell state c_{t-1}:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{2.7}$$

The contribution of these two components is decided by the previously defined forget gate (Equation 2.3) and input gate (Equation 2.4). The forget gate determines how much of the previous hidden state is remembered. If the forget gate is completely *open* ($f_t = 1$), the entire content of the previous memory cell is carried on to the next time step; if it is closed ($f_t = 0$), all this content is forgotten.

The input gate instead modulates how much information flows in from the candidate cell state $\tilde{c}_t$, which contains information about the current input of the network. An open input gate lets all information of c_t pass to the next time step, a closed input gate instead ignores any activation flowing from the current input word. If the forget gate is completely closed and the input gate completely open, the computation step to compute c_t is identical to the computation of the activations in a simple recurrent network.

Hidden state The next hidden state h_t of the network is computed by multiplying the memory cell contents with the output gate of the network:

$$h_t = o_t \odot \tanh(c_t) \tag{2.8}$$

In LSTM models, the hidden state h_t is typically connected with an *output layer* or, in case of a multilayer model, propagates its activation forward to the next layer.

Long-distance dependencies The availability of gates allow the LSTM model to easily track long-distance dependencies. An individual unit (or group of units) can easily store information over long periods of time by opening its forget gate and closing its input gate.

2.1.3 Gated recurrent units

Chung et al. (2014) introduced a gated recurrent network that is simpler than the LSTM model. Their gated recurrent unit (GRU) does not have an explicit memory cell. Like the SRN, its state consists of only one hidden state h_t.

Reset gate Similar to an LSTM cell, a GRU computes a *candidate state* activation. In the GRU, this candidate state is itself modulated by a gate, called the reset gate r_t. This reset gate determines to what extent the next hidden state depends on the current input and previous hidden state:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.9}$$

$$\tilde{h}_t = \tanh(W x_t + r_t \odot U(h_{t-1}) + b) \tag{2.10}$$

If the reset gate is completely open ($r_t = 1$), all information of the previous hidden state is passed on to the next hidden state. When r_t is open, the candidate hidden state computation is identical to the hidden state computation of the SRN. If the reset gate is closed ($r_t = 0$), the hidden state activations are 'reset', and the candidate activations do not depend on the previous time step, but only on the current input case. In that case, the computation matches the computation for a feed-forward layer. The reset gate thus determines to what extent the computation of h_t is feed-forward or recurrent.

Forget gate To compute the next hidden activation h_t from the previous hidden activation h_{t-1} and the candidate activation $\tilde{h}_t$, the GRU uses a single 'update' gate. This gate decides the extrapolation factor between the previous state and the candidate state:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.11}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{2.12}$$

If the update gate is completely open ($z_t = 1$), the next hidden state is identical to the candidate hidden activation (if r_t in that time step was 1 as well, the processing step is identical to the processing step of an SRN). If the update gate is closed, no information from the input passes through, and the next hidden state is identical to the previous one.

2.1.4 LSTM vs GRU

As they have fewer gates and have a less complex structure, GRU models are computationally more efficient than LSTMs. While there are some papers that point out differences between the computational power of GRUs and LSTMs with finite-precision (e.g. Weiss et al., 2018), in practice their performance is often comparable. In the studies presented in the rest of this dissertation, GRU models sometimes outperformed LSTM models, while other times LSTM models showed more desirable behaviour. I am not aware of any work that elaborately compares GRU and LSTM models in different situations.

2.2 Recurrent architectures

When SRNs, LSTMs, and GRUs are used to perform a particular task, they are typically embedded in a larger architecture that contains one or more recurrent layers but usually also several other components. In this section, I describe two types of recurrent architectures, as well as several components that are commonly added to them.

2.2.1 Types of architectures

Recurrent architectures can be roughly divided into two categories. First, there are architectures that generate predictions at every time step. I will refer to this type of architectures with the term *immediate-prediction architectures*. Then, there are architectures that first encode an entire input sequence and then generate an output based on this encoding. They are often called *encoder-decoder* or *seq2seq* models.

Immediate-prediction architectures Immediate-prediction architectures are used for tasks that require immediate processing, such as language modelling or speech processing. Typically, in tasks requiring immediate-prediction architectures, the input space (consisting of the input words) is the same as the output space, although this is not a strict requirement. I will use immediate-prediction architectures in Chapter 5, 6 and 7.

Encoder-decoder models Encoder-decoder models are used for tasks that require the generation of an output based on an *entire* input sequence. Encoder-decoder architectures are used for tasks such as translation – in which case the *decoder* itself is also a sequential model, or to assign labels to entire sequences (e.g. sentiment classification). In the latter case, the decoder of the model may be as simple as a feed-forward layer, while in the former case, the decoder can also be a recurrent model. I will use encoder-decoder models with a sequential

decoder in Chapter 4 and 8), and encoder-decoder models with a simple simple feed forward layer as decoder in Chapter 3.

2.2.2 Word embeddings

Recurrent cells define a how sequences of continuous input vectors are transformed into sequences of continuous output vectors. However, the input sequences to recurrent models usually consist of discrete symbols. This discrepancy is solved by adding an *embedding* layer to the model, that transforms the discrete input tokens (represented by *one-hot vectors*) into continuous vectors. This layer is parametrised by a matrix whose values are learned along with the rest of the model's parameters. The number of columns of this matrix matches the number of tokens in the input space; the number of rows is equal to the desired *embedding size*. Each column represents the *embedding* of a particular input token.

2.2.3 Output layers

Like the inputs to recurrent models, commonly also the required outputs consist of discrete symbols or classes. Architectures thus require also a mechanism to map their continuous outputs back to a discrete space. To do so, models typically include an *output layer*, which contains a matrix that maps the output of the recurrent layer back to a vector with the correct number of classes (e.g. the number of tokens or classes in the output space). The result of this is a continuous vector with the right dimensions; this vector is then transformed into a probability distribution over the output words or classes by applying a *softmax* function to it. There are several methods to obtain a discrete prediction from this probability distribution, the most popular of which are *sampling* a class or token from the distribution, or simply using *argmax* to select the class or token with the highest probability.

2.2.4 Attention

A component that is very frequently added to encoder-decoder models is an *attention* mechanism (Bahdanau et al., 2015). The invention of this mechanism started from the observation that the decoder always uses the same fix-length vector to generate its output, irrespective of the length of the input. Bahdanau et al. (2015) suggest that this my result in too much inormation loss and propose to give the decoder model access to *all* encoder representations. To search these representations for parts that are relevant to the current prediction, the model is equipped with an attention mechanism. This mechanism is typically instantiated by an additional matrix, that is used to compute a *relevance score* to every encoder state. With those, a weighted average of those states is computed, which is then given as additional input to the decoder.

Relatively recently, a model was proposed that is based only on a multitude of such attention mechanism (Vaswani et al., 2017). This model, which was named *Transformer* by the authors, has achieved large successes on many NLP tasks. I will use this model as test-case in Chapter 4, but will not otherwise explore it, for reasons laid out before. I will use recurrent models with attention components in Chapter 8 and Chapter 4.

2.3 Approach 1: artificial data

I discussed three recurrent cells and several components that are frequently used along which such cells in recurrent architectures. In the remainder of this chapter, I discuss two types of approaches that investigate to what extent recurrent architectures can learn to process hierarchical structure, using either artificial or naturalistic data. In this section, I start with discussing approaches using artificial data, which can be divided into roughly two categories.

2.3.1 Formal grammars

In the earlier days of neural networks, a frequently wandered path to test neural networks was to evaluate whether they can approximate formal languages at different levels in the Chomsky hierarchy (CH). In such studies, a grammar is defined – often a context-free grammar with specific types of embedded structures – which is used to generate a corpus of sentences. An immediate-prediction architecture (see subsection 2.2.1) is then trained to predict the sentences in this corpus and evaluated on how well it does so. Not uncommonly, studies that follow this paradigm also include an analysis of the state space trajectories within the network.

Most studies considering formal languages have looked at SRNs. Some use grammars with constructions specifically motivated by natural language (e.g. Elman, 1991; Christiansen and Chater, 1999); others instead focus on formal languages exemplary for particular classes in the Chomsky Hierarchy, such as $a^n b^n$ (e.g. Rodriguez, 2001; Wiles and Elman, 1995; Rodriguez et al., 1999; Batali, 1994). Some studies have considered also gated recurrent neural networks (e.g. Gers and Schmidhuber, 2001; Weiss et al., 2018).

2.3.2 Compositional signal-meaning mappings

A newly popular approach with artificial languages focuses on the extent to which neural networks can represent compositional mappings from signals to meanings. As such tasks require to first read a signal and then produce the correct meaning for this signal, they require some kind of encoder-decoder model (see subsection 2.2.1).

Recently, several studies appeared that focused explicitly on the ability of networks to infer compositional structure.

SCAN Quite a few recent studies consider the *SCAN* language. This language describes a simple navigation task with input sentences like *jump thrice* or *run around left*, which have to be translated into a series of actions (for instance *JUMP JUMP JUMP*). The *SCAN* language was proposed by Lake and Baroni (2018) – who used it to illustrate particular shortcomings of neural networks when it comes to compositional rule learning – and was later used in several follow-up studies (Loula et al., 2018; Bastings et al., 2018; Korrel et al., 2019; Dessì and Baroni, 2019).

The lookup table task

The study I selected to discuss in this section assumes an even more minimal setup than the *SCAN* language. In this task, called the *lookup table task*, the atomic operations that form the basis of more complex compositions are not rules, but simple lookup tables (Liška et al., 2018). The task then consists in computing the outcome of a composition of one or more lookup tables applied to an input.

The lookup tables are defined as bijective mappings from the domain of all binary strings of length L onto itself. For instance, a lookup table `t1` may be defined as:

$$\begin{aligned} &\texttt{t1}\ 00 \rightarrow 01 \\ &\texttt{t1}\ 01 \rightarrow 11 \\ &\texttt{t1}\ 10 \rightarrow 10 \\ &\texttt{t1}\ 11 \rightarrow 00 \end{aligned}$$

When receiving an input sequence like `t1 t2 00`, the model should consecutively apply its tables to the binary input string: in the current example, it should first apply `t2` to `00` and then apply `t1` to the outcome of this computation.

Lookup tables as a navigation task Like SCAN, also the lookup table task can be interpreted as a navigation task, where the tables are represented by *paths* and the binary strings by *locations*. In Figure 2.1, I sketched an example that matches with a setup with two lookup tables and four locations. If the hotel, conference and park are location `00`, `01` and `11`, respectively, the sequence `t1 t2 00` would be translated as *From the hotel, take path* `t2`*, then path* `t1`. The correct outcome would be the park, which is where one ends up following these paths.

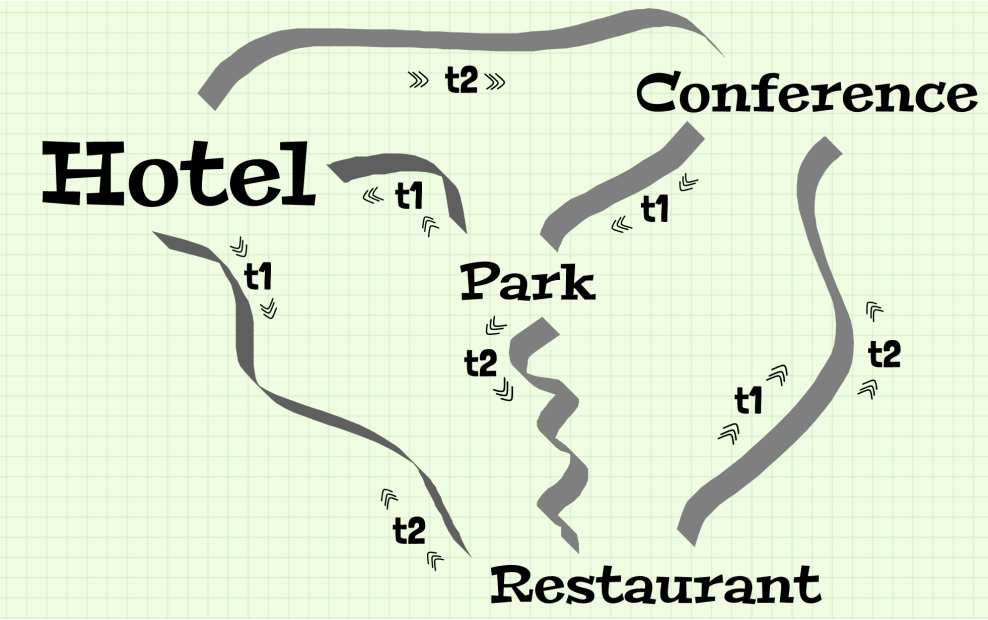


Figure 2.1: Visualisation of the lookup table task as a navigation task. The tables `t1` and `t2` are seen as paths, the binary strings `00`, `01`, `10` and `11` as locations.

Compositionality From the example above it is easy to see what kind of compositional behaviour is desired: if a human knows how to walk from their hotel to the conference site via path `t2`, and from the conference to a park via route `t1`, they won't have trouble going from their hotel to the park by composing these two routes, even if they are unfamiliar in the city. Conversely, if a human knows how to walk from the hotel to the park via the conference site, we'd expect them to also be able to reach the park from the conference site if they arrived at the conference from a different location, such as the restaurant where they had lunch.

Results Liška et al. (2018) evaluate if LSTM-based encoder-decoder models behave in a the desired compositional way, by testing if they are able to generalise to path combinations from unseen locations (e.g. a model may have seen `t1 t2 00` and `t1 t2 01` during training and is then tested on `t1 t2 11`). To encourage the models to interpret the inputs compositionally, they ask the model not only to output the *final* location (output), but also the intermediate ones, including the start location. For instance, in the previous example, the correct answer would not just be "park", but the entire list of locations that is visited: hotel, conference, park.

Liška et al. (2018) train 50K LSTM models and report that only very few of them find a solution that generalises in a compositional way: 2% of the models have a test accuracy above 80%, 0.75% of the runs result in models that have

an accuracy higher than 90%. These findings suggest that LSTM models thus can represent the desired compositional solutions, but – at least on this task – do not frequently converge to them when they are trained with the current learning regimes.

Both SCAN and the lookup-table task implement some notion of what it means to be compositional. In part one of this dissertation (Chapter 3 and Chapter 4), I consider two other artificial languages, that focus on different parts of compositionality, hierarchy and structure, and I evaluate to what extent the previously described findings also holds for those languages. In Chapter 8, I reconsider the lookup table task and investigate if a change in learning signal can elicit a change in the type of solution found by recurrent models.

2.4 Approach 2: naturalistic data

The approaches I described in the previous section considered models trained on data that has structure explicitly built in. Another way of investigating if neural networks can learn the types of structures required to adequately model natural language is to instead consider neural networks that are trained on *naturalistic data*, and probe their structural abilities. Most dominantly, such studies are done with models trained to do machine translation (i.a. Shi et al., 2016; Belinkov et al., 2017b; Blevins et al., 2018; Raganato and Tiedemann, 2018; Tenney et al., 2019b; Vig and Belinkov, 2019) or language modelling (i.a. Linzen et al., 2016; Wilcox et al., 2018; Jumelet and Hupkes, 2018; Giulianelli et al., 2018; Gulordava et al., 2018). In general, such studies are relatively positive about the amount of non-trivial structure that is picked up by (recurrent or non-recurrent) neural networks. In this section, I review some of the main evidence for the ability of recurrent language models to capture structure, upon which I build in part two of this dissertation (Chapter 5, 6 and 7).

2.4.1 Language models as psycholinguistic subjects

The first study to probe the ability of neural language models to model hierarchical structure was presented by Linzen et al. (2016), who proposed to assess this by evaluating whether models can correctly process long-distance subject-verb agreement, using a paradigm well known from psycholinguistics.

The premise of their test is that in English the form of a third-person present tense verb depends on the *number* of its syntactic subject. For instance, if the syntactic subject of a sentence is *paper*, the corresponding verb should be singular (The paper *is*), whereas the plural subject *papers* would go with a plural verb (The papers *are*). In the given example, the subject and verb are adjacent, but, in English, subjects and verbs can be separated by an arbitrary number of tokens. Such tokens can potentially include intervening *attractor* nouns which

are irrelevant to the form of the main verb of the sentence, or even embedded subject-verb pairs. Since identifying all subject-verb pairs in a sentence within an arbitrarily large window of tokens requires some sort of syntactic analysis, subject-verb agreement is a phenomenon that is commonly regarded as evidence for the presence of hierarchical structure in natural language and can – by extension – thus also be used to probe the abilities of neural networks to process hierarchical structure.

2.4.2 The number agreement task

To test whether a model can correctly process subject-verb relationships, Linzen et al. (2016) designed a task in which a model needs to predict the correct verb number given a sentence prefix. For instance, building on top of the previous example, a model might be given the sentence prefix

> *The paper that was written by several authors from the University of Amsterdam and one from the University of Tilburg ...*

and then has to predict that the correct verb number following this prefix is singular. Linzen et al. (2016) created a data set containing circa 1.35 million number prediction problems with present tense verbs, drawn from an automatically parsed corpus with Wikipedia data. The sentences in this corpus differ with respect to the number of words between the subject and the verb (later I will refer to this distance with the term *context size*); the number of intervening nouns and agreement attractors; and whether the intervening material contains a relative clause.

2.4.3 Supervised number prediction

In their 2016 article, Linzen et al. primarily focus on LSTMs that are trained directly on predicting the correct number of the main verb. They train an LSTM with 50 hidden units that – when trained on 9% of the corpus and validated on 1% – has a very low error (0.83%) on the remaining 90% of the corpus, which does not increase considerably with the length of the intervening sentential material. They conclude that with explicit supervision, LSTM models can very well learn the number agreement task. The most difficult case is the one with multiple attractors between the subject and the verb: for a sentence with four attractors with different syntactic numbers, the error rate was 17.6%. Linzen et al. compare their results with a baseline that only receives the nouns of the sentence as input and show that this model performs significantly worse in all cases. Their overall conclusion is that LSTM models can learn the number-prediction task very well when given explicit supervision. Such models are thus able to approximate structure-sensitive dependencies to some extent, although they still struggle with more difficult cases.

2.4.4 SV agreement in language models

Using the same task, Linzen et al. (2016) also evaluate how well neural language models (thus trained only with a language modelling objective) can perform the number prediction task. As in the previous case, the model is given the sentence up to the main verb. Then, to evaluate the model's performance for that particular sentence, they compare the probabilities that the model assigns to the verb with the correct number and the wrong number. The overall accuracy of the corpus is determined by how often the network assigns a higher probability to the correct verb form.[1]

Linzen et al. (2016) find that an LSTM language model with 50 hidden units makes around eight times as many errors as the LSTM trained with explicit supervision on the task and also does substantially worse than the noun-only baselines. For difficult dependencies, the network performs worse than chance, but slightly better than the noun-only baselines, suggesting that the networks are confused by the attractors. To exclude the possibility that the results are due to lack of training data or the model's size, Linzen et al. (2016) repeat their study using a large publically available language model (Jozefowicz et al., 2016) that is trained on more data, has a larger vocabulary and two layers with 8192 units each. While – in some cases – this language model outperformed the smaller language model trained by Linzen et al., it still performed poorly with respect to previously mention baselines, exhibiting a strong increase in error when even a single attractor was present. The authors conclude that a language modelling objective is not a sufficient signal to induce the syntactic knowledge that is required to process long-distance dependencies.

2.4.5 SV agreement in language models, revisited

Later, the study of Linzen et al. was repeated and extended by a different – not entirely disjoint – research group, who tested a number of different long-distance dependencies for English, Italian, Hebrew and Russian (Gulordava et al., 2018). The results do not match the earlier findings of Linzen et al. (2016): Gulordava et al. (2018) find that an LSTM language model can solve the subject-verb agreement problem well, comparably to the model supervised on the task that was presented by Linzen et al. Furthermore, they find that the model still performs reasonably well when the words in the sentence are replaced by syntactically nonsensical words, and the model can thus not rely on semantic clues.

The results of Gulordava et al. (2018) convincingly demonstrate that models

[1] Linzen et al. noted that another possibility would be to aggregate the probability mass over all plural and singular verbs. In a project I did on this topic with 3 master students we considered this method for computing the accuracy on the number prediction task. We did not find strong differences with respect to only comparing the probabilities of the two verb forms of the verb that originally occurred in the corpus.

trained with a language modelling objective can in fact learn to represent non-trivial grammatical structure, even when they can not rely on semantic cues. This exciting finding forms the premise of part two of this dissertation (Chapter 5, 6 and 7), in which I replicate their results and investigate how recurrent language models model these relationships.

2.5 Summary

In this chapter I discussed some background concepts important to appreciate the content of this thesis. In particular, I described three recurrent cells – SRNs, GRUs and LSTMs. I provided both a mathematical and an intuitive description of their main components, and explained how they can be embedded in larger architectures. All studies in this dissertation consider one or more of these three cells types, and I will sometimes refer back to the explanations in this chapter when a more detailed understanding of their inner dynamics is required.

I also described two types of approaches to investigate the extent to which neural networks can understand structure: using either artificial or naturalistic data. For both these approaches, I gave several examples and then selected one specific study to discuss in more detail.

The multitude of studies on the topic illustrates the importance of evaluating the behaviour of neural models, but the variety in conclusions drawn from those studies also exemplifies that whether neural networks can in fact learn to process hierarchical compositional structure is still an open question. To better understand whether and under which circumstances they learn interesting types of structure more studies are required, as well as better interpretability techniques to understand the meaning of the results. This observation forms the starting point of this work, in which I address both these issues.

The remainder of this dissertation is divided in three parts. In part one, I follow the first described controlled approach and use artificial signal-to-meaning mappings to investigate how neural networks can learn to process hierarchical compositional languages. To do so, I do not only look at their *behaviour* under different circumstances and on different types of stimuli, but I also propose techniques to investigate their hidden states (Chapter 3). Further, in Chapter 4, I take a careful look at what it means and implies that a neural network *is able to process compositional structure*. I consider several potential interpretations that I formalise into concrete tests, which I apply to different architectures.

In the second part of this dissertation, I use setups with naturalistic data, that directly build upon the studies described in Section 2.4. These studies illustrated that recurrent neural language models can learn to process long-distance subject verb agreement relationships, a very promising result regarding their potential usefulness as explanatory models. However, these studies do not address *how* these networks are implementing these feats, which is what I investigate in Chapter 5, 6

and 7. Also these chapters contain a mix of behavioural experiments intertwined with the development of interpretability techniques.

In part three of this dissertation, I take a slightly different approach. I consider the *lookup-table* task described in Section 2.3, but I do not only *observe*, but instead try to change the types of solutions found with an additional feedback signal to the model. As the capstone of this work, I then use several of the techniques proposed earlier in the dissertation to investigate the difference between the solutions learned with and without the additional learning signal.

Part One

Artificial languages

In the first part of this dissertation, I consider *artificial languages*. While ultimately I am interested in the structures occurring in natural language, human language is a complex phenomenon, of which many facets are still not well-understood. Both symbolic structure and hierarchical compositionality are widely considered to play an important role, but it is still not known what type of structural or compositional skills are required to successfully model tasks involving natural language.

As a consequence, insight in the structure-processing capabilities of neural networks is not easily acquired by studying natural language with all its complexities directly. If it cannot be excluded that successful heuristics or syntax-insensitive approximations exist, a positive result cannot be taken as evidence that a model does in fact understand structure. Similarly, a negative result does not necessarily indicate that a particular type of model cannot, it merely indicates that this exact model instance did not capture it in this exact case.

Therefore, I first consider artificial languages to analyse the extent to which neural networks can model languages with a hierarchical compositional semantics. Artificial languages provide a clean setup in which different facets of processing can be considered in isolation. In Chapter 3, I use this to focus specifically on the ability of different types of recurrent architectures to process *hierarchy*. Because both the training data and the test data are highly controlled, the impact of different data properties can be easily tested. In Chapter 4, I exploit this property by generating several data sets with different properties that consider different aspects of compositional processing.

Additionally, well-understood data make it easier to formulate *hypotheses* concerning potential strategies that models may be pursuing. This, in turn, facilitates the development and assessment of different interpretation techniques for the models, as we will see in Chapter 3.

Chapter 3

Diagnostic classification and the arithmetic language

This chapter is the first of two chapters in which I use artificial data to analyse the extent to which neural networks can model languages with a hierarchical compositional semantics. I present a study that focuses explicitly on the aspect of hierarchy and the internal strategies implemented by different types of networks. I also introduce the notion of *diagnostic classification*, which will be a recurring technique in this dissertation.[1] In the next chapter, I take a broader perspective on hierarchical compositionality, and dive deeper into what potential aspects of it might be measurable in a model.

Outline In Section 3.1, I first present the artificial language that I use for my study, the *arithmetic language* which consists of spelled out, nested, arithmetic expressions. I discuss the language itself, but also potential formal *strategies* that could be pursued to compute the meanings of the sentences and the predictions

[1] This chapter is based on my contributions to the work described in:

> Dieuwke Hupkes, Sara Veldhoen, and Willem Zuidema. Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure. *Journal of Artificial Intelligence Research*, 61:907–926, 2018b

This paper was itself based on:

> Dieuwke Hupkes and Willem Zuidema. Diagnostic classification and symbolic guidance to understand and improve recurrent neural networks. In *Proceedings of Neural Information Processing Systems – Workshop track*, 2017

Upon invitation, a short version of this paper was also presented at the journal track of IJCAI2018:

> Dieuwke Hupkes and Willem Zuidema. Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure (extended abstract). In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5617–5621, 7 2018.

that follow from them. In Section 3.2 I first consider whether three different recurrent architectures can correctly compute the meaning of the sentences of this language. The most important parts of this chapter then follow in Section 3.3 - 3.6, in which I describe how I applied a variety of analysis techniques to capture the internal dynamics of the networks and introduce diagnostic classification. In Section 5.7, I provide a brief conclusion of the study.

3.1 Arithmetic language

The arithmetic language used for the study described in this chapter was introduced in the master thesis of Sara Veldhoen (2015). The vocabulary of this language consists of words for all integers in the range $\{-10, \ldots, 10\}$, the operators `plus` and `minus` and the brackets `(` and `)`. The phrases in the language – i.e. sequences of words – comprise all grammatically correct, fully bracketed arithmetic expressions that can be formed with these symbols. The (compositional) meaning of an expression is the solution of the arithmetic expression that it represents. For instance, the meaning of the phrase `( ten minus ( five plus three ))` is 2.

Throughout this chapter, I often abbreviate the full forms such as `left_bracket five plus three right_bracket` as `( 5 + 3 )`. I refer to expressions and sets of expressions by using the number of numeral words they contain (see Figure 3.1 for a formal definition). For instance, **L5** refers to all expressions with exactly 5 numerals and $\mathbf{l_5}$ is an expression belonging to **L5**. Table 3.1 contains some example sentences of the arithmetic language, along with the name of the subset they may appear in.

	Form	*Meaning*
vocab	{ `-ten, -nine, ..., nine, ten, plus, minus, left_bracket, right_bracket`}	{-10, -9, …, 9, 10, +, -, (,)}
L1	{`-ten, -nine, ..., nine, ten`}	{-10, -9, …, 9, 10}
Lk	$\{$ `(` $\mathbf{l_m}$ `op` $\mathbf{l_n}$ `)` $\mid \mathbf{l_m} \in \mathbf{Lm}, \mathbf{l_n} \in \mathbf{Ln},$ `op` $\in \{$`plus`, `minus`$\}$, $m + n = k\}$	$\langle l_m \rangle \langle$ `op` $\rangle \langle l_m \rangle$

Figure 3.1: Formal description of the sentences s in the arithmetic language and their meanings $\langle s \rangle$.

3.1.1 Symbolic strategies

The symbolic nature of the arithmetic language allows us to formulate strategies to compute the meaning of expressions, which can be used to aid the analysis of the dynamics of the internal dynamics of a network while it processes sentences.

L1	`one, -three, nine`
L2	`( five plus three )`
L3	`( ( two minus -three ) minus six ), ( two minus ( -three minus six ) )`
L4	`( ( ( -two minus seven ) plus eight ) plus -ten)`

Table 3.1: Sentences from different subsets of the arithmetic language. Both numerals, operators, and brackets are treated as words; words are represented by n-dimensional numerical vectors.

Before moving on to the description of our models and experiments, I will describe two possible strategies to incrementally solve arithmetic expressions.

```
result_stack = [], mode_stack = []
result = 0, mode = +
for symbol in expression do
    if symbol == '(' then
        mode_stack.append(mode)
        result_stack.append(result)
        result = 0; mode = +
    else if symbol == ')' then
        mode = mode_stack.pop()
        prev_result =
         result_stack.pop()
        result = apply(mode,
         prev_result, result)
    else if symbol == '+' then
        mode = +
    else if symbol == '-' then
        mode = -
    else
        result = apply(mode,
         result, symbol)
end
return result
```

(a) Recursive strategy

```
mode_stack = []
result = 0, mode = +
for symbol in expression do
    if symbol == '(' then
        mode_stack.append(mode)
    else if symbol == ')' then
        mode = mode_stack.pop()
    else if symbol == '+' then
        pass
    else if symbol == '-' then
        if mode == - then
            mode = +
        else
            mode = -
    else
        result = apply(mode,
         result, symbol)
end
return result
```

(b) Cumulative strategy

Figure 3.2: Different symbolic strategies for incrementally solving arithmetic expressions. The function `apply`(*mode*, *result*, *symbol*) applies the operator specified by *mode* $(+, -)$ to the two numbers specified by *result* and *symbol*.

Recursive strategy

Perhaps the most obvious candidate for a symbolic strategy to compute the meaning of an arithmetic expression involves traversing through the expression,

computing the outcome of all subtrees, and combining them to compute the outcome of the full tree . To do this in an incremental fashion, the intermediate result of the computation of the current subtree should be pushed onto a stack – the `result_stack` – whenever a new, smaller subtree begins. At that point, also the operator that will later be used to integrate the outcome of the newly started subtree with its parent should be stored on a stack. Because this stack determines whether the procedure is in *additive* or *subtractive mode*, I call it the `mode_stack`. Figure 3.2a contains a procedural description of this strategy; For a worked-out example indicating the stack contents at different time steps, see the upper part of Figure 3.3.

Cumulative strategy

Alternatively, the meaning of a sentence from the arithmetic language can be computed by accumulating the numbers immediately at the moment they are encountered (see Figure 3.2b and the bottom part of Figure 3.3). In this case, a prediction of the solution of the expression is maintained at any point during the computation. Consequently, this cumulative strategy does not require a stack with previous results, but it does require keeping track of previously seen operators to decide whether the next number should be added or subtracted when a bracket closes (in Figure 3.2b captured by the variable `mode`).

3.1.2 Predictions following from strategies

As illustrated in Figure 3.3, the cumulative and recursive strategies do not only differ with respect to the computations they are executing, but also require different memory contents. Both the cumulative and the recursive strategy require a stack to store encountered operators. Consider for instance computing the outcomes of the expressions `( 8 - ( ( 5 - 7 ) - 2 ) )` and `( 8 + ( ( 5 - 7 ) - 2 )`, for which information about the previous series of operators is required to understand whether the `2` should be subtracted or added to the subtotal. In addition, the recursive strategy requires storage of the previously computed outcomes of subtrees. These differences result in different predictions about the sensitivity of the network to noise on the stack, implementation of the operator or depth of an expression, and hence in different predictions about the difficulty of processing certain structures under memory limitations and noise.

3.2 Can RNNs learn the arithmetic language?

I first assess to what extent (gated) recurrent neural networks can learn to correctly process the deep hierarchical structure of sentences from the arithmetic language, without being provided any explicit feedback on this structure. For this study,

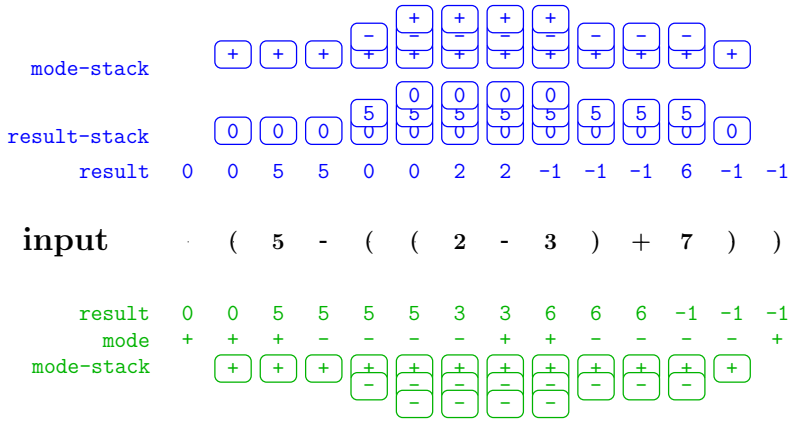


Figure 3.3: Different strategies to compute the meaning of a sentence of the arithmetic language. Top (in blue): `mode-stack`, `result-stack`, and current `mode` of the recursive strategy; bottom (in green): cumulative `result`, `mode` and `mode-stack` of the cumulative strategy.

I use an encoder-decoder architecture with a non-recurrent decoder. For the encoder, I consider all three recurrent cells described in the previous chapter: the SRN (Section 2.1.1), the LSTM (Section 2.1.2) and the GRU (Section 2.1.3). Below, I describe the procedures I used for training and testing.

3.2.1 Training

I train 20 GRU, SRN and LSTM models on a randomly sampled sets of expressions from **L1**, **L2**, **L4**, **L5** and **L7** (3000 expressions from each subset) with various syntactic structures.[2] The composition of lengths of the training set remains the same across all experiments, but which exact expressions of these five lengths are chosen depends on the random seed during initialisation of the experiment. For both models, I use an input embedding size of 2, and a hidden layer size of 15. The word embeddings are randomly initialised within a small range, and are trained along with the rest of the model's parameters. The models are trained on an error signal backpropagated from a simple linear perceptron predicting the real-valued solution of the expressions from the final hidden state, using Adam Kingma and Ba (2015) as optimiser (learning rate=0.001, β_1=0.9, β_2=0.999, ϵ=1e-08, decay=0.0), minibatches of size 24 and mean squared error as loss function. The models thus have implicit access to the syntactic structure of sentences via the brackets, which are presented to the network as words, but are not provided with explicit feedback on this structure.

[2]For the implementation I use the Python library Keras Chollet et al. (2015) with Theano Theano Development Team (2016) as backend. The Python package containing the source code of this project can be found at `https://github.com/dieuwkehupkes/processing_arithmetics`.

3.2.2 Evaluation

To evaluate the resulting models, I create a test set containing a large sample of expressions from **L1**, **L2**, ..., **L9**. Except for **L1** and **L2**, all these test sets contain expressions that were not seen during training; **L3**, **L6**, **L8**, and **L9** also contain longer and shorter unseen *structures*. A summary of the results is plotted in Figure 3.4.

SRNs Of the 20 trained SRN models, three did not learn to capture any structural knowledge, reflected by a high error for short (but unseen) sentences with three numerals (**L3**). Also the remaining 17 models have a high error compared to the gated models: the best SRN model has an error high above the average of the gated models. I have not thoroughly investigated the extent to which the solutions learned by these models incorporate the syntactic structure of the sentences. They have perhaps learned something, but from the experiments conducted for this chapter it was not clear what.

Gated models The GRU and LSTM models, on the other hand, show a more convincing ability to generalise. Their mean absolute error slowly increases with the length of the sentences. Contrary to the SRN models, there is no error increase visible for sentence lengths that were not included in the training set. These results suggest that the gated recurrent models – at least to some extent – learn to process the hierarchical structures of the expressions in the arithmetics language. In the rest of his chapter, I focus on analysing the dynamics of the network while processing such expressions.

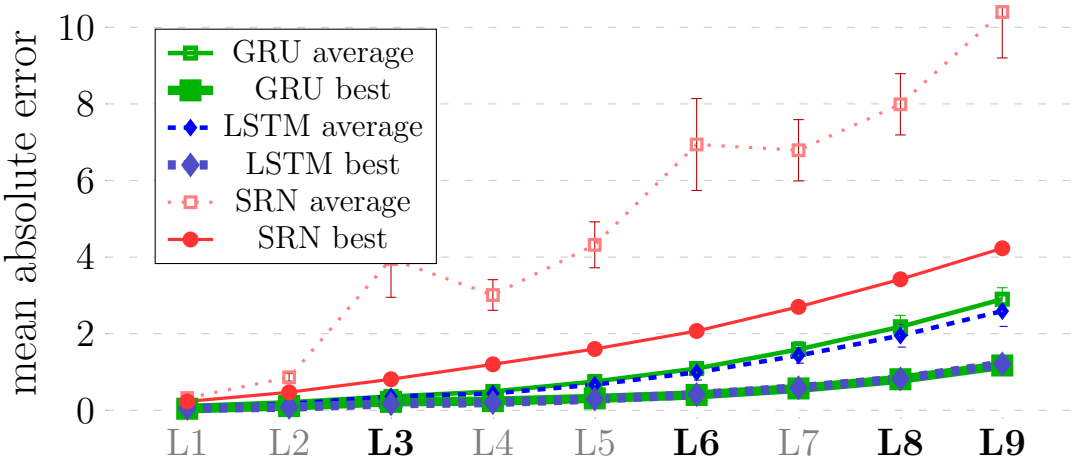


Figure 3.4: Average and best mean absolute error for 20 GRU, LSTM and SRN model instances. Error bars indicate standard error. Sentences of lengths that were not included in the training set are bold-faced.

3.3 Interpreting hidden activations

While there are quite some interesting studies focussing on investigating the internal dynamics of neural networks, at the moment that I am this dissertation this field is still very much in development. In this section, I briefly discuss some of the previously proposed methods to inspect the internal behaviour of artificial neural networks as well as their shortcomings. In the next section, I propose a new method that addresses some of their shortcomings.

3.3.1 Individual cell dynamics

A potential strategy to gain a better understanding of the internal dynamics of a recurrent neural network is to consider the interpretability of individual cells. By plotting individual cell activations, Karpathy et al. (2015) discovered several interpretable cells in a character-based neural language model, including cells that keep track of the scope of quotes and cells that represent the length of the sentence. We here apply this approach to the best-performing GRU model from our experiments.

In Figure 3.5 I plot the hidden layer activations while the network is processing three different input sequences. Next to the activations, I show the input sequence (starting from the top) as well as the `mode` of operation of the cumulative strategy in Figure 3.2 (i.e. `+` or `-`) at each point in time. Under the hidden layer activations, I also show the weights of the output layer reading out the meaning of the sentence, using the same colour scale as for the hidden layers (plotted at the right of the figure).

From this picture, several interesting observations can be made. Both the first cell (the first column from the left in all three graphs, black arrow) and the twelfth cell (the fourth column from the right, black arrow) show a sharp change in activation whenever the mode changes from `+` to `-`. The last layer of the network (bottom of the figure) indicates that the leftmost cell is negatively influencing the prediction of the solution of the expression, while the twelfth cell is hardly involved. The very last cell (red arrow) seems to respond to a minus in the input but appears to have also other functions. The tenth cell (blue arrow) could potentially be representing the scope of a minus.

Studying hidden layer activations is an interesting puzzle and can – especially for relatively low dimensional networks such as ours – give pointers to which aspects should be studied in more depth. However, it is hard to draw definite (and quantitative) conclusions, and the usefulness of the method decreases with the dimensionality of the networks.

3.3.2 Gate activation statistics

Karpathy et al. (2015), in the same paper, also study gate activations, focusing in

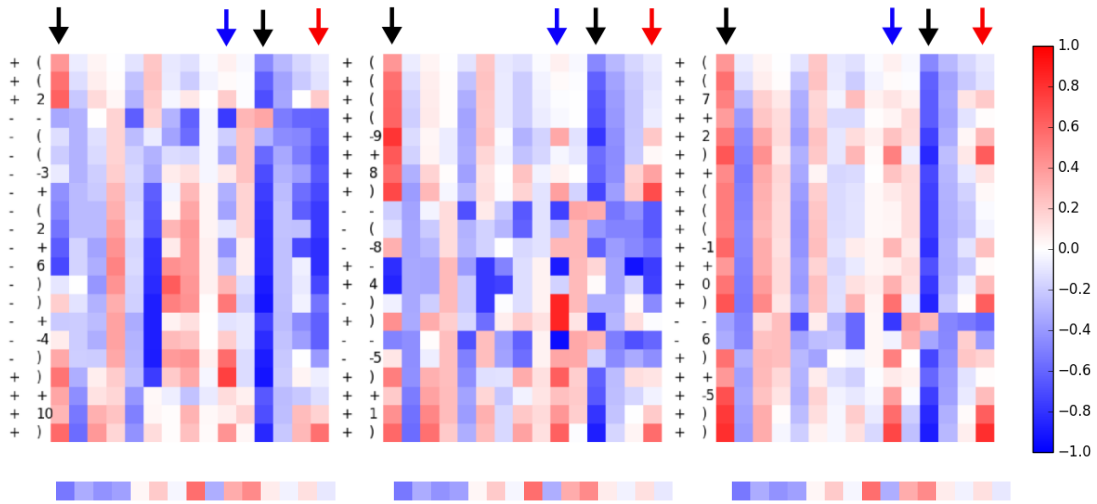
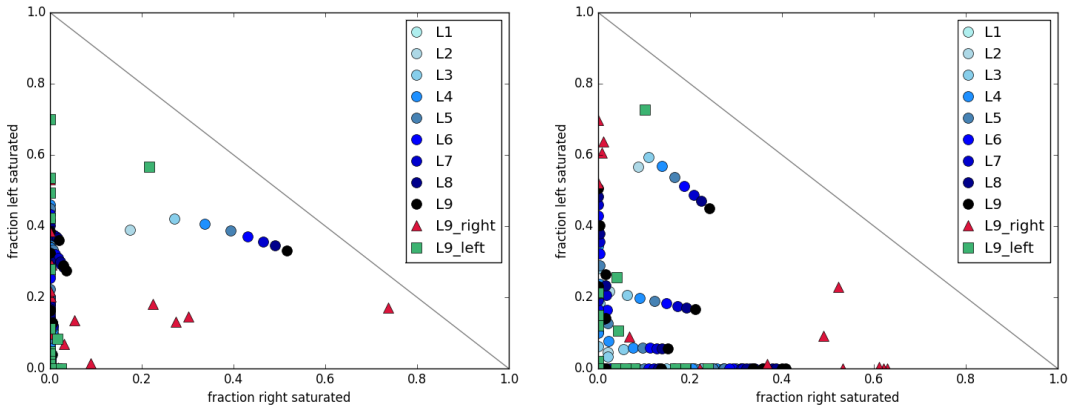


Figure 3.5: Hidden layer activations of a trained GRU network while processing different sequences. The input labels, along with the `mode` (addition/subtraction) at every point in time are printed left of the activation values. The activation values of cell 1 and 12 (black arrows) seem to be correlated with the `mode`, but it is is not easy to determine whether this value is in fact computed by the network. The 10th cell could be keeping track of the scope of a minus.

particular on the fraction of time that gates spend being left- or right-saturated (activations less 0.1 than or more than 0.9, respectively). Such results are not easy to interpret but might indicate what roles specific cells play in the information processing a network performs. For example, a cell with a right-saturated update gate remembers its previous activation, whereas a cell with a left-saturated update gate and a left-saturated reset gate ignores all previous activations and bases its value only on the current input. Are the cells mostly acting as a memory, in a feed-forward fashion, or as standard recurrent cells without any additional form of memory?

In Figure 3.6 I plot the left- and right-saturation statistics for a GRU network for different lengths of expressions. As the dimensionality of our network is considerably lower than that of the network considered by Karpathy et al. (2015), we can easily visualise the gate saturation values for different sentence lengths in the same plot. We observe that most update gates are either on the x- or y-axis. Some cells (at the right of the picture) act as a memory a substantial amount of the time. A few cells show an interesting context-dependency, spending an increasing fraction of the time being right-saturated. The reset gate saturation values show that several cells spend a considerable amount of time in 'feedforward-mode' (indicated by a high fraction of left-saturation). For three update cells and one reset gate cell, the activation appears to be dependent on the length of the expression.



(a) Reset gate **r** (Equation 4) (b) Update gate **z** (Equation 6)

Figure 3.6: Gate activation statistics of the update and reset gate of a GRU model while processing sentences from different lengths. Following Karpathy et al. (2015), each circle represents a gate in the GRU. Its position is determined by the fraction of the time it is left- (activation value less than 0.1) or right-saturated (activation value more than 0.9).

3.4 Diagnostic classification

As mentioned earlier in this work, the development of interpretability techniques constitutes an important contribution of this dissertation. In the previous section, I discussed some existing methods, that are primarily based on visual inspection. While such methods might in some cases provide interesting hypotheses, they also illustrate the need for further development of interpretability methods: the potential conclusions that can be drawn from visual inspection concern only small parts of the network's overall behaviour and are often qualitative rather than quantitative, as manually analysing the behaviour of cells over a large number of examples is infeasible. Additionally, such visualisation methods are restricted to finding functions or features that are encoded by one cell, while being insensitive to operations distributed over multiple cells, or cells that encode multiple features at the same time. Disentangling the behaviour of networks through visual inspection of activations is searching for a needle in a haystack.

I therefore propose to use an alternative approach, that we call *diagnostic classification*.[3] Diagnostic classification is based on the idea that if a sequential model is computing certain information, or merely keeping track of it, it should be possible to extract this information from its internal state space. To test whether a network is computing or representing a certain variable or feature,

[3]The name *diagnostic classification* was a result of a brainstorm session with Willem Zuidema and Sara Veldhoen, both co-authors on the article where we first proposed to use this technique (Veldhoen et al., 2016).

I determine the sequence of values that this variable or feature should take at each step while processing a sentence. I then train an additional classifier – a *diagnostic classifier* – to predict this sequence of variable values (representing our hypothesis) from the sequence of hidden representations a trained network goes through while processing the input sentence. If the sequence of values can be predicted with a high accuracy by the diagnostic classifier, this indicates that the hypothesised information is indeed computed by the network. Conversely, a low accuracy suggests that this information is not represented in the hidden state.

Diagnostic classification is a generic method that addresses most of the shortcomings we listed for existing methods. It can be used to quantitatively test hypotheses about neural networks that range from very simple to fully-fledged (symbolic) strategy descriptions. For instance, diagnostic classifiers can be used to test whether a network has an internal representation of certain features of its input. Say that one wants to evaluate whether a network is keeping track of the length of a sentence, another example taken from Karpathy et al. (2015), one can simply train a diagnostic classifier to predict the value of this variable at each point in time from the hidden state of the network while processing a corpus of sentences. The accuracy of this diagnostic classifier will be high not only if there is a single cell acting as a length counter, but also when multiple cells are together encoding this information. Furthermore, the accuracy with which this classifier can predict the sentence length from the sequence of hidden states gives a quantitative measure of how well this information is kept track of. Analysing the accuracy of the classifier in more detail (i.e. inspecting at which points it fails to correctly predict the sentence length) or looking at its weights can provide more insight in what the network is doing.

In a similar fashion, diagnostic classifiers can be used to probe the strategy networks could be implementing on an algorithmic level (Marr, 1982), provided such strategies can be translated into sequences of targets for each time step. The cumulative and recursive strategy I defined in Figure 3.2b and Figure 3.2a, respectively, result in very different predictions about the intermediate results stored (and computed) during the processing of a sequence. For instance, after seeing the word `three` in the sequence `( five minus ( ( two mines three ) plus seven ) )`, the recursive strategy should have a representation of the value within the current brackets (which is -1), whereas the cumulative strategy should maintain a representation of the value of the expression up to that point (6).

3.5 Cumulative or recursive?

I now use diagnostic classifiers to attempt to understand how the models I trained are computing the meaning of arithmetic expressions. In particular, I will focus on their algorithmic strategy, using the cumulative and recursive strategy (Figure 3.2b and 3.2a) as working hypotheses. To test whether the network is

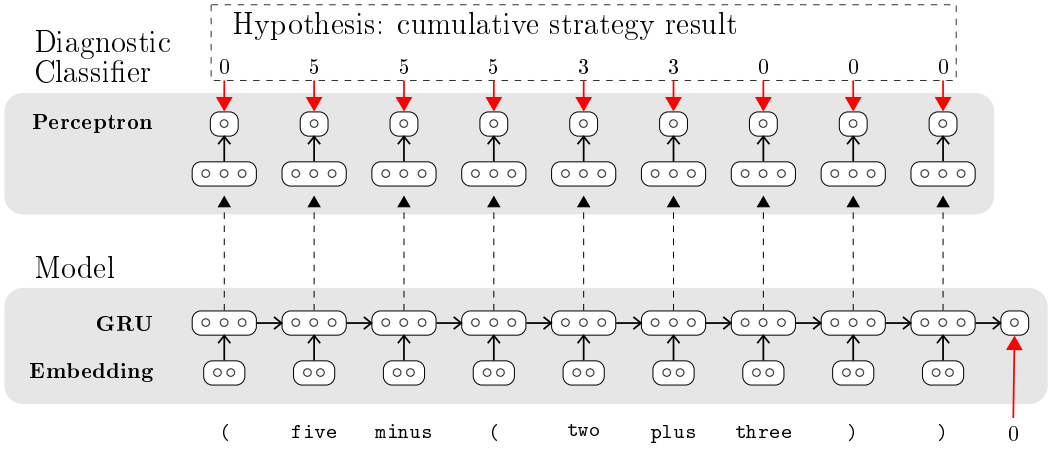


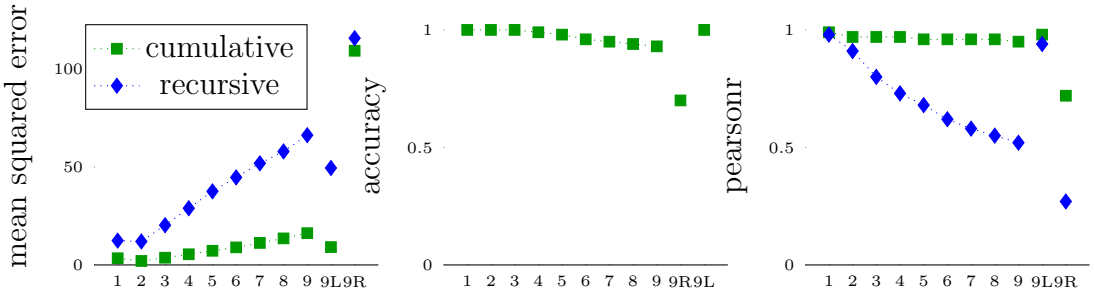
Figure 3.7: Testing a single hypothesis with a diagnostic classifier.

following either one of these strategies, I train diagnostic classifiers to predict the sequences of intermediate results of both these strategies, as well as the variable `mode` used by the cumulative strategy to determine whether the next number should be added or subtracted. As the diagnostic model should merely read out whether certain information is present in the hidden representations rather than perform complex computations itself, we use a simple linear model as diagnostic classifier. Figure 3.7 shows the setup for training a diagnostic classifier to predict the intermediate results for the cumulative strategy.

3.5.1 Diagnostic accuracies

We find that the values required for the cumulative strategy (`mode` and `result`) can be more accurately predicted than the intermediate recursive strategy values (see Figure 3.8a and 3.8b). From these findings, it appears unlikely that the network implements a fully recursive strategy that employs a number stack of intermediate results.

For the cumulative strategy, the predictions are generally accurate, even for longer sentences. The same is true for the `mode` of subtraction of the cumulative strategy (see Figure 3.8b), which can be predicted almost perfectly for sentences up until length 5 (with accuracies in the range of 0.98 – 1.0), but is also accurately kept track of for longer sequences (an accuracy 0.93 for **L9** sentences). However, the fit with the cumulative strategy is not perfect: the diagnostic classifiers trained to test the cumulative hypothesis perform excellently for left-branching sentences but show low accuracy for right-branching sentences. This is inconsistent with the symbolic description of the cumulative strategy, where the stack is crucial for left-branching sentences, but not relevant at all for right-branching sentences.



(a) prediction of `result` (b) prediction of `mode` cumulative (c) trajectory correlations

Figure 3.8: Results of diagnostic models for a GRU model on different subsets of languages.

3.5.2 Plotting trajectories

To get a better insight in how the predictions of the diagnostic classifier develop over time and to understand which are the points here it potentially goes wrong, I compare the trajectories of the predicted variables with the trajectories of the target variables while the network processes different sentences.

In Figure 3.9, I show the predictions of the diagnostic classifiers on two randomly picked **L9** sentences, along with their target trajectories as defined by the hypotheses. These trajectories confirm that the curve representing the cumulative strategy is much better predicted than the recursive one. A correlation test over 5000 **L9** sentences shows the same trend: pearson's $r = 0.52$ and 0.95 for recursive and cumulative, respectively. Figure 3.8c shows the trajectory correlations for test sentences of different lengths.

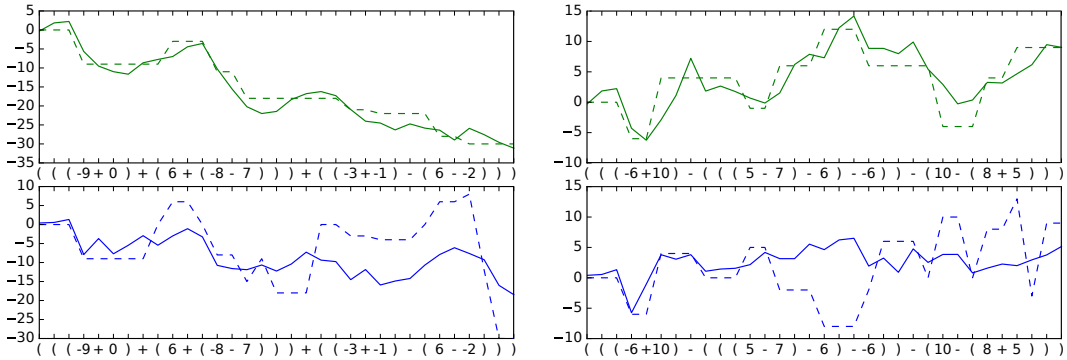
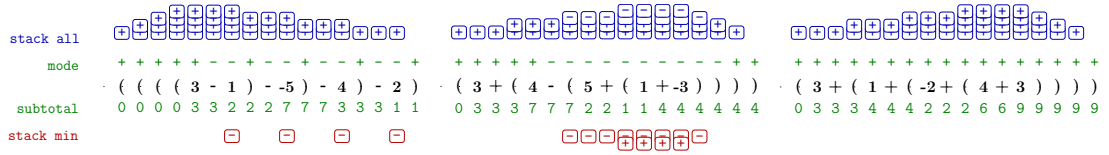


Figure 3.9: Trajectories of the cumulative (green, upper) and recursive (blue, lower) classifier, along with their targets trajectories for the `result` values. Dashed lines show target trajectories.

We also observe an important qualitative difference between the diagnostic classifier trajectories and the target values: the diagnostic classifier trajectories are smooth, changing value at every point in time, whereas the target trajectories are jumpy and often stay on the same value for longer time spans. This once more indicates that a refinement of the symbolic cumulative hypothesis, in which information is integrated more gradually, would be more suitable for a network like this.

3.6 Refining the cumulative hypothesis

The diagnostic classifier results presented in the previous section convincingly demonstrate that trained networks accurately keep track of the *operator* `mode` of the cumulative strategy, required to understand whether the next number should be added or subtracted. What the results do not show is *how* the network computes this variable.



(a) Left-branching, minus (b) Left-branching, mixed (c) Right-branching, plus

Figure 3.10: Three expressions and the values of `mode` and `subtotal` during processing, plus two different examples of stacks used to compute the current `mode`.

3.6.1 Two hypotheses

One possible way to compute the `mode` is to store the current `mode` on a stack whenever an opening bracket is encountered, and pop the next `mode` from the stack whenever a bracket closes (upper blue stack in Figure 3.10). Although elegant for its simplicity, this solution is not consistent with the apparent ease of processing left-branching expressions, which in the described strategy require heavy employment of the stack. The more difficult right-branching expressions instead, lead to accessing the stack only when all computation is already finished (e.g. Figure 3.10b and 3.10c).

An alternative possibility is to start using the stack only when brackets are preceded by a minus (lower, red stack in Figure 3.10). The stack for the latter strategy is shorter and remains primarily empty for left-branching expressions (see Figure 3.10a).

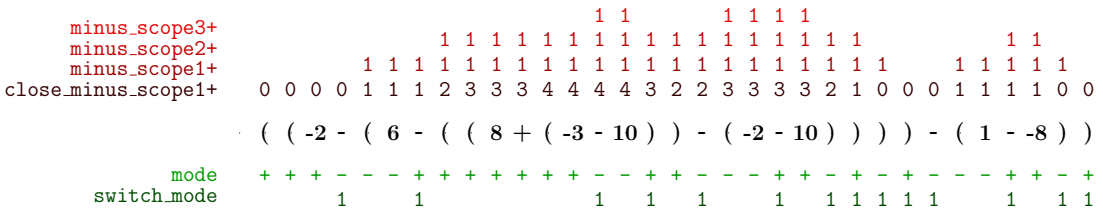
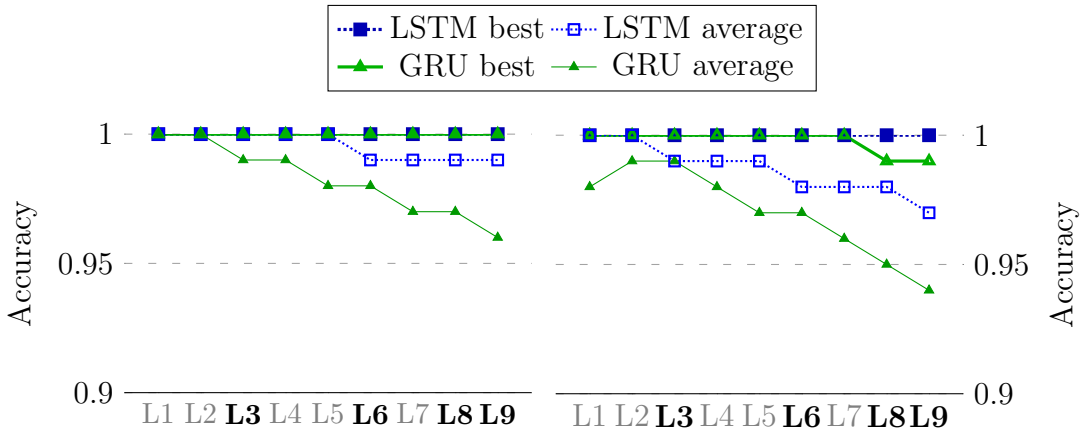


Figure 3.11: An example sentence with the values of different variables at different points in the sentence. The sentence itself is written in black in the middle, the values of the variable `mode` and whether or not this variable should change in the next step (`switch_mode`) are printed below. Above the sentence, I show the values of the variables related to the prediction of the *minus depth* of the sentence. For instance, the row `close_minus_scope1+` indicates how many brackets should be closed before the model is not within the scope of any minus anymore. This feature takes the value 0 when the sentence starts and switches to 1 when the first minus occurs. It then continues to count up with every opening bracket and down with every closing bracket, until the end scope of the highest minus is reached. The three rows above (`minus_scope1+`, `2+` and `3+`) represent binary variables that indicate within the scope of how many minuses the model is at every point in time. For clarity reasons, I did not print the zero values of binary features.

3.6.2 Computing scopes

One feature that distinguishes the full-stack computation of the operator mode from the one where the stack is only employed after a minus (upper blue and bottom red stack in Figure 3.10, respectively), is that the latter requires keeping track of the *scope* of minus operators. I train diagnostic classifiers to test if the hidden state encodes the property of being within the scope of at least one minus (I call this feature `minus_scope1+`), and how many closing brackets should still be seen to close this scope (`close_minus_scope1+`). I refer to Figure 3.11 for a worked out example of an expression and the values these variables take as the expression progresses.

In Figure 3.12a, we see that virtually all networks maintain a representation of `minus_scope1+` during processing. Furthermore, the error on the prediction of the number of closing brackets that need to be encountered to close the scope of the minuses is very low (Figure 3.12b), indicating that `minus_scope1+` is indeed a salient feature for the network. Additional diagnostic classifiers (results not plotted) show that also the scope of minuses of deeper levels (minuses that are within the scope of another minus, such as `minus_scope2+` in Figure 3.11), can be inferred from the hidden states of the networks. Together, these plots confirm that the network indeed approximates the hypothesised cumulative strategy, for which it found a short-cut that only employs a stack when a minus occurs in the expression.



(a) `minus_scope1+` (b) `close_minus_scope`

Figure 3.12: Binary accuracy of diagnostic classifiers trained on predicting the features `minus_scope1+` (left) and `close_minus_scope` (right) from the hidden layer activations of trained models.

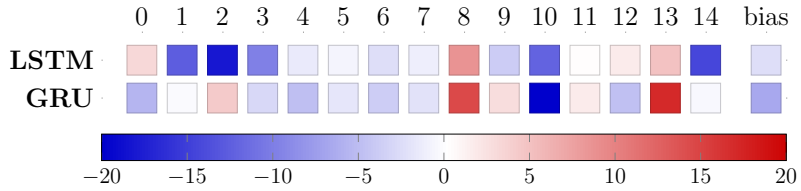
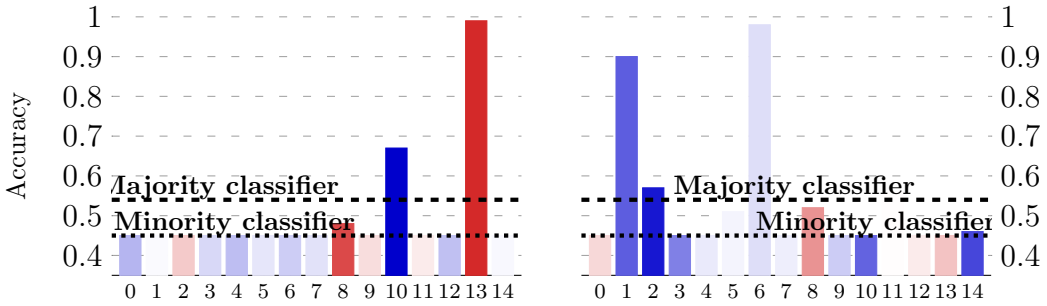


Figure 3.13: Diagnostic classifier weights to predict `minus_scope1+`.

3.6.3 What is encoded where?

Aside from providing a way to quantitatively determine whether information is encoded in the hidden states of a network, diagnostic classifiers can be also used to inspect where and how this information is represented. There are many possible ways to do so, I present one case study that revolves around understanding how information and processes are distributed and located inside a neural network, an aspect which may play a role when one wants to re-utilise learned functions and address the problem of catastrophic forgetting (Goodfellow et al., 2013; McCloskey and Cohen, 1989).

In Figure 3.13 I plot the values of the weights of the diagnostic classifiers trained to predict the variable `minus_scope1+` for the best GRU and LSTM models. Looking at these weights, the representation of this variable seems to be distributed over several different units, that are connected to the diagnostic classifier with a high weight. To measure the contributions of different units of the network to the final accuracy, I test how well individual weights of the diagnostic classifier predict the feature on their own, when all other weights of the classifier



(a) GRU (b) LSTM

Figure 3.14: Accuracy of individual diagnostic classifier weights in predicting `minus_scope1+`.

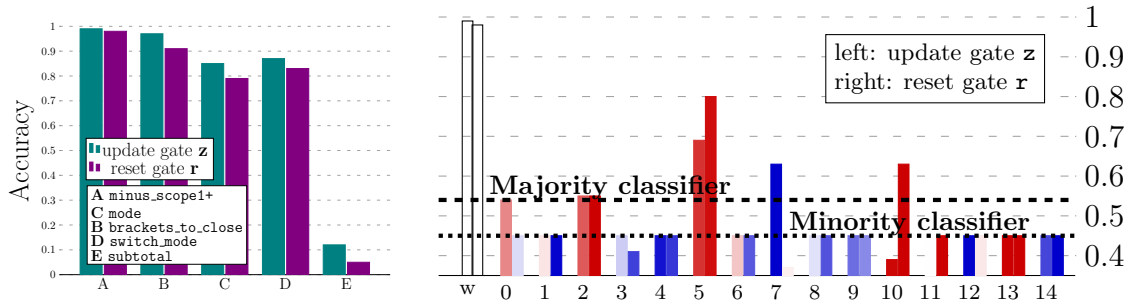
are set to 0. The result of this masking experiment tells a different story than the weights visualisation: in both the LSTM and GRU model, there is one single unit that can predict `minus_scope1+` with almost perfect accuracy on its own, without using any of the other units (Figure 3.14a and Figure 3.14b, colours match the weight visualisation in Figure 3.13). Whether this is an artefact of the training regime leaving traces of previous attempts to solve the problem, a way to ensure robustness of the network, or simply a consequence of the fact that different functions implemented by the network are related, it is certainly worth exploring further. Later, in Chapter 6, we will see another example in which a much larger network maintains a very local representation of particular information.

3.6.4 Diagnosing gates

Diagnostic classifiers may also be used to diagnose gates. I investigate this possibility by focusing on the gates of the best-performing GRU model. As the function of gates is fundamentally different from the function of the hidden units – they are meant to modulate the information flow rather than to encode and memorise information and develop representations – the types of hypotheses for gates differ from hypotheses for the hidden layer activations. For instance, a gate would not likely maintain a prediction of the `subtotal`, as this is not impacting *how* information should be processed by the hidden layer,[4] but it may contain information about the operator `mode`. We train diagnostic classifiers to predict `minus_scope1+` and `close_minus_scope1+`, whether the mode should be switched in the current time step (`switch_mode`) and what the current `mode` is.

From the results in Figure 3.15a, we can see that `minus_scope1+` and `close_minus_scope1+` are much better represented by the gates than the operator `mode`.

[4]And in fact, a diagnostic classifier to predict this value from the gates has a very low accuracy, see Figure 3.15a.



(a) Diagnostic classification accuracy

(b) Masked weights for `in_1minus_scope` diagnostic classifier

Figure 3.15: Diagnosing the gates of the best GRU model.

This is in itself not surprising, as under the hypothesised strategy the former two are sub-features used to compute the latter, and confirms our current strategy hypothesis. Interestingly, contrary to the hidden layer, the computation of `minus_scope1+` seems to be distributed over the gate values, with no single gate value predicting it with high accuracy on its own (Figure 3.15b). In the light of the difference in function of the gates and the hidden layer, this is not entirely surprising: while the hidden layer is required to keep track of `minus_scope1+`, the gate units have to use this information to decide how much every network unit should contribute to the next time step. Furthermore, whereas a hidden layer unit can send information to all gates, a gate unit impacts only one hidden layer unit, making distribution of information over the gates more essential.[5]

Another striking observation is that for this particular model, the update gate (left bars in Figure 3.15a) outperforms the reset gate (right bar) for all features. Both the division of work between gates as well as their exact function is a largely unexplored area, that is worth our consideration. This study demonstrates that diagnostic classifiers could play a role in such a quest. While I will not dig deeper into the functions of the gates for the arithmetics language, later, in Chapter 6, I will discuss their role in the encoding of long-distance subject-verb relations in *language models*.

3.7 Conclusion

In this chapter, I studied how different types of recurrent neural network process hierarchical structures, using an arithmetic language as a convenient, idealised task with unambiguous syntax and semantics. I showed that GRU and LSTM models can learn to compute the meanings of sentences in the arithmetic language

[5]In Chapter 8, we see a similar difference between the way that information is represented in hidden layer and gate activations.

and can generalise to longer expressions than the ones seen in training.

I presented a detailed analysis of *how* the models process these sequences, starting from existing analysis techniques that mostly rely on visual inspection of the hidden state space. To better understand what the network is doing, I developed an approach based on training *diagnostic classifiers* on the internal representations.

The qualitative and quantitative analyses of the results of a diagnostic classifier allowed me to draw conclusions about possible strategies the network might be following. In particular, I found that the successful networks follow a strategy very similar to my hypothesised symbolic 'cumulative strategy'. From this we learn something about how neural networks may process languages with a hierarchical compositional semantics and also provide an example of how we can *open the black box* of the deep learning models in natural language processing when visualisation alone is not sufficient.

Chapter 4

PCFG SET

In the previous chapter I, investigated how recurrent neural networks can process a *hierarchically* structured artificial language. I first tested whether recurrent networks can learn the predict the meanings of the hierarchically structured sentences of this language and then focused on looking inside the model to understand the underlying processes by which they do so. What I did not do is provide an explicit motivation of why I specifically focused on *hierarchy*. In this chapter, I take a slightly broader perspective and consider different aspects of compositionality that may be considered important to be captured by neural networks for them to adequately model the structure of language.

While I was writing this dissertation, the compositional abilities of neural networks have attracted a considerable amount of attention. In the background chapter of this dissertation, I already mentioned several studies that consider this topic (Lake and Baroni, 2018; Liška et al., 2018; Loula et al., 2018). There are many others that consider it from different angles (without claiming to be complete: Bahdanau et al., 2018; Johnson et al., 2017; Saxton et al., 2019; Bowman et al., 2015; Mul and Zuidema, 2019; Tran et al., 2018; Andreas, 2019; Saphra and Lopez, 2019a). Across these studies, there is quite some variation in the way that different researchers test for compositionality and also in the conclusions that are drawn. Some studies test if models are able to productively use symbolic rules, some consider if models can segment inputs into reusable parts, others focus specifically on models' ability to process hierarchical structures. Some researchers conclude that neural networks perform surprisingly well, whereas others are skeptical.

It is my conviction that an important reason for the differences in how researchers test for compositionality as well as the discrepancy between their conclusions is that there is little clarity on what exactly it means for a neural network to behave compositionally. When can we say that a network behaves 'compositionally'? What should it imply when a network is considered compositional? In this chapter, starting from the literature about compositionality in linguistics and philosophy, I reflect upon these questions and propose a series of tests that

tease apart different aspects of compositionality and try to relate them to each other. As a case study, I will illustrate the application of these tests to a recurrent sequence-to-sequence model and contrast these findings with the results for convolution and attention-based models.[1]

Chapter outline This chapter is structured as follows. First, I briefly discuss compositionality and explain and motivate the five concepts we consider in our tests. Then, I discuss the artificial data that the tests we defined are applied to. In Section 4.3 I describe the architectures that we used to showcase our tests. In Section 4.4, I discuss the experiments and results.

4.1 Compositionality

There exist several versions of *The principle of compositionality*. My personal favourite is the formulation by Partee (1995):

> "*The meaning of a whole is a function of the meanings of the parts and of the way they are syntactically combined*"

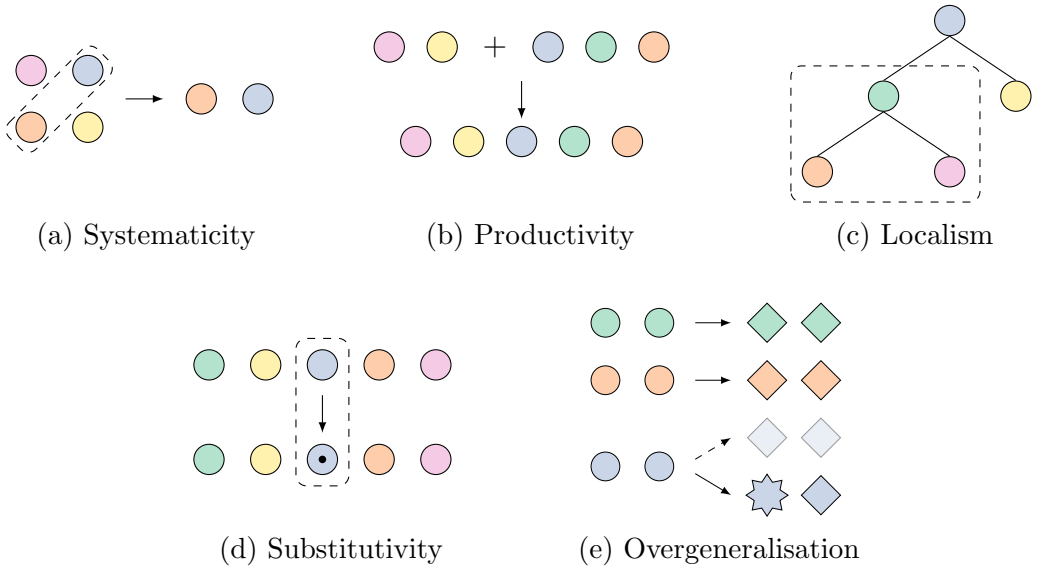
While there is ample support for this principle, there is less consensus about its exact interpretation and practical implications. One important reason for this is that the principle is not theory-neutral: it requires a theory of both syntax and meaning, as well as functions to determine the meaning of composed parts. Without these components, the principle of compositionality is formally vacuous (Janssen, 1983; Zadrozny, 1994). because also trivial and intuitively non-compositional solutions that cast every expression as one part and assign it a meaning as a whole do not formally violate the principle of compositionality. Furthermore, the principle of compositionality describes a property of a *language*, not the property of a language user or learner. What does it mean for a *learner* such as a neural network model to be compositional? Is it enough that it can correctly model compositional data? Are there restrictions on how it has to do so?

In this section, I address this question by collecting different aspects of and intuitions about compositionality from philosophy and linguistics and translating

[1] This chapter is a condensed and adapted version of the following paper:

> Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. The compositionality of neural networks: integrating symbolism and connectionism. *To appear in Journal of Artificial Intelligence Research (JAIR)*, 2019a.

This paper was accepted for publication at the Journal of Artificial Intelligence Research (JAIR) and will appear in 2020. The experimental part of the project was conducted by Verna Dankers and Mathijs Mul, under supervision of dr. Elia Bruni and I. The base language was proposed by Mathijs Mul, the individual tests were designed in collaboration; many of the additional analysis methods were proposed by me. The text of the article was primarily written by me, while the plots are mostly made by Mathijs Mul and Verna Dankers.



(a) Systematicity (b) Productivity (c) Localism

(d) Substitutivity (e) Overgeneralisation

Figure 4.1: Schematic depictions of the five tests for compositionality proposed in this paper. (a) To test for systematicity, we evaluate models' ability to recombine known parts to form new sequences. (b) While the productivity test also requires recombining known parts, the focus there lies on unboundedness: we test if models can understand sequences *longer* than the ones they were trained on. (c) The localism test targets how local the composition operations of models are: are smaller constituents evaluated before larger constituents? (d) With the substitutivity test, we evaluate how robust models are towards the introduction of synonyms, and, more specifically, in which cases words are considered synonymous by different models. (e) The overgeneralisation task evaluates how likely models are to infer rules: is a model instantly able to accommodate exceptions, or does it need more evidence to deviate from applying the general rule instantiated by the rest of the data?

them into tests for neural networks. With this, I aim to provide an evaluation paradigm that can be used to better understand the requirements for compositional learning as well as the actual composition functions learned by neural models trained end-to-end on a downstream task.

In particular, I describe tests that target (i) if models systematically recombine known parts and rules (*systematicity*) (ii) if models can extend their predictions beyond the length they have seen in the training data (*productivity*) (iii) if models' composition operations are local or global (*localism*) (iv) if models' predictions are robust to synonym substitutions (*substitutivity*) and (v) if models favour rules or exceptions during training (*overgeneralisation*). A schematic depiction of the five tests is shown in Figure 4.1.

4.1.1 Systematicity

The first property we propose to test for – following many of the earlier-mentioned studies on the compositionality of neural networks – is *systematicity*, a notion frequently used in the context of compositionality. The term was introduced by Fodor and Pylyshyn (1988) – notably, in a paper that argued against connectionist architectures – who used it to denote that

> "[t]he ability to produce/understand some sentences is intrinsically connected to the ability to produce/understand certain others" (Fodor and Pylyshyn, 1988, p. 25)

This ability concerns the recombination of known parts and rules: anyone who understands a number of complex expressions also understands other complex expressions that can be built up from the constituents and syntactical rules employed in the familiar expressions. To use a classic example from Szabó (2012): someone who understands 'brown dog' and 'black cat' also understands 'brown cat'.

Rules and memorisation

Fodor and Pylyshyn (1988) contrast systematicity with storing all sentences in an atomic way, in a dictionary-like mapping from sentences to meanings. Someone who entertains such a dictionary would not be able to understand new sentences, even if they were similar to the ones occurring in their dictionary. Since humans are evidently able to infer meanings for sentences they have never heard before, they must use some systematic process to construct these meanings from the ones they internalised before.

By the same argument, however, any model that is able to generalise to a sequence outside its training space (its test set), should have learned to construct outputs from parts it perceived during training and some rule of recombination. Thus, rather than asking if a model is systematic, a more interesting question is whether the rules and constituents the model uses are in line with what we believe to be the actual rules and constituents underlying a particular data set or language.

Testing systematicity

With our *systematicity* test, we aim to pull out that specific aspect, by testing if a model can recombine constituents that have not been seen together during training. In particular, we focus on combinations of words a and b that meet the requirements that (i) the model has only been familiarised with a in contexts excluding b and vice versa but (ii) the combination $a\ b$ is plausible given the rest of the corpus.

4.1.2 Productivity

A notion closely related to systematicity is *productivity*, which concerns the open-ended nature of natural language: language appears to be infinite, but has to be stored with finite capacity. Hence, there must be some productive way to generate new sentences from this finite storage.[2] While this 'generative' view of language became popular with Chomsky in the early sixties (Chomsky, 1956), Chomsky himself traces it back to Von Humboldt, who stated that 'language makes infinite use of finite means'.

Both systematicity and productivity rely on the recombination of known constituents into larger compounds. However, whereas systematicity can be – to some extent – empirically established, productivity cannot, as it is not possible to prove that natural languages in fact contain an infinite number of complex expressions. Even if humans' memory allowed them to produce infinitely long sentences, their finite life prevents them from doing so. Productivity of language is therefore more controversial than systematicity.

Testing productivity

To separate systematicity from productivity, in our productivity test we specifically focus on the aspect of unboundedness. We test whether different learners can understand sentences that are *longer* than the ones encountered during training. To test this, we separate sequences in the data based on length and evaluate models on their ability to cope with longer sequences after having been familiarised with the shorter ones.

4.1.3 Substitutivity

A principle closely related to the principle of compositionality is the principle of *substitutivity*. This principle, which finds its origin in philosophical logic, states that if an expression is altered by replacing one of its constituents with another constituent with the same meaning (a synonym), this does not affect the meaning of the expression (Pagin, 2003). In other words, if a substitution preserves the meaning of the parts of a complex expression, it also preserves the meaning of the whole. In the latter formulation, the correspondence with the principle of compositionality itself can be easily seen: since substituting part of an expression with a synonym changes neither the structure of the expression nor the meaning of its parts, it should not change the meaning of the expression itself either.

As the principle of compositionality, the substitutivity principle in the context of natural language is subject to interpretation and discussion. Husserl (1913) pointed out that the substitution of expressions with the same meaning can result

[2]The term productivity also has a technical meaning in morphology, which we do not imply here.

in nonsensical sentences if the expressions belong to different semantic categories (the philosopher Geach (1965) illustrated this considering the two expressions *Plato was bald* and *baldness was an attribute of Plato*, which are synonymous but cannot be substituted in the sentence *The philosopher whose most eminent pupil was Plato was bald*).

A second context that poses a challenge for the substitutivity principle concerns embedded statements about beliefs. If X and Y are synonymous, this does not necessarily imply that the expressions *Peter thinks that X* and *Peter thinks that Y* are both true. In this work, we do not consider these cases, but instead focus on the more general question: is substitutivity a salient notion for neural networks and under what conditions can and do they infer synonymity?

Testing substitutivity

We test substitutivity by probing under which conditions a model considers two atomic units to be synonymous. To this end, we artificially introduce synonyms and consider how the prediction of a model changes when an atomic unit in an expression is replaced by its synonym. We consider two different cases. Firstly, we analyse the case in which synonymous words occur equally often and in comparable contexts. In this case, synonymity can be inferred from the corresponding meanings on the output side but is aided by distributional similarities on the input side. Secondly, we consider pairs of words in which one of the words occurs only in very short sentences (we will call those *primitive contexts*). In this case, synonymity can only be inferred from the (implicit) semantic mapping, which is identical for both words, but not from the context that those words appear in.

4.1.4 Localism

In its basic form, the principle of compositionality states that the meaning of a complex expression derives from the meanings of its constituents and how they are combined. It does not impose any restrictions on what counts as an admissible way of combining different elements, which is why the principle taken in isolation is formally vacuous.[3] As a consequence, the interpretation of the principle of compositionality depends on the strength of the constraints that are put on the semantic and syntactic theories involved. One important consideration concerns – on an abstract level – how *local* the composition operations should be.

When operations are very local (a case also referred to as *strong* or *first-level* compositionality), the meaning of a complex expression depends only on its local structure and the meanings of its immediate parts (Pagin and Westerståhl, 2010; Jacobson, 2002). In other words, the meaning of a compound is only dependent on

[3]We previously cited Janssen (1983), who proved this claim by showing that arbitrary sets of expressions can be mapped to arbitrary sets of meanings without violating the principle of compositionality, as long as one is not bound to a fixed syntax.

the *meaning* of its immediate 'children', regardless of the way that their meaning was built up. In *global* or *weak* compositionality, the meaning of an expression follows from its total (global) structure and the meanings of its atomic parts. In this interpretation, compounds can have different meanings, depending on the larger expression that they are a part of.

Carnap (1947) presents an example (similar to the one sketched in the previous section) that nicely illustrates the difference between these two interpretations, in which he considers sentences with tautologies. Under the view that the meaning of declarative sentences is determined by the set of all worlds in which this sentence is true, any two tautologies X and Y are synonymous. Under the local interpretation of compositionality, this entails that also the phrases 'Peter thinks that X' and 'Peter thinks that Y' should be synonymous, which is not necessarily the case, as Peter may be aware of some tautologies but unaware of others. The global interpretation of compositionality does not give rise to such a conflict, as X and Y, despite being identical from a truth-conditional perspective, are not structurally identical. Under this representation, the meanings of X and Y are locally identical, but not globally, if also the phrase they are a part of is considered.

For natural language, contextual effects, such as the disambiguation of a phrase or word by a full utterance or even larger piece of discourse, makes the localist account highly controversial. As a contrast, consider an arithmetic task, where the outcome of `14 - (2 + 3)` does not change when the subsequence `(2+3)` is replaced by `5`, a sequence with the same (local) meaning, but a different structure.

Testing localism

We test if a model's composition operations are local or global by comparing the meanings it assigns to stand-alone sequences to those it assigns to the same sequences when they are part of other sequences. More specifically, we compare a model's output when it is given a composed sequence `X`, built up from two parts `A` and `B` with the output the same model gives when it is forced to first separately process `A` and `B` in a local fashion. If the model employs a local composition operation that is true to the underlying compositional system that generated the language, there should be no difference between these two outputs. A difference between these two outputs, instead, indicates that the model does not compute meanings by first computing the meanings of all subparts, but pursues a different, more global, strategy.

4.1.5 Overgeneralisation

The previously discussed compositionality arguments are of mixed nature. Some – such as productivity and systematicity – are intrinsically linked to the way that humans acquire and process language. Others – such as substitutivity and localism – are properties of the mapping from signals to meanings in a particular language.

While it can be tested if a language user abides by these principles, these principles themselves do not relate directly to language users. To complete our set of tests to assess whether a model learns compositionally, we include also a notion that exclusively concerns the acquisition of the language by a model: we consider if models exhibit *overgeneralisation* when faced with *non*-compositional phenomena.

Overgeneralisation (or overregularisation) is a language acquisition term, which refers to the scenario in which a language learner applies a general rule in a case that forms an exception to this rule. One of the most well-known examples, which served also as the subject of the famous *past-tense debate* between symbolism and connectionism (Rumelhart et al., 1986; Marcus et al., 1992), concerns the rule that English past tense verbs can be formed by appending *-ed* to the stem of the verb. During the acquisition of past tense forms, learners infrequently use this rule also for irregular verbs, resulting in forms like *goed* (instead of *went*) or *breaked* (instead of *broke*).

The relation of overgeneralisation with compositionality comes from the supposed evidence that overgeneralisation errors provide for the presence of symbolic rules in the human language system (see, e.g. Penke, 2012). In this work, we follow this line of reasoning and take the application of a rule in a case where this is contradicted by the data as evidence that the model in fact internalised this rule. In particular, we regard a model's inclination to apply rules as the expression of a compositional bias. This inclination is most easily observed in the case of exceptions, where the correct strategy is to ignore the rules and learn on a case-by-case basis. If a model overgeneralises by applying the rules also to such cases, we hypothesise that this in particular demonstrates compositional awareness.

Testing overgeneralisation

We propose an experimental setup where a model's tendency to overgeneralise is evaluated by monitoring its behaviour on exceptions. We identify samples that do not adhere to the rules underlying the data distribution– *exceptions* – in the training data sets and assess the tendency to overgeneralise by observing how architectures model these exceptions during training: (when) do they consistently follow a global rule set, and (when) do they (over)fit the training samples individually?

4.2 Data

As in the previous chapter, the task we consider consists in predicting the meaning of input sequences. Contrary to the previous chapter, here, the meanings of the sequences are sequential themselves. The task can thus be seen as a sequence-to-sequence *translation* task, which requires an encoder-decoder model with a

sequential decoder.

The input sequences of this task are sequences that are generated by a probabilistic context-free grammar (PCFG), they are to be translated into output sequences that represent their meanings. These output sequences are constructed from input sequences by recursively applying the *string edit* operations that are specified in the latter. We call this string edit task PCFG SET.

4.2.1 Input sequences: syntax

The input alphabet of PCFG SET contains three types of words: words for unary and binary functions that represent *string edit operations* (e.g. `append`, `copy`, `reverse`), elements to form the string sequences that these functions can be applied to (e.g. `A`, `B`, `A1`, `B1`), and a separator to separate the arguments of a binary function (`,`). The input sequences that are formed with this vocabulary are sequences describing how a series of such operations are to be applied to a string argument. For instance:

```
repeat A B C
echo remove_first D , E F
append swap F G H , repeat I J
```

We generate input sequences with a PCFG, shown in Figure 4.2 (for clarity, production probabilities are omitted). As the grammar we use for generation is recursive, we can generate an infinite number of admissible input sequences. Because the operations can be nested, the parse trees of valid sequences can be arbitrarily deep and long. Additionally, the distributional properties of a particular PCFG SET data set can be controlled by adjusting the probabilities of the grammar and varying the number of input characters. We will use this to *naturalise* the data set such that its distribution of lengths and depths correspond to the distribution observed in a data set containing English sentences.

4.2.2 Output sequences: semantics

The meaning of a PCFG SET input sequence is constructed by recursively applying the string edit operations specified in the sequence. This mapping is governed by the interpretation functions listed in Figure 4.3. Following these interpretation functions, the three sequences listed above would be mapped to output sequences as follows:

```
repeat A B C                    →  A B C A B C
echo remove_first D , E F       →  E F F
append swap F G H , repeat I J  →  H G F I J I J
```

Non-terminal rules	
S	$\rightarrow F_U\ S \ \mid \ F_B\ \ S\ ,\ S$
S	$\rightarrow X$
X	$\rightarrow XX$
Lexical rules	
F_U	$\rightarrow$ `copy` | `reverse` | `shift` | `echo` | `swap` | `repeat`
F_B	$\rightarrow$ `append` | `prepend` | `remove_first` | `remove_second`
X	$\rightarrow$ `A` | `B` | ... | `Z` | `A2` | ... | `B2` | ...

Figure 4.2: The context-free grammar that describes the entire space of grammatical input sequences in PCFG SET. Rule probabilities (not depicted) can be used to control the distributional properties of a PCFG SET data set. We will use this property to make sure that our data matches a corpus with natural English sentences in terms of length and depth distributions.

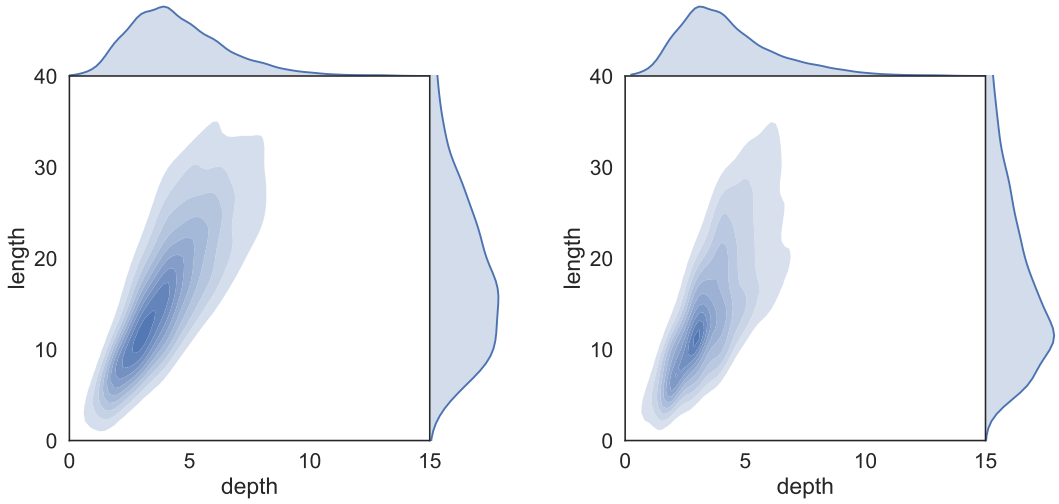
Unary functions F_U:		**Binary functions** F_B:	
`copy` $x_1 \cdots x_n$	$\rightarrow x_1 \cdots x_n$	`append` **x**, **y**	$\rightarrow$ **x y**
`reverse` $x_1 \cdots x_n$	$\rightarrow x_n \cdots x_1$	`prepend` **x**, **y**	$\rightarrow$ **y x**
`shift` $x_1 \cdots x_n$	$\rightarrow x_2 \cdots x_n\ x_1$	`remove_first` **x**, **y**	$\rightarrow$ **y**
`swap` $x_1 \cdots x_n$	$\rightarrow x_n\ x_2 \cdots x_{n-1}\ x_1$	`remove_second` **x**, **y**	$\rightarrow$ **x**
`repeat` $x_1 \cdots x_n$	$\rightarrow x_1 \cdots x_n\ x_1 \cdots x_n$		
`echo` $x_1 \cdots x_n$	$\rightarrow x_1 \cdots x_n\ x_n$		

Figure 4.3: The interpretation functions describing how the meanings of PCFG SET input sequences are built up compositionally.

The definitions of the interpretation functions specify the systematic procedure by which an output sequence can be constructed from an input sequence, without having to enumerate particular input-output pairs. In this sense, PCFG SET differs from a task such as the lookup table task introduced by Liška et al. (2018) – where functions must be exhaustively defined because there is no systematic connection between arguments and the values to which functions map them – but shares some aspects with SCAN (Lake and Baroni, 2018).

4.2.3 Data construction

PCFG SET describes a general framework for producing many different data sets. We used several criteria to select the PCFG SET input-output pairs for our experiments.



(a) WMT17 (b) PCFG SET data

Figure 4.4: Distribution of lengths and depths in (a) the PCFG SET data and (b) English WMT 2017 test data.

Naturalisation of structural properties

We use the probabilistic nature of the PCFG SET input grammar to enforce an input distribution that resembles the statistics of a more natural data set in two relevant respects: the length of the expressions and the depth of their parse trees. To obtain these statistics, we use the English side of a large machine translation corpus: WMT 2017 (Bojar et al., 2017). We parse this corpus with a statistical parser (Manning et al., 2014) and extract the distribution of length and depths from the annotated corpus. We then use expectation maximisation to tune the PCFG parameters in such a way that the resulting bivariate distribution of the generated data mimics the one extracted from the WMT data. In Figure 4.4a and Figure 4.4b, respectively, we plot the distributions of the WMT data and a sample of around ten thousand sentences of the resulting PCFG SET data.

Sentence selection

We set the size of the string alphabet to 520 and create a base corpus of 100 thousand distinct input-output pairs. We use 85% of this corpus for training, 5% for validation and 10% for testing.

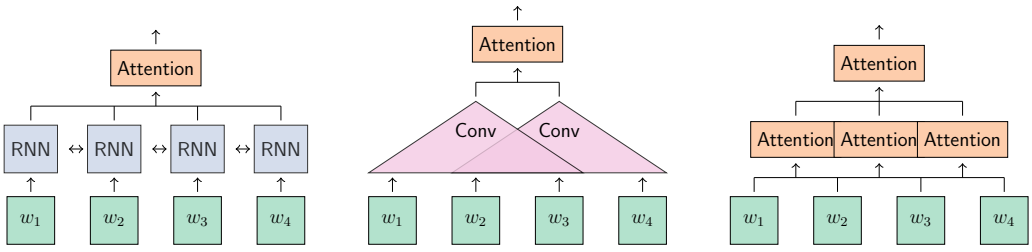
As argued earlier in this chapter, the fact that a data set is generated by a compositional system does not necessarily imply that successfully generalising to a particular test set requires knowing this underlying system. Often, a learner may get away with concatenating memorised strings or following another strategy that is unrelated to the compositional rules of the system. With PCFG SET, we aim

to create a task for which it should not be possible to obtain a high test accuracy by following alternative strategies.

In particular, we assure that the train and test data are linked *only* by implicit systematic rules, by never repeating the same arguments to an input function. As a consequence, models should not profit from memorising the answers of specific function-argument pairs. Furthermore, since the accuracy on PCFG SET is directly linked to a model's ability to infer and execute compositional rules, the training signal a model receives during training should convey that a compositional solution should be found. Thereby, we aim to give models the best possible chance to learn a compositional solution.

4.3 Architectures

As a use-case for our compositionality test-suite, I now apply it to compare three currently popular neural architectures for sequence-to-sequence language processing: recurrent networks (Sutskever et al., 2014), convolutional neural networks (Gehring et al., 2017b) and Transformer networks (Vaswani et al., 2017). I explain their most important features, give a brief overview of their potential strengths and weaknesses in relation to compositionality, and I describe how they are implemented in our experiments.



(a) LSTMS2S (b) ConvS2S (c) Transformer

Figure 4.5: High-level graphical depictions of the most important features of the encoding mechanisms in LSTMS2S, ConvS2S, and Transformer models. (a) LSTMS2S processes the input in a fully sequential way, iterating over the embedded elements one by one in both directions before applying an attention layer. (b) ConvS2S divides the input sequence into local fragments of consecutive items that are processed by the same convolutions, before applying attention. (c) Transformer immediately applies several global attention layers to the input, without incrementally constructing preliminary representations.

4.3.1 LSTMS2S

The first architecture we consider is a recurrent encoder-decoder model with attention. This setup is considered to be the most basic of the three setups we consider, but is still the basis of many MT applications (e.g. OpenNMT, Klein et al., 2017) and has also been successful in the fields of speech recognition (e.g. Chorowski et al., 2015) and question answering (e.g. He and Golub, 2016). We consider the version of this model in which both the decoder and encoder are LSTMs and refer to this setup with the abbreviation *LSTMS2S*.

Model basics

LSTMS2S is a fully recurrent, bidirectional model. The encoder processes an input by iterating over all of its elements in both directions and incrementally constructing a representation for the entire sequence. Upon receiving the encoder output, the decoder performs a similar, sequential computation to unroll the predicted sequence. Here, LSTMS2S uses an attention mechanism, which allows it to focus on the parts of the encoded input that are estimated to be most important at each moment in the decoding process.

The sequential fashion with which LSTMS2S model processes each input potentially limits the model's abilities to recombine components hierarchically. While depth – and, as shown by Blevins et al. (2018), thus hierarchy – can be created by stacking neural layers, the number of layers in popular recurrent sequence-to-sequence setups tends to be limited. The attention mechanism of the encoder-decoder setup positively influences the skills of LSTMS2S to hierarchically process inputs, as it allows the decoder to focus on input tokens out of the sequential order.

Model implementation

We use the LSTMS2S implementation of the OpenNMT-py framework (Klein et al., 2017). We set the hidden layer size to 512, number of layers to 2 and the word embedding dimensionality to 512, matching their best setup for translation from English to German with the WMT 2017 corpus, which we used to shape the distribution of the PCFG SET data. We use mini-batches of 64 sequences and stochastic gradient descent with an initial learning rate of 0.1.

4.3.2 ConvS2S

The second architecture we consider is a convolutional sequence-to-sequence model. Convolutional sequence-to-sequence models have obtained competitive results in machine translation (Gehring et al., 2017a) and abstractive summarisation (Denil et al., 2014). In this paper, we follow the setup described by Gehring et al. (2017b) and use their nomenclature: we refer to this model with the abbreviation *ConvS2S*.

Model basics

ConvS2S uses a convolutional model to encode input sequences, instead of a recurrent one. The decoder uses a multi-step attention mechanism – every layer has a separate attention mechanism – to generate outputs from the encoded input representations. Although the convolutions already contextualise information in a sequential order, the source and target embeddings are also combined with position embeddings that explicitly encode order. As at the core of the ConvS2S model lies the local mechanism of one-dimensional convolutions, which are repeatedly and hierarchically applied, ConvS2S has a built-in bias for creating compositional representations. Its topology is also biased towards the integration of local information, which may hinder modelling long-distance relations. However, convolutional networks have found to maintain a much longer effective history than their recurrent counterparts (Bai et al., 2018). Within ConvS2S, such distance portions in the input sequence may be combined primarily through the multi-step attention, which has been shown to improve the generalisation abilities of the model compared to single-step attention (Dessì and Baroni, 2019).

Model implementation

We use the ConvS2S setup that was presented by Gehring et al. (2017b). Word vectors are 512-dimensional. Both the encoder and decoder have 15 layers, with 512 hidden units in the first 10 layers, followed by 768 units in two layers, all using kernel width 3. The final three layers are 2048-dimensional. We train the network with the Fairseq Python toolkit[4], using the predefined `fconv_wmt_en_de` architecture. Unless mentioned otherwise, we use the default hyperparameters of this library. We replicate the training procedure of Gehring et al. (2017b), using Nesterov's accelerated gradient method and an initial learning rate of 0.25. We use mini-batches of 64 sentences, with a maximum number of tokens of 3000. The gradients are normalised by the number of non-padded tokens in a batch.

4.3.3 Transformer

The last model we consider is the recently introduced Transformer model (Vaswani et al., 2017). Transformer models constitute the current state-of-the-art in machine translation and are becoming increasingly popular also in other domains, such as language modelling (e.g. Radford et al., 2019). We refer to this setup with simply the name *Transformer*.

[4]Fairseq toolkit: `https://github.com/pytorch/fairseq`

Model basics

Transformers use neither RNNs nor convolutions to convert an input sequence to an output sequence. Instead, they are fully based on a multitude of attention mechanisms. Both the encoder and decoder of a transformer are composed of a number of feed-forward layers, each containing two sub-layers: a multi-head attention module and a traditional feed-forward layer. In the multi-head attention layers, several attention tensors (the 'heads') are computed in parallel, concatenated and projected. In addition to a self-attention layer, the decoder has another layer, which computes multi-head attention over the outputs of the encoder.

Since Transformers do not have any inherent notion of sequentiality, the input embeddings are combined with position embeddings, from which the model can infer *order*. For Transformer models, the cost of relating symbols that are far apart is thus not higher than relating words that are close together, which – in principle – should ease modelling long-distance dependencies. The setup of attention-based stacked layers furthermore makes the architecture suitable for modelling hierarchical structure in the input sequence that needs not necessarily correspond to the sequential order. On the other hand, the non-sequential nature of the Transformer could be a handicap as well, particularly for relating consecutive portions in the input sequence. Transformers' receptive field is inherently global, which can be challenging in such cases.

Model implementation

We use a Transformer model with an encoder and decoder that both contain 6 stacked layers. The multi-head self-attention module has 8 heads, and the feed-forward network has a hidden size of 2048. All embedding layers and sub-layers in the network produce outputs of dimensionality 512. In addition to word embeddings, positional embeddings are used to indicate word order. We use OpenNMT-py[5] (Klein et al., 2017) to train the model according to the guidelines provided by the framework[6]: with the Adam optimiser ($\beta_1 = 0.9$ and $\beta_2 = 0.98$) and a learning rate increasing for the first 8000 'warm-up steps' and decreasing afterwards.

4.4 Experiments and results

For every experiment, we conduct three runs per model type (LSTMS2S, ConvS2S and Transformere) and report both the average and standard deviation of their

[5]Pytorch port of OpenNMT: `https://github.com/OpenNMT/OpenNMT-py`.
[6]Visit `http://opennmt.net/OpenNMT-py/FAQ.html` for the guidelines.

Experiment	LSTMS2S	ConvS2S	Transformer
Task accuracy	0.79 ± 0.01	0.85 ± 0.01	0.92 ± 0.01
Systematicity*	0.53 ± 0.03	0.56 ± 0.01	0.72 ± 0.00
Productivity*	0.30 ± 0.01	0.31 ± 0.02	0.50 ± 0.02
Substitutivity, *equally distributed*†	0.75 ± 0.00	0.92 ± 0.00	0.96 ± 0.01
Substitutivity, *primitive*†	0.60 ± 0.01	0.58 ± 0.01	0.90 ± 0.00
Localism†	0.46 ± 0.00	0.59 ± 0.04	0.54 ± 0.02
Overgeneralisation*	0.68 ± 0.04	0.79 ± 0.06	0.88 ± 0.07

Table 4.1: General task performance and performance per tests for PCFG SET. The results are averaged over three runs and the standard deviation is indicated. Two performance measures are used: *sequence accuracy*, indicated by *, and *consistency score*, indicated by †.

scores.[7] We train all models of all architectures for 25 epochs, or until convergence, and select the best-performing model based on the performance on the validation set. Below, I describe the precise experiments we conducted and report their results, going test by test. A summary is provided in Table 4.1.

4.4.1 Task accuracy

I first consider the correctness of the output sequences of the three different architectures on the data as described in Section 4.2.3. In particular, I consider their *sequence accuracy*, where only instances for which the entire output sequence equals the target are considered correct. This accuracy measure is used to evaluate the overall task performance, and later also for the systematicity, productivity and overgeneralisation tests. In the rest of this chapter, I denote accuracy scores with *.

The average task performance on the PCFG SET data for the three different architectures is shown in the first row of Table 4.1. The Transformer outperforms both LSTMS2S and ConvS2S ($p \approx 10^{-4}$ and $p \approx 10^{-3}$, respectively), with a surprisingly high accuracy of 0.92. ConvS2S, in turn, is with its 0.85 accuracy significantly better than LSTMS2S ($p \approx 10^{-3}$), which has an accuracy 0.79. The scores of the three architectures are robust with respect to initialisation and order of presentation of the data, as evidenced by the low variation across runs. I now present a breakdown of this task accuracy on different types of subsets of the data.

[7]Some experiments, such as the localism experiment, can be conducted directly on models trained for other tests and thus do not require training new models.

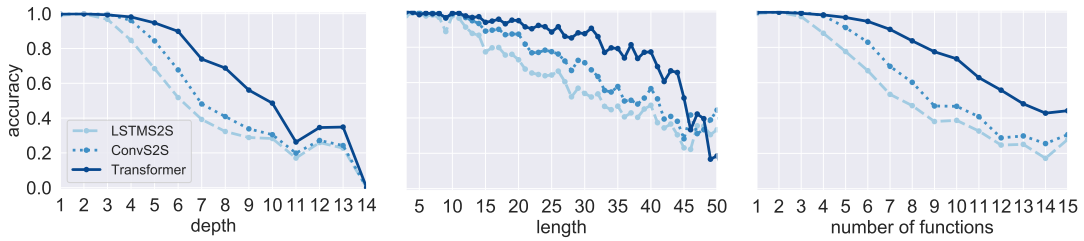


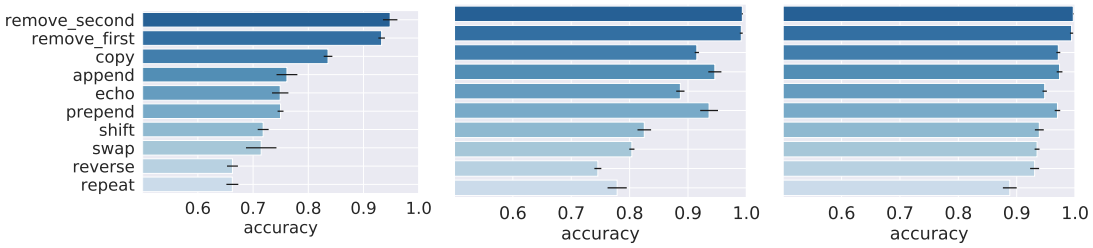
Figure 4.6: Sequence accuracy of the three models as a function of several properties of the input sequences for the general PCFG SET test set: *depth* of input's parse tree, the input sequence's *length* and the *number of functions* input sequence. The results are averaged over three model runs and computed over ten thousand test samples.

Impact of length, depth, and number of functions

We explore how the accuracy of the three different architectures develops with increasing difficulty of the input sequences, as measured in the input sequence's depth (the maximum level of nestedness observed in a sequence), the input sequence's length (number of tokens) and the number of functions in the input sequence. In Figure 4.6, I show the average accuracy for all three architectures as a function of depth, length and number of functions in the input. Unsurprisingly, the accuracy of all architecture types decreases with the length, depth, and number of functions in the input. All architectures have learned to successfully model sequences with low depths and lengths and a small number of functions (reflected by accuracies close to 1). Their performance drops for longer sequences with more functions. Overall the Transformer > ConvS2S > LSTMS2S trend is preserved across the different data subsets.

Function difficulty

Since the input sequences typically contain multiple functions, it is not possible to directly evaluate whether some functions are more difficult for models than others. On sequences that contain only one function, all models achieve a maximum accuracy. To compare the difficulty of the functions, we create one corpus with composed input sequences and derive for each function a separate corpus in which this function is applied to those composed input sequences. We then express the comparative difficulty of a function for a model as this model's accuracy on the corpus corresponding to this function. For example, to compare the functions `echo` and `reverse`, we create two minimally different corpora that only differ with respect to the first input function in the sequence (e.g. `echo append swap F G H , repeat I J` and `reverse append swap F G H , repeat I J`), and compute



(a) LSTMS2S (b) ConvS2S (c) Transformer

Figure 4.7: Accuracy of the three models per PCFG SET function, as computed by applying the different functions to the same complex input sequences.

the model's accuracy on both corpora.[8] The results are shown in Figure 4.7.

The ranking of functions in terms of difficulty is similar for all models, suggesting that the difficulties are to a large extent stemming from the objective complexity of the functions themselves, rather than from specific biases in the models. In some cases, it is very clear why. The function `echo` requires copying the input sequence and repeating its last element – regardless of the bias of the model this should be at least as difficult as `copy` which requires just to copy the input. Similarly, `prepend` and `append` require repeating two string arguments, whereas for `remove_first` and `remove_second` only one argument needs to be repeated. The latter functions should thus be easier, irrespective of the architecture. The relative difficulty of `repeat` reflects that generating longer output sequences proves challenging for all architectures. As this function requires to output the input sequence twice, its output is around twice as long as the output of another unary function applied to an input string of the same length.

An interesting difference between architectures occurs for the function `reverse`. For both LSTMS2S and ConvS2S this is a difficult function (although `repeat` is even harder than `reverse` for LSTMS2S). For the Transformer, the accuracy for `reverse` is on par with the accuracies of `echo`, `swap` and `shift`, functions that are substantially easier than `reverse` for the other two architectures. This difference follows directly from architectural differences: while LSTMS2S and ConvS2S are forced to encode ordered local context – as they are recurrent or apply local convolutions – the Transformer is not bound to such an ordering and can thus more easily deal with inverted sequences.

4.4.2 Systematicity

The task accuracy for PCFG SET already reflects whether models are able to recombine functions and input strings that were not seen together during training.

[8]Note that the since inputs to unary and binary functions are different, we have to use two different corpora to compare binary and unary function difficulty. The unary and binary function scores in Figure 4.7 are thus not directly comparable.

In the systematicity test, we focus explicitly on the ability of model to interpret pairs of functions that were never seen together while training.

Test details

We evaluate four pairs of functions: `swap repeat`, `append remove_second`, `repeat remove_second` and `append swap`.[9] We redistribute the training and test data such that the training data does not contain any input sequences including these specific four pairs, and all sequences in the test data contain at least one. After this redistribution, the training set contains 82 thousand input-output pairs, while the test set contains 10 thousand examples. Note that while the training data does not contain any of the function pairs listed above, it still may contain sequences that contain both functions. E.g. `reverse repeat remove_second A B , C D` cannot appear in the training set, but `repeat reverse remove_second A B , C D` might.

Results

Following the task accuracy, also for the systematicity test the Transformer model obtains higher scores than both LSTMS2S and ConvS2S ($p \approx 10^{-2}$ and $p \approx 10^{-3}$, respectively). The difference between the latter two, however, is for this test statistically insignificant ($p \approx 10^{-1}$). The relative differences between the Transformer model and the other two models gets larger. In Table 4.2, we show the average accuracies of the three architectures on all four held-out function pairs.

The large difference between task accuracy and systematicity is to some extent surprising, since PCFG SET is constructed such that a high task accuracy requires systematic recombination. As such, these results serve as a reminder that models may find unexpected solutions, even for very carefully constructed data sets. A potential explanation for this particular discrepancy is that, due to the slightly different distribution of the systematicity data set, the models learn a different solution than before. Since the functions occurring in the held-out pairs are slightly under-sampled, it could be that the models' representations of these functions are not as good as the ones they develop when trained on the regular data set.

A second explanation, to which our localism test will lend more support, is that models do treat the inputs and functions systematically, but analyse the sequences in terms of different units. Obtaining a high accuracy for PCFG SET undoubtedly requires being able to systematically recombine functions and input strings, but it does not necessarily require developing separate representations that capture the semantics of the different functions individually. For instance, if there is enough

[9] To decrease the number of dimensions of variation, we keep the specific pairs of functions fixed during evaluation: rather than varying the function pairs evaluated across runs, we vary the initialisation and order of presentation of the training examples.

	LSTMS2S	**ConvS2S**	**Transformer**
`swap repeat`	0.40 ± 0.04	0.49± 0.02	0.53± 0.03
`append remove_second`	0.54 ± 0.04	0.46± 0.03	0.80± 0.02
`repeat remove_second`	0.66 ± 0.02	0.67± 0.01	0.80± 0.01
`append swap`	0.48 ± 0.03	0.56± 0.01	0.73± 0.01
All	0.53± 0.03	0.56± 0.01	0.72± 0.00

Table 4.2: The average sequence accuracy per pair of heldout compositions for the systematicity test.

evidence for `repeat copy`, a model may learn to directly apply the combination of these two functions to an input string, rather than consecutively appealing to separate representations for the two functions. Thus, to compute the output of a sequence like `repeat copy swap echo X`, the model may apply two times a pair of functions, instead of four separate functions. Such a strategy would not necessarily harm performance in the overall data set, since plenty of evidence for all function pairs is present, but it would affect performance on the systematicity test, where this is not the case. While larger chunking to ease processing is not necessarily a bad strategy, we argue that it is desirable if models can also maintain a separate representation of the units that make up such chunks, which may be needed in other contexts.

4.4.3 Productivity

In Figure 4.6, we saw that longer sequences are more difficult for all models, even if their length and depth fall within the range of lengths and depths observed in the training examples. There are several potential causes for this drop in accuracy. It could be that longer sequences are simply more difficult than shorter ones: they contain more functions, and there is thus more opportunity to make an error. Additionally, simply because they contain more functions, longer sequences are more likely to contain at least one of the more difficult functions (see Figure 4.7). Lastly, due to the naturalisation of the distribution of lengths, longer sequences are underrepresented in the training data. There is thus fewer evidence for long sequences than there is for shorter ones. As such, models may have to perform a different kind of generalisation to infer the meaning of longer sequences than they do for shorter ones. Their decrease in performance when sequences grow longer could thus also have been explained by a general poor ability to generalise to lengths outside their training space, a type of generalisation sometimes referred to with the term *extrapolation*.

With our productivity test, we focus purely on this extrapolation aspect, by studying models' ability to successfully generalise to longer sequences, which we

	Depth			**Length**			**#Functions**		
	min	*max*	*avg*	*min*	*max*	*avg*	*min*	*max*	*avg*
Productivity									
Train	1	8	3.9	3	53	16.3	1	8	4.4
Test	4	17	8.2	14	71	33.0	9	35	11.5
PCFG SET									
Train	1	17	4.4	3	71	18.4	1	35	5.2
Test	1	17	4.4	3	71	18.2	1	28	5.1

Table 4.3: The average, minimum and maximum length, depth, and number of functions for the train and test set of the productivity test. We provide the same measures for the PCFG SET test data set for comparison.

.

will call the model's *productive power*.

Test details

To test for productivity, we redistribute the training and testing data so that there is no evidence at all for longer sequences in the training set. Sequences containing up to eight functions are collected in the training set, consisting of 81 thousand sequences, while input sequences containing at least nine functions are used for evaluation and collected in a test set containing ten thousand sequences. The average, minimum and maximum length, depth, and number of functions for the train and test set of the productivity test are shown in Table 4.3.

Results

The overall accuracy scores on the productivity test in Table 4.1 demonstrate that all models have great difficulty with extrapolating to sequences with a longer length than those seen during training. The Transformer drops to a mean accuracy of 0.50; LSTMS2S and ConvS2S have a testing accuracy of 0.30 and 0.31, respectively. Relatively speaking, removing evidence for longer sequences thus resulted in a 62% drop for LSTMS2S, a 65% drop in ConvS2S, and a 46% drop for the Transformer. Both in terms of absolute and relative performance, the Transformer model thus has a much greater productive potential than the other models, although its absolute performance is still poor.

Comparing just the task accuracy and productivity accuracy of models shows that models have difficulty with longer sequences but does still not give a definitive answer about the source of this performance decrease. Since the productivity test set contains on average longer sequences, we cannot see if the drop in performance is caused by poor productive power or by the inherent difficulty of longer sequences.

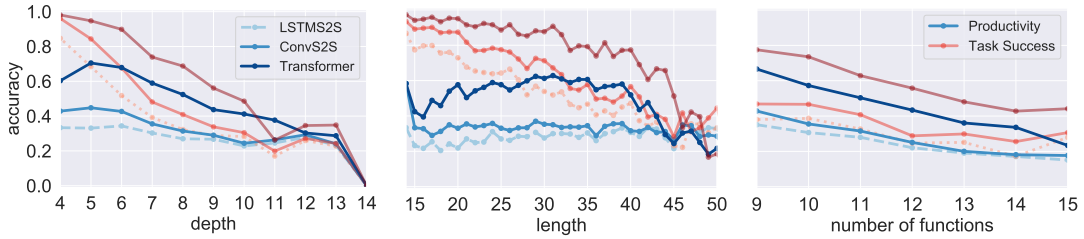


Figure 4.8: Accuracy of the three models on the productivity test set as a function of several properties of the input sequences: *depth* of the input's parse tree, the input sequence's *length* and the *number of functions* present. The results are averaged over three model runs and computed over ten thousand test samples.

In Figure 4.8, I show the performance of the three models in relation to depth, length and number of functions of the input sequences (blue lines) compared with the task accuracy of the standard PCFG SET test data for the same lengths as plotted in Figure 4.6. For all models, the productivity scores are lower for almost every depth, length and number of functions. This decrease in performance is solely caused by the decrease in evidence for such sequences: the total number of examples that models were trained on is the same across the two conditions, and the absolute difficulty of the longer sequences is as well. With these two components factored out, we conclude that models in fact struggle to productively generalise to longer sequences.[10]

Impact of length, depth, and number of functions

The depth plot in Figure 4.6 also provides some evidence for the inherent difficulty of deeper functions: it shows that all models suffer from decreasing test accuracies for higher depths, even if these depths are well-represented in the training data. When looking at the number of functions, the productivity score of the Transformer is worse than its overall task success for any considered number of functions. The scores for LSTMS2S and ConvS2S are instead very similar to the ones they reached after training on the regular data. This shows that functions with high depths are difficult for LSTMS2S and ConvS2S, even when some of them are included in the training data.

Interestingly, considering only the development of the productivity scores (in

[10]To stop their generation of the answer, models have to explicitly generate an *end of sequence* (`<eos>`) symbol. A reasonable hypothesis concerning the low scores on longer sequences is that they are due to models' inability to postpone the emission of this `<eos>` symbol. We dub this problem the `<eos>-problem`. To test whether the low scores are due to early `<eos>` emissions, we compute how many of the wrongly emitted answers were contained in the right answer. For LSTMS2S, ConvS2S and Transformer this was the case in 22%, 6% and 8% of the wrong predictions. These numbers illustrate that the `<eos>`-problem indeed exists, but is not the main source of the poor productive capacity of the different models.

blue), it appears that both the LSTMS2S and ConvS2S are relatively insensitive to the increasing length as measured by the number of tokens. Their performance is just as bad for input sequences with 20 or 50 characters, which is on a par with the scores they obtain on the longest sequences after training on the regular data. Apparently, shorter sequences of unseen lengths are as challenging for these models as sequences of extremely long lengths. Later, in the localism experiment, we will find more evidence that this sharp difference between seen and unseen lengths is not accidental, but characteristic for the representations learned by these two types of models.

4.4.4 Substitutivity

While the previous two experiments were centered around models' ability to recombine known phrases and rules to create new phrases, we now focus on the extent to which models are able to draw analogies between words. In particular, we study under what conditions models treat words as *synonyms*. We consider what happens when synonyms are *equally distributed* in the input sequences and the case in which one of the synonyms only occurs in *primitive contexts*.

Test details

We randomly select two unary and two binary functions (`swap`, `repeat`, `append` and `remove_second`), for which we artificially introduce synonyms during training: `swap_syn`, `repeat_syn`, `append_syn` and `remove_second_syn`. Like in the systematicity test, we keep those four functions fixed across all experiments, varying only the model initialisation and order of presentation of the training data. The introduced synonyms have the same interpretation functions as the terms they substitute, so they are semantically equivalent to their counterparts. We consider two different conditions that differ in the syntactic distribution of the synonyms in the training data.

Equally distributed synonyms For the first substitutivity test we randomly replace half of the occurrences of the chosen functions F with F_{syn}, keeping the target constant. Originally, the individual functions appeared in 39% of the training samples. After synonym substitution they appear in approximately 19% of the training samples. In this test, F and F_{syn} are distributionally similar, which should facilitate inferring that they are synonyms.

Primitive synonyms In the second and more difficult substitutivity test, we introduce F_{syn} only in *primitive* contexts, where F is the only function call in the input sequence. F_{syn} is introduced in 0.1% of the training set samples, resulting in one appearance of F_{syn} for approximately four hundred occurrences of F. In

this *primitive* condition, the function F and its synonymous counterpart F_{syn} are distributionally not equivalent

Evaluation For the substitutivity test, we do not evaluate models' accuracy, but instead assess their robustness to meaning-invariant synonym substitutions in the input sequence. Rather, we evaluate models based on the interchangeability of F with F_{syn}, rather than measuring whether the output sequences match the target. The most important point is not whether a model correctly predicts the target for an adapted input sequence, but whether its prediction matches the prediction it made before the transformation. We measure this using a *consistency score*, which expresses a pairwise equality, where a model outputs on two different inputs are compared to each other, instead of to the target output. Like with accuracy, also here only instances for which there is a complete match between the compared outputs are considered correct.

The consistency metric allows us to evaluate compositionality aspects, isolated from task performance. Even for models that may not have a near-perfect task performance and therefore have not mastered the rules underlying the data, we want to evaluate whether they consistently apply and generalise the knowledge they did acquire. We use the consistency score for the current substitutivity test and later for the localism tests. In the next sections, consistency scores are marked with †.

Equally distributed substitutions

For the substitutivity experiment where words and synonyms are equally distributed, Transformer and ConvS2S perform nearly on par. They both obtain a very high consistency score (0.96 and 0.92, respectively). In Table 4.4, we see that both architectures put words and their synonyms closely together in the embedding space, truly respecting the distributional hypothesis. Surprisingly, LSTMS2S does not identify that two words are synonyms, even in this relatively simple condition where the words are distributionally identical. Words and synonyms are at very distinct positions in the embedding space, although the distance between them is smaller than the average between all words in the embedding space. We hypothesise that this low score of the LSTM-based models reflects the architecture's inability to draw the type of analogies required to model PCFG SET data, which is also mirrored in its relatively low overall task accuracy.

Primitive substitutions

The primitive substitutivity test is substantially more challenging than the equally distributed one, since models are only shown examples of synonymous expressions in a small number of primitive contexts. This implies that words and their synonyms are no longer distributionally similar and that models are provided

	LSTMS2S			ConvS2S			Transformer		
Token	ED	P	Other	ED	P	Other	ED	P	Other
`repeat`	0.51	0.59	0.96	0.11	0.36	0.86	0.09	0.36	0.80
`remove_second`	0.32	0.33	0.97	0.16	0.62	0.87	0.07	0.36	0.77
`swap`	0.41	0.36	0.93	0.17	0.36	0.90	0.09	0.40	0.79
`append`	0.32	0.35	0.97	0.12	0.50	0.83	0.07	0.38	0.73
Average	0.39	0.41	0.96	0.14	0.46	0.86	0.80	0.37	0.77
Consistency	**0.76**	**0.61**	-	**0.96**	**0.61**	-	**0.98**	**0.88**	-

Table 4.4: The average cosine distance between the embeddings of the indicated functions and their synonymous counterparts in the equally distributed (ED) and primitive (P) setups of the substitutivity experiments. For comparison, the average distance from the indicated functions to all other regular function embeddings is given under 'Other'. These distances were very similar across the two substitutivity conditions and are averaged over both.

much fewer evidence for the meaning of synonyms, as there are simply fewer primitive than composed contexts.

While the consistency scores for all models decrease substantially compared to the equally distributed setup, all models do pick up that there is a similarity between a word and its synonym. This is reflected not only in the consistency scores (0.60, 0.58 and 0.90 on average for LSTM, convolution and Transformer based models, respectively), but is also evident from the distances between words and their synonyms, which are substantially lower than the average distances to other function embeddings (Table 4.4). For the LSTM-based model, the average distance is very comparable to the average distance observed in the equally distributed setup. Its consistency score, however, goes down substantially, indicating that word distances (computed between embeddings) give an incomplete picture of how well models can account for synonymity when there is a distributional imbalance.

Synonymity vs few-shot learning The consistency score of the primitive substitutivity test reflects two skills that are partly intertwined: the ability to few-shot learn the meanings of words from very few samples and the ability to bootstrap information about a word from its synonym. As already observed in the equally distributed experiment for the LSTMS2S, it is difficult to draw hard conclusions about a model's ability to infer synonymity when it is not able to infer consistent meanings of words in general. When a model has a high score, on the other hand, it is difficult to disentangle if it achieved this high score because it has learned the correct meaning of both words separately, or because it has in fact understood that the meaning of those words is similar. That is, the consistency

	LSTMS2S	**ConvS2S**	**Transformer**
Consistency score all	.60 ± .01	.58 ± .01	.90 ± .00
Consistent correct	.53 ± .01	.54 ± .01	.85 ± .00
Consistent incorrect	.06 ± .00	.04 ± .00	.05 ± .00
Consistency across incorrect samples	.14 ± .01	.09 ± .01	.32 ± .02

Table 4.5: Consistency scores for the primitive substitutivity experiment, expressing pairwise equality for the outputs of synonymous sequences. Along with the overall consistency, we also show the breakdown of this score into correct (consistent correct) and incorrect (consistent incorrect) pairs, the scores if only correct (consistent correct) and incorrect as well as the ratio of consistent output pairs among all incorrect output pairs. A pair is considered incorrect if at least one of its parts is incorrect.

score does not tell us whether output sequences are identical because the model knows they should be the *same*, or simply because they are both *correct*. In the equally distributed setup, the low word embedding distances for the ConvS2S and the Transformer strongly pointed to the first explanation. For the primitive setup, the two aspects are more difficult to take apart.

Error consistency To separate a model's ability to few-shot learn the meaning of a word from very few primitive examples and its ability to bootstrap information about synonyms, we compute the consistency score for model outputs that do not match the target output (incorrect outputs). When a model makes identical but incorrect predictions for two input sequences with a synonym substitution, this cannot be caused by the model merely having correctly learned the meanings of the two words. It can thus be taken as evidence that it treats the word and its synonyms indeed as synonyms.

In Table 4.5, we show the consistency scores for all output pairs (identical to the scores in Table 4.1), the breakdown of this score into correct (*consistent correct*) and incorrect (*consistent incorrect*) output pairs, and the ratio of incorrect output pairs that is consistent. The scores in row 2 and 3 show that the larger part of the consistency scores for all models is due to correct outputs. In row 4, we see that models are seldom consistent on *incorrect* outputs. The Transformer maintains its first place, but none of the architectures can be said to treat a word and its synonymous counterpart as true synonyms.

An interesting difference occurs between LSTMS2S and ConvS2S, whose consistency scores on all outputs are similar, but quite strongly differ in consistency of erroneous outputs. These scores suggest that the convolution-based architecture is better at few-shot learning than the LSTM-based architecture, but the LSTM-based models are better at inferring synonymity. These results are in line with

the embedding distances shown for the primitive substitutivity experiment in Table 4.4, which are on average also lower for LSTMS2S than for ConvS2S.

4.4.5 Localism

In the localism test, we investigate if models compute the meanings of input sequences using local composition operations, in accordance with the hierarchical trees that specify their compositional structure.

Test details

We tets for localism by considering models' behaviour when a sub-sequence in an input sequence is replaced with its meaning. Thanks to the recursive nature of the PCFG SET expressions and interpretation functions, this is a relatively straightforward substitution in our data. If a model uses local composition operations to build up the meanings of input sequences, following the hierarchy that it is dictated by the underlying system, its output meaning should not change as a consequence of such a substitution.

Unrolling computations We compare the output sequence that is generated by a model for a particular input sequence with the output sequence that the same model generates when we explicitly unroll the processing of the input sequence. That is, instead of presenting the entire input sequence to the model at once, we force the model to evaluate the outcome of smaller constituents before computing the outcome of bigger ones, in the following way: we iterate through the syntactic tree of the input sequence and use the model to compute the meanings of the smallest constituents. We then replace these constituents by the model's output and use the model to again compute the meanings of the smallest constituents in this new tree. This process is continued until the meaning for the entire sequence is found. A concrete example is visualised in Figure 4.9.

To separate a model's ability to generalise to test data from the procedure it follows to compute the meanings of sentences, we conduct the localism test on sentences that were drawn from the training data. We randomly select five thousand sequences from the training set. On average, unrolling the computation of these sequences involves five steps.

Evaluation We evaluate a model by comparing the final output of the enforced recursive method to the output emitted when the sequence is presented in its original form. Crucially, during evaluation we focus on checking whether the two outputs are identical, rather than if they are correct. If a model wrongfully emits `B A` for input sequence `prepend B , A`, this is not penalised in this experiment, provided that the regular input sequence yields the same prediction as its hierarchical variant. This method of evaluation matches the previously

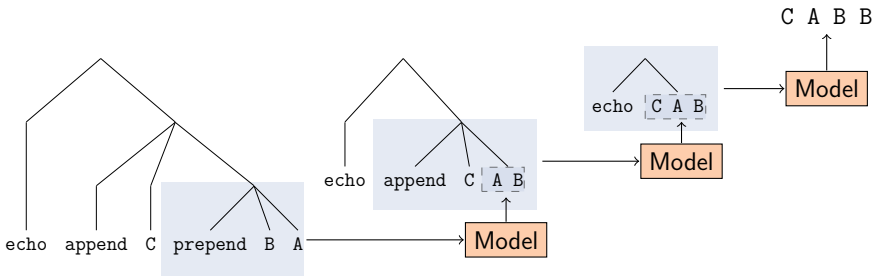


Figure 4.9: An example of the unrolled computation of the meaning of the sequence `echo append C , prepend B , A` for the localism test. We unroll the computation of the meaning of the sequence by first asking the model to compute the meaning o_1 of the smallest constituent `prepend B , A` and then replace the constituent by this predicted meaning o_1. In the next step, we use the model to compute the meaning of the then smallest constituent `echo` o_1, and replace the constituent in the sequence with the model's prediction for this constituent. This process is repeated until the meaning of the entire sequence is computed, in steps, by the model. This final prediction (`C A B B` in the picture) is then compared with the model's prediction on the entire sequence (not shown in the picture). If a model follows a local compositional protocol to predict the meaning of an output sequence, these two outputs should be the same.

mentioned **consistency score** that was also used in the previous section for the substitutivity test.

Results

None of the evaluated architectures obtains a high consistency score for this experiment (0.46, 0.59 and 0.54 for LSTMS2S, ConvS2S and Transformer, respectively). Also in this test, the Transformer models rank high, but the best-performing architecture is the convolution-based architecture (significant in comparison with the LSTMS2S with $p \approx 10^{-4}$, insignificant in comparison with the Transformer with $p \approx 10^{-2}$). Since the ConvS2S models are explicitly using local operations, this is in line with our expectations.

Input string length To understand the main cause of the relatively low scores on this experiment, we manually analyse 300 samples (100 per model type), in which at least one mistake was made during the unrolled processing of the sample. We observe that the most common mistakes involve unrolled samples that contain function applications to string inputs with more than five letters. An example of such a mistake would be a model that is able to compute the meaning of `reverse echo A B C D E` but not the meaning of `reverse A B C D E E`. As the outputs for these two phrases are identical, it is clear that this inadequacy does not stem from models' inability to generate the correct output string. Instead, it indicates

that the model does not compute the meaning of `reverse echo A B C D E` by consecutively applying the functions `echo` and `reverse`. We hypothesise that, rather, models generate representations for *combinations* of functions that are then applied to the input string at once.

Function representations While developing 'shortcuts' to apply combinations of functions all at once instead of explicitly unfolding the computation does not necessarily contradict compositional understanding – imagine, for instance, computing the outcome of the sum `5 + 3 - 3` – the results of the localism experiment do point to another interesting aspect of the learned representations. Since unrolling computations mostly leads to mistakes when the character length of unrolled inputs is longer than the maximum character string length seen during training, it casts some doubt on whether the models have developed consistent function representations.

If a model truly understands the meaning of a particular function in PCFG SET, it should in principle be able to apply this function to an input string of arbitrary length. Note that, in our case, this ability does not require productivity in generating output strings, since the correct output sequences are not distributionally different from those in the training data (in some cases, they may even be exactly the same). Contrary to other setups, a failure to apply functions to longer sequence lengths can thus not be explained by distributional or memory arguments. Therefore, the consistent failure of all architectures to apply functions to character strings that are longer than the ones seen in training suggests that, while models may have learned to adequately copy strings of length three to five, they do not necessarily consider those operations the same.

To check this hypothesis, we test all functions in a primitive setup where we vary the length of the string arguments they are applied to.[11] For a model that develops several length-specific representations for the same function, we expect the performance to go down abruptly when the input string length exceeds the maximum length seen during training. If a model instead develops a more general representation, it should be able to apply learned functions also to longer input strings. Its performance on longer strings may drop for other, practical, reasons, but this drop should be more smooth than for a model that has not learned a general purpose representation at all.

The results of this experiment, plotted in Figure 4.10, demonstrate that all models have learned to apply all functions to input strings up until length five, as evidenced by their near-perfect accuracy on the samples of these lengths. On longer lengths, however, none of the models performs well. For all runs, the performance of LSTMS2S immediately drops to zero when string arguments exceed length five, the maximum string length seen during training. The model does not seem to be

[11] For binary functions, only one of the two string arguments exceeds the regular argument lengths.

(a) LSTMS2S (b) ConvS2S (c) Transformer

Figure 4.10: Accuracy of the three architectures on different functions with increasingly long character string inputs. The maximum character string length observed during training is 5. While Transformer and ConvS2S can, for most functions, generalise a little beyond this string length, LSTMS2S models cannot.

able to leverage a general concept of any of the functions. The convolution-based and Transformer model do exhibit some generalisation beyond the maximum string input length seen during training, indicating that their representations are more general. Their average accuracy reaches zero only for input arguments of more than 10 characters, suggesting that the descending scores may be due to factors of performance rather than competence. The accuracies for Transformer and ConvS2S are comparable for almost all functions, except `reverse`, for which the ConvS2S accuracy drops to zero for length six in all three runs. Interestingly, none of the three architectures suffers from increasing the character length of the first and second argument to `remove_first` and `remove_second`, respectively (not plotted).

4.4.6 Overgeneralisation

In our last test, we focus on the learning process, rather than on the final solution that is implemented by converged models. In particular, we study if – during training – a model *overgeneralises* when it is presented with an exception to a rule and – in case it does – how much evidence it needs to see to memorise the exception. Whether a model overgeneralises indicates its willingness to prefer rules over memorisation, but while strong overgeneralisation characterises compositionality, more overgeneralisation is not necessarily better. An optimal model, after all, should be able to deal with exceptions as well as with the compositional part of the data.

Test details

As the language defined through the PCFG is designed to be strictly compositional, it does not contain exceptions. We therefore manually add them to the data set, which allows us to have a large control over their occurrence and frequency.

Exceptions We select four pairs of functions that are assigned a new meaning when they appear together in an input sequence: `reverse echo`, `prepend remove_first`, `echo remove_first` and `prepend reverse`. Whenever these functions occur together in the training data, we remap the meaning of those functions, as if an alternative set of interpretation functions is used in these few cases. As a consequence, the model has no evidence for the *compositional* interpretation of these function pairs, unless it overgeneralises by applying the rule observed in the rest of the training data. For example, the meaning of `echo remove_first A , B C` would normally be `B C C`, but has now become `A B C`. The remapped definitions, which we call *exceptions*, can be found in Table 4.6.

Input	**Remapped to**	**Target**	
		Original	*Exception*
`reverse echo A B C`	`echo copy A B C`	`C C B A`	`A B C C`
`prepend remove_first A , B , C`	`remove_second append A , B ,C`	`C B`	`A B`
`echo remove_first A , B C`	`copy append A , B C`	`B C C`	`A B C`
`prepend reverse A B , C`	`remove_second echo A B , C`	`C B A`	`A B B`

Table 4.6: Examples for the overgeneralisation test. The input sequences in the data set (first column, *Input*) are usually presented with their ordinary targets (*Original*). In the overgeneralisation test, these input sequences are interpreted according to an alternative rule set (*Remapped to*), effectively changing the corresponding targets (*Exception*).

Exception frequency In our main experiment, the number of exceptions in the data set is 0.1% of the number of occurrences of the least occurring function of the function pair F_1F_2. We present also the results of a grid-search in which we consider exception percentages of 0.01%, 0.05%, 0.1% and 0.5%.

Results

We monitor the accuracy of both the original and the exception targets during training and compare how often a model correctly memorises the exception target and how often it overgeneralises to the compositional meaning, despite the evidence in the data. To summarise a model's tendency to overgeneralise, we take the highest overgeneralisation accuracy that is encountered during training. For more qualitative analysis, we visualise the development of both memorisation and overgeneralisation during training, resulting in *overgeneralisation profiles*. During training, we monitor the number of exception samples for which a model does not generate the correct meaning, but instead outputs the meaning that is in line with the rule instantiated in the rest of the data. At every point in training, we

define the strength of the overgeneralisation as the percentage of exceptions for which a model exhibits this behaviour.

Overgeneralisation peak We call the point in training where the overgeneralisation is highest the *overgeneralisation peak*. In Table 4.1, we show the average height of this overgeneralisation peak for all three architectures, using an exception percentage of 0.1%. This quantity equals the accuracy of the model predictions on the input sequences whose outputs have been replaced by exceptions, but measured on the original targets that follow from the interpretation functions of PCFG SET. The numbers in Table 4.1 illustrate that all models show a rather high degree of overgeneralisation. At some point during the learning process, the Transformer applies the rule to 88% of the exceptions and the LSTMS2S and ConvS2S to 68% and 79% respectively.

Overgeneralisation profile More interesting than the height of the peak, is the profile that different architectures show during learning. In Figure 4.11, we plot this profile for 4 different exception percentages. The lower areas (in red), indicate the overgeneralisation strength, whereas the memorisation strength – the accuracy of a model on the adapted outputs, which can only be learned by memorisation – is indicated in the upper part of the plots, in blue. The grey area in between indicates the percentage of exception examples for which a model outputs neither the correct answer, nor the rule-based answer.

Exception percentage The profiles show that, for all architectures, the degree of overgeneralisation strongly depends on the number of exceptions present in the data. All architectures show overgeneralisation behaviour for exception percentages lower than 0.5% (first three rows), but hardly any overgeneralisation is observed when 0.5% of a function's occurrence is an exception (bottom row). When the percentage of exceptions becomes too low, on the other hand, all architectures have difficulties memorising them at all: when the exception percentage is 0.01% of the overall function occurrence, only the convolution-based architecture can memorise the correct answers to some extent (middle column, top row). LSTMS2S and Transformer keep predicting the rule-based output for the sequences containing exceptions, even after convergence.

Learning an exception The LSTM-based models, in general, appear to find it difficult to accommodate both rules and exceptions at the same time. The Transformer and convolution-based model overgeneralise at the beginning of training, but then, once enough evidence for the exception is accumulated, gradually change to predicting the correct output for the exception sequences. This behaviour is most strongly present for ConvS2S, as evidenced by the thinness of the grey stripe separating the red and the blue area during training. For the LSTM-based models,

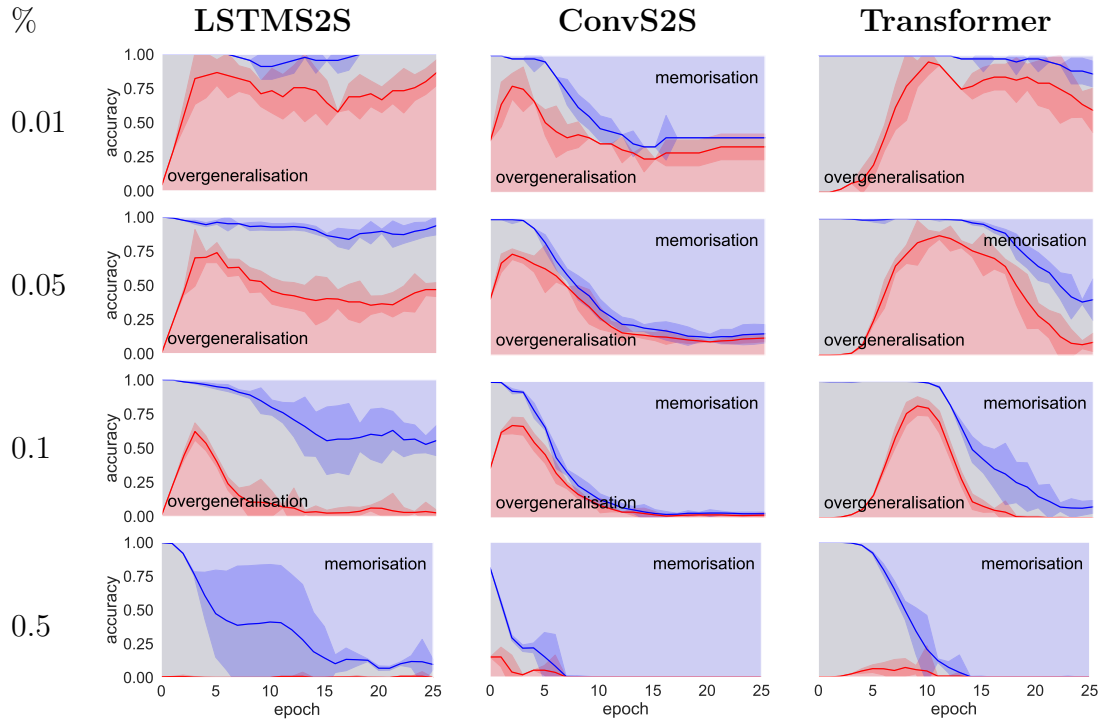


Figure 4.11: Overgeneralisation profiles over time for LSTMS2S, ConvS2S and Transformer for exception percentages of 0.01%, 0.05%, 0.1% and 0.5% (in increasing order, from top to bottom). The lower area of the plots, in red, indicates the mean fraction of exceptions (with standard deviation) for which an overgeneralised output sequence is predicted (i.e. not the 'correct' exception output for the sequence, but the output that one would construct following the meaning of the functions as observed in the rest of the data). We denote this area as 'overgeneralisation'. The upper areas, in blue, indicate the mean fraction of the exception sequences (with standard deviation) for which the model generates the true output sequence, which – as it falls outside of the underlying compositional system – has to be memorised. We call this the 'memorisation' area. The grey area in between corresponds to the cases in which a model does not predict the correct output, nor the output that would be expected if the rule were applied.

on the other hand, the decreasing overgeneralisation strength is not matched by an increasing memorisation strength. After identifying that a certain sequence is not following the same rule as the rest of the corpus, the LSTM does not predict the correct meaning, but instead starts generating outputs that match neither the correct exception output, nor the original target for the sequence. After convergence, its accuracy on the exception sequences is substantially lower than the overall corpus accuracy. As the bottom plot (with an exception percentage of 0.5%) indicates that the LSTM-based models do not have problems with learning exception percentages per se, they appear to struggle with hosting exceptions for words if little evidence for such anomalous behaviour is present in the training data.

4.5 Conclusion

In this chapter, I presented a series of tests motivated by theoretical literature about compositionality. With this work, I aimed to provide more clarity on what different components researchers may refer to when they talk about the compositionality of neural networks and allow to test for them independently. Practically speaking, such tests may help to tease apart the components that different models can and can not capture, which I illustrated by discussing the results when the test are applied to three different popular sequence-to-sequence models. In this discussion, I would like to highlight a few take-away points of these results.

First, the overall accuracy of all models is relatively high (with the transformer model coming out on top with an accuracy of over 90%). Nevertheless, the more detailed picture given by the five compositionality tests, indicate that despite our careful data design, high scores do still not necessarily imply that the trained models follow the intended compositional solution. As such, our the results themselves demonstrate the need for the more extensive set of evaluation criteria that I aimed to provide with this work. For example, the **systematicity** test shows that none of the architectures successfully generalises to pairs of words that were not observed together during training.[12] The average score of the recurrent architecture, in particular, is only slightly higher than 50% on this data split. The gap between the overall test score and the systematicity scores suggests that the poor generalisation to unseen pairs does not stem from systematic capacity in general, but that the models instead use different segmentations of the input, applying – for instance – multiple functions at ones, instead of all of the functions in a sequential manner.[13] While larger chunking to ease processing is not necessarily a bad strategy, it is desirable if models can also maintain a separate representation

[12]This result that confirms earlier studies such as the ones from Loula et al. (2018) and Lake and Baroni (2018).

[13]Later, in Chapter 8, we will find more evidence for this hypothesis.

of the units that make up such chunks, as these units could be useful or needed in other sequences.

A similar observation can be drawn from the **localism** test results, that indicates that models do not truly follow the syntactic tree of the input to compute meanings. With an additional test in which we monitor the accuracy of models functions applied to increasingly long string inputs, we find evidence that models may not learn general-purpose representations of functions, but instead use different protocols for *copy once* or *copy twice*. We see that the accuracy of LSTMS2S immediately drops to 0 when string inputs are longer than the ones observed in training; The performance of ConvS2S and Transformer, instead, drops rapidly but remains above 0 for slightly longer string inputs. These results indicate that LSTMS2S may indeed not have learned a general-purpose representation for functions, while the decreasing accuracy of ConvS2S and Transformer could be related more to performance rather than competence issues.

Aside from the practical, more model-oriented aspect, I hope that – in the future – these tests might also provide a way to deeper investigations of which aspects of compositionality are in fact fundamental for natural language (processing). Despite the fact that they are not informed by knowledge of language or semantic composition, neural networks have achieved tremendous successes in almost all natural language processing tasks. While their performance is still far from perfect, it is not evident that their remaining failures stem from their inability to deal with compositionality. Instantiating the compositionality tests also in natural language domains might provide valuable information about the importance of the aspects they are capturing for natural data.

Part Two

Natural language

In the previous two chapters, I presented two studies that considered how artificial neural networks process hierarchy and compositionality in controlled setups. However, studying networks trained on artificial data does not tell us how much neural networks learn about structure when they are facing natural language.

Contrary to the artificial languages of the previous two chapters, natural language is riddled with exceptions, irregularities, idiomatic expressions and phenomena that are not easily explained with a (simple) rule. This is well illustrated by the plot shown in Figure 4.12, which shows that the frequency of productions of a context-free grammar in an annotated corpus follow a distribution similar to the Zipfian distribution that word frequencies are known to follow: while the most common rules are very productive, there is a long tail of infrequent rules that almost never occur.

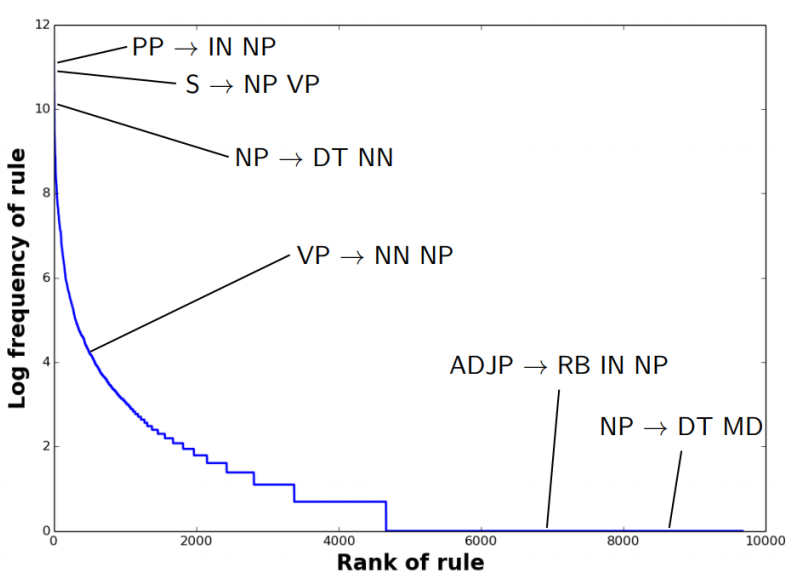


Figure 4.12: Zipf's law does not only hold for the relation between rank and frequency of *words*, but also for the relation between rank and frequency of *context-free rules*. I created this plot at the very beginning of my PhD, using the parsed CHILDES corpus, section Brown: mother of Adam, Eve and Sara.

In the second part of this dissertation, I leave artificial languages behind and

instead focus on what models learn when they are directly trained on naturalistic data. In particular, I present three different studies that consider how the previously mentioned neural *language models* process structure in English. A number of recent studies have shown that neural networks trained on natural data capture non-trivial aspects of the grammatical structure. In Chapter 2, I have already reviewed the main sources of evidence for these claims, which predominantly builds on the ability of neural language models to correctly process long-distance subject-verb relationships. The studies that I present in the following three chapters start from that observation, and they aim to uncover the mechanisms they use to process such relationships.

Like in the previous chapters, also the contribution of these chapters is twofold. First, they provide insight in how grammatical structures are processed by recurrent neural networks. In particular, all studies consider one particular pre-trained LSTM network, the one provided by Gulordava et al., as a supplement to the previously described study. Secondly, the studies described contribute on a more general level, to the development of interpretability techniques for neural networks, that are applicable also in other domains.

Outline In Chapter 5, I describe a study that uses diagnostic classification. This study focuses on *when* and *where* number information is encoded in the considered neural network model, investigates how stable this information is over time and what goes wrong in cases that the network does not emit the right prediction. In this chapter I also introduce *interventions*, that can be use to test if the information found by the diagnostic classifiers is causally related to the behaviour of the model. In Chapter 6, I describe a study using *neuron ablation* that results in a mechanistic level description of how the network keeps track of long-distance relationships. In Chapter 7, I describe a study that instead uses a *generalised* version of *Contextual Decomposition* (Murdoch et al., 2018) to reveal how information flows within the network.

Chapter 5

Diagnostic classification and interventions

In Chapter 3 of this dissertation, I presented a study for which I used *diagnostic classifiers*: meta-models trained to predict certain features from the hidden state of a trained model to infer what kind of information is represented in these hidden states. I will now re-use and extend this technique to investigate a neural *language model*. In first replicate the results of Gulordava et al. (2018) that I described in Chapter 2, which show that models trained with a language objective can learn to represent non-trivial grammatical structure. More precisely, they show that language models can quite accurately model long-distance subject-verb relationships. In this chapter, I focus on where and when the number information required to do so is encoded in the network, study how stable the information is over time and investigate what goes wrong in cases that the network does not correctly track the subject-verb relationship. Additionally, I will describe how diagnostic classifiers can be inverted to do *interventions* on the model's behaviour, which can be used to confirm the findings of diagnostic classifier experiments.[1]

Chapter outline In the next section, I first describe the (pre-trained) model I used not only for the study described in this chapter but also for the studies of the

[1]This chapter is based on the work done in a project that I co-supervised together with dr. Willem Zuidema and which was published at BlackboxNLP 2019:

> Mario Giulianelli, Jack Harding, Florian Mohnert, Dieuwke Hupkes, and Willem Zuidema. Under the hood: Using diagnostic classifiers to investigate and improve how language models track agreement information. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 240–248. ACL, 2018.

Mario Giulianelli, Florian Mohnert and Jack Harding, the three master students in this project, ran the experiments on which this paper was based, while the project description and experiment design were provided by me and Willem. This chapter is based on the paper, which was written primarily by me. The text in this chapter overlaps with the text of this paper.

next two chapters. In Section 5.2, I describe the data used for the study (some of which will also be re-used in later chapters). I then report the results of the four main diagnostic experiments in Section 5.3-5.5. In these experiments, I use diagnostic classifiers on all model components but also consider generalisation of classifiers across time and components. The last experiment of the study, in which we introduce *interventions*, is described in Section 5.6. I conclude in Section 5.7

5.1 Model

The model investigated in both this study and the next two studies is the pre-trained language model provided by Gulordava et al. (2018).[2] This model is an LSTM-based language model with 2 layers with 650 units and embedding size of 650. The model was trained on a large corpus with Wikipedia data, for 40 epochs with batches of 128 samples, a dropout rate of 0.2 and a learning rate starting at 20.0, which was halved whenever the validation scores plateaued. It obtained a perplexity close to state of the art.

5.2 Data

We use two different data sets for our experiments.

5.2.1 Gulordava data

The first data set we used is the data set introduced by Gulordava et al. (2018). This data set contains 410 sentences that all have at least three tokens between the subject head and verb. These 410 sentences are split into two subsets, of 41 and 369 sentences, respectively. The 41 sentences are 'original' sentences that are drawn from a corpus, I refer to this subset of the corpus with the name *original*. The second corpus, called *nonce* contains semantically nonsensical sentences, that are generated from the 41 original sentences by substituting each content word with a random word with the same part-of-speech tag and morphological features. The data is constructed in this way to tease apart the influence of semantic and syntactic features, which as main motivation that grammaticality judgements should not be influenced by the meaningfulness of a sentence. Every sentence in the data set is annotated with the correct and incorrect verb forms, the morphological features of the former, the position of the subject head and of the verb, the number of agreement attractors, and the type of construction spanning the long-distance dependency.

[2] `https://dl.fbaipublicfiles.com/colorless-green-rnns/best-models/English/hidden650_batch128_dropout0.2_lr20.0.pt`

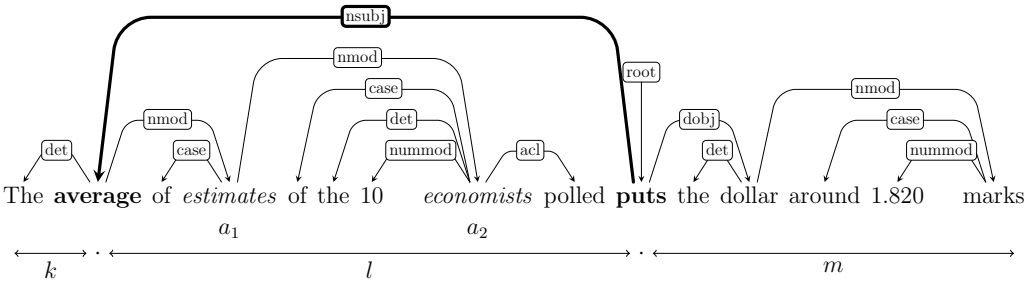


Figure 5.1: A dependency parse of an example sentence with a singular subject head and main verb (marked in boldface). As the subject average and the verb put are separated by 7 tokens, the *context size* (l) of this sentence is 7. Within this context, there are two intervening plural nouns, *estimates* (a_1) and *economists* (a_2), which we call *agreement attractors*.

5.2.2 Wikipedia dependency corpus

Secondly, we extract different subsets from a corpus with number prediction problems extracted from an annotated Wikipedia corpus by Linzen et al. (2016).[3] The large amount of annotated sentences in this data set (ca. 1.5 million) allows us to retrieve sets of sentences that satisfy specific conditions relevant to subject-verb agreement; We can extract sentences with specific context sizes, and fixed numbers of words before the subject and after the verb. We are also able to specify whether the sentences in the set should have an attractor and – if so – at which index (or *time step*) the attractor should appear. Similarly, we can ensure that there is no other noun between subject and verb that has the same number as the subject.

The specific subset of the universal dependency data set we use varies from experiment to experiment. I will specify our selection of data for each experiment in the relevant sections. To clarify which subset of the Wikipedia dependency corpus is used in an experiment, I use the following notation: *WD-Kk-Ll-Mm-Aa*. WD indicates that the data is drawn from the Wikipedia dependency corpus, k refers to the minimal number of words appearing before the subject, l to the number of words between the subject and verb (the context size), m to the minimal number of words after the verb, and a to the position of the attractor relative to the subject (see Figure 5.1 for an example). I use an asterisk to indicate that no restrictions are placed on one of the above-mentioned variables; e.g. $A*$ indicates that there may or may not be an attractor. Finally, I denote data sets with sentences that have no attractor with $A-$.

5.3 Predicting number from activations

Gulordava et al. (2018) reported an accuracy of 81.1 for the sentences in their

[3] `https://github.com/TalLinzen/rnn_agreement`

original corpus and an accuracy of 74.1 for their *nonce* corpus. As a saliency check, we replicate their results using their data and their code. I report the results of this replication in Table 5.1.

While overall we obtain similar trends, the accuracy scores of our replication are slightly lower than those reported by Gulordava et al. (2018). As the results obtained with our own implementation exactly match those obtained with the script shared by Gulordava et al. (2018), I currently have no explanation for the discrepancy between these scores. I do, however, consider the differences small enough to proceed with the real purpose of this study: using diagnostic classification to understand how the model under scrutiny represents number information.

	Gulordava et al.	**Our Accuracy**
Original	81.0	78.0
Nonce	74.1	70.7

Table 5.1: Accuracy on both the original and nonce (English) test set Gulordava et al. (2018). Accuracies represent the percentages of sentences for which the correct verb form is assigned a higher probability under the LM than the incorrect form.

To remind the reader: the key idea of diagnostic classification – introduced earlier in this thesis – is to test whether a model's intermediate representations contain information about a particular phenomenon – such as subject-verb agreement – by training another model to recognise the information relevant to the phenomenon in the internal activations of the model. More precisely, given a data set of intermediate LSTM representations and a set of labels that describe the hypothesis to be tested, a meta-model (the diagnostic classifier) can be trained to predict the correct label from the representations. If the model succeeds in this task (i.e. if it achieves a performance significantly above chance on test data), this indicates that the LSTM is in fact computing or keeping track of the hypothesised information. In the first and simplest diagnostic classifier experiment, we train a classifier to predict the number of syntactic subject (and thus of the main verb that agrees with it) of the sentence from the hidden activations of the model.

5.3.1 Diagnostic classifier training

We create a training set containing 1000 sentences that all have 5 words between subject and verb (i.e. the context size is 5), have at least one word before the subject and after the verb, and for which no attractor based constraints are placed on the training set (*WD-K1-L5-M1-A∗*). We run the pre-trained LM on this corpus, and for both layers we extract activation data for both the hidden layer activations $\mathbf{h_t}$, the memory cell activations $\mathbf{c_t}$ and the forget, input and output

gate activations $\mathbf{f_t}$, $\mathbf{i_t}$ and $\mathbf{o_t}$, respectively. For example, for a single sentence of length n we obtain $5 \times 2 \times n$ activation vectors, because the model has 2 layers, there are n time steps, and there are 5 types of activations at each time step t: $\mathbf{h_t}, \mathbf{c_t}, \mathbf{f_t}, \mathbf{i_t}, \mathbf{o_t}$. We then label all activations with the number of the main verb of the sentence from which it was generated (either 'singular' or 'plural') and train a separate diagnostic classifier for each of the 10 components of the LSTM.

5.3.2 Results

We test the trained diagnostic classifiers on two test sets, that differ with respect to whether the model generated a correct or an incorrect prediction for that sentence (i.e. a sentence s is in the 'correct' set iff the model assigns higher probability to the correct form of the sentence than to the incorrect form). We label these test sets the *correct* and *wrong* set, based on these predictions. Otherwise, the sentences in the two sets have similar features, containing both sentences from *WD-K1-L5-M1-A3*. While we strive to generate the 'wrong' and 'correct' test sets with 100 sentences each, this is not always possible due to data sparsity. However, we ensure that both test sets have approximately the same size and do contain at least 50 sentences.

	$\mathbf{h_t}$	$\mathbf{c_t}$	$\mathbf{f_t}$	$\mathbf{i_t}$	$\mathbf{o_t}$
Layer 0	**0.74** / 0.57	**0.76** / 0.58	**0.69** / 0.55	**0.68** / 0.56	**0.69** / 0.56
Layer 1	**0.90** / 0.62	**0.91** / 0.65	**0.86** / 0.61	**0.86** / 0.60	**0.87** / 0.60

Table 5.2: Mean diagnostic classifier accuracies (*correct* test set/*wrong* test set) across time steps, averaged over data sets drawn from different context sizes and attractor positions (with $K=0$, $M=0$, $5 \leq L \leq 7$ and with a variable number of attractors at different positions).

In Table 5.2, I print the average diagnostic classifier accuracies. For both the *wrong* and the *correct* test sets, the accuracies are highest at the second layer across almost all LSTM components, suggesting that the last LSTM layer reaches the level of abstraction which can best capture long-distance dependencies. The highest accuracy is achieved by the hidden layer and memory cell of the network, although number can also be encoded with a relatively high accuracy from the gates of the network, which is somewhat surprising, given the function of the gates.

In Figure 5.2, I show the average diagnostic classifier accuracies at different time steps (for a set with a context size of 5 and a single attractor located three words after the subject). An interesting first observation concerns the accuracy at time step 0. Already at this time step, which represents activations from *before* the model has seen the subject, the accuracy of the model is above chance level. I did not further investigate this issue but hypothesise that it originates from the

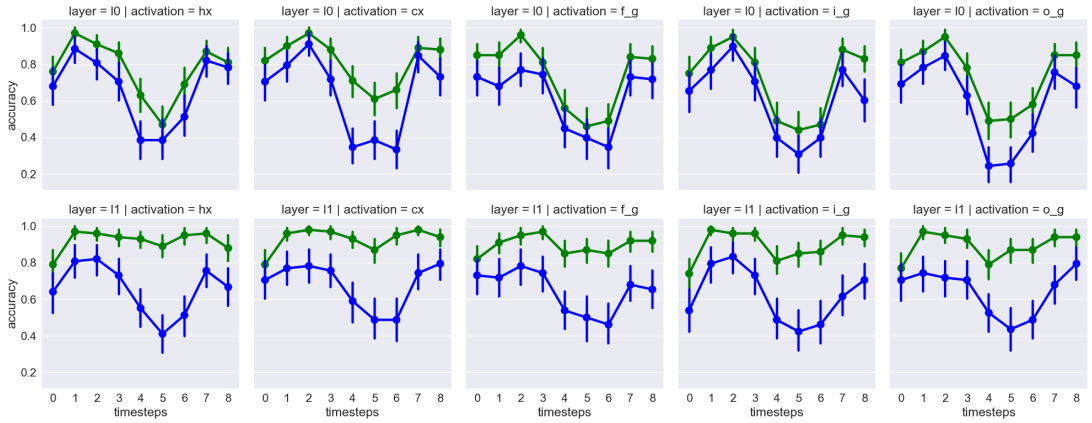


Figure 5.2: Accuracies over time (on WD-K1-L5-M1-A3) of 10 diagnostic classifiers trained and tested on data from different components of the LSTM. The subject is at time step 1, attractor at time step 4 and the verb at time step 7. Green lines represent sentences for which the LSTM predicts the correct verb, blue lines sentences for which the LSTM assigns a higher probability to the incongruent counterpart.

fact that even words that occur before the subject can give information about the number of the subject. For instance, the determiner *a* should be followed by a singular subject.

Unsurprisingly, the diagnostic classifiers obtain their best accuracy scores at (or just after) the subject and verb time step. This pattern is consistent across context sizes, attractor positions, and number of words before the subject and after the verb, and regardless of whether the LSTM prediction was correct or incorrect. This result confirms that the model indeed learns to recognise the number information of subject heads and present tense verbs.

The figure furthermore shows that performance differs between layers and between components. The diagnostic classifier performance of the second layer components, moreover, critically differs for 'correct' and 'wrong' sentences. While for 'correct' instances, the accuracy in all second layer components increases again after dropping after the subject time step, 'wrong' sentences do not show such a correction.

Aside from showing differences among layers, our results also show differences across components. For example, classifiers that make predictions based on $\mathbf{c_t}$ and $\mathbf{h_t}$ activations of 'correct' sentences are the most stable in terms of accuracy, in particular at the second layer. Although all LSTM components outperform the random baseline of 50%, these results imply that the cell state and the hidden activation are the LSTM components that are most specialised at processing number information. We further test this claim in Section 5.4.

Another cause of differences across diagnostic classification error rates is the presence of agreement attractors. Accuracies for the test sets with an attractor

are overall lower than those obtained on sentences without an attractor. While the error rate rises in Figure 5.2 and diverges between 'correct' and 'wrong' at the position of the attractor, the same does not happen for sentences without attractors (not plotted).

5.4 Representations across time steps

The results so-far show us that number information is most easily retrieved from the internal states of the language model when the noun or verb have just been presented, but not very well from the internal states at intermediate time steps. In this section, I focus on these changing representations.

In the previous experiment, we trained diagnostic classifiers on activation data for all words in the sentence. In contrast, we now train *separate* diagnostic classifiers for each time step: each DC_t is trained with activation data at time step t only. We *test* each DC_t on data from all other time steps as well. With a total of T time steps, this gives us $T \times T$ diagnostic classifier accuracies that together constitute a *Temporal Generalization Matrix* (King and Dehaene, 2014; Fyshe et al., 2016). Effectively, we are forcing each diagnostic classifier to specialise on time step-specific representations of subject-verb agreement information. If this information is represented similarly across time steps, a classifier trained at the subject time step should also have a high accuracy when applied to the activations corresponding with the time step in which the attractor occurs. If, on the other hand, information is dynamically encoded, no such generality of classifiers is to be expected.

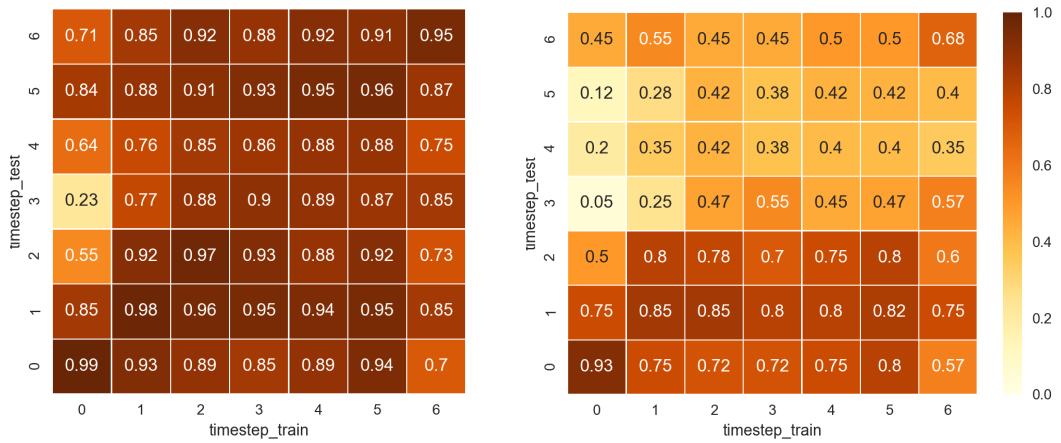
5.4.1 Diagnostic classifier training

To test the development of the encoding over time, we create a corpus with sentences that are identical with respect to the position of the subject, attractor and main verb. We train on sentences with five intervening words between the subject, containing one attractor three time steps after the subject, and a variable number of words before the subject and after the verb (*WD-K*-L5-M*-A3*). After computing the activations for all sentences, we crop off the words before the subject and after the verb and split the remaining activations according to time step. We collect the activations corresponding to all six time steps from subject to verb, in 6 different bins. For each bin, we train a separate diagnostic classifier.

5.4.2 Results

For testing we create again a *correct* and a *wrong* test set, drawing both sets from *WD-K*-L5-M*-A3*. Following the same procedure as for the training data, we split both test sets up into six time steps. In the remainder of this section,

position 0 thus always refers to the position of the subject, while the attractor and main verb of the sentence occur at time step 3 and 6, respectively.



(a) *Correct* test set (b) *Wrong* test set

Figure 5.3: The temporal generalisation matrices for diagnostic classifiers trained on memory cell activation at different timesteps, for correct (left) and wrong (right) sentences. Timestep 0 corresponds to the subject of the sentence, the attractor and main verb of the sentence occur at timesteps 3 and 6, respectively. The corpus used for testing is *WD-K*-L5-M*-A3*.

In Figure 5.3, I plot the Temporal Generalization Matrix for the second layer memory cell ($\mathbf{c}_t^1$) activation data, containing the accuracies of T DC's evaluated on T time step data sets each. The left figure shows results for sentences from the *correct* test set, the right figure for sentences from the *wrong*.

A first observation is that accuracies on the diagonals – which correspond to classifiers that were trained and tested on the same time step – are typically high for sentences that are processed correctly while being lower for incorrectly processed sentences. Interestingly, this difference already emerges at the first two time steps, where no attractor has yet appeared – suggesting that an important part of the problem with misclassified sentences is the encoding of the relevant information already when the subject occurs.

Comparing the plots for correctly and incorrectly processed sentences, we notice that the attractor (time step 3) has a very large effect on the accuracies for incorrectly classified sentences. For those sentences, the language model's internal states contain no information anymore after the attractor is processed: time steps 4 and 5 receive below chance accuracies, whereas for correctly processed sentences the attractor prompts only a slight dip in accuracy.

Focusing on the correctly processed sentences, an interesting observation that can be made is the discrepancy between column 0 and 6 (the columns corresponding to the subject and verb of a sentence) and the rest of the columns. While the first

and last column generalise poorly to different time steps, the classifiers trained and tested on time steps 1-5 show a different pattern: despite potential effects from the attractor at time step 3, the accuracy scores do not change substantially across time steps. This implies that the LSTM represents subject-verb agreement information in at least two different ways: a short-term 'surface' level at and around the subject time step, and a longer-term 'deep' level for successive sequence processing. This deep level information seems to be represented most generically at time step 4, the classifier for which has the highest average accuracy across time steps.

In the next section, I delve deeper into the representations at this time step and investigate which components of the LSTM are most crucial in representing this information.

5.5 Representations across components

In this section, I briefly investigate the stability of information across components of the LSTM. Rather than comparing diagnostic classifiers that are trained on different *time steps*, I now compare diagnostic classifiers that are trained on different *components*. I focus on time step 4 which, following the previous experiments, optimally represents 'deep' information about subject-verb agreement. For our experiments, we use the same training set as for the previous experiment, with sentences with a context size of 5 and a single attractor located three words after the subject (*WD-K*-L5-M*-A3*).

Figure 5.4 shows the 'spatial generalization matrix', with diagnostic classifiers trained at time step 4 with data from each components separately. The horizontal axis represents the components the diagnostic classifier is trained on, the vertical axis the component used for testing. On the diagonal, we thus see the accuracies of classifiers trained on the same components as they are tested on. The results on these diagonal are consistent with the results shown in Figure 5.3. On the diagonal of the right plot, we see that for sentences for which the model made an incorrect prediction, there is no component in either layer that consistently represents the grammatical number of the subject. For correct sentences (shown in the left plot), the diagonal shows that the second layer components represent the grammatical subject well, while the first layer components have a much lower accuracy. The highest accuracies are obtained by the second layer hidden and memory cell activations (with accuracies of 0.92 and 0.90, respectively). The plot furthermore shows that the representations in these two components are similar: a classifier trained on the memory cell activations at time step 4 obtains an accuracy of 0.85 when tested on the hidden activations; a classifier trained on the hidden activations an accuracy of 0.89 when trained on the memory cell activations.

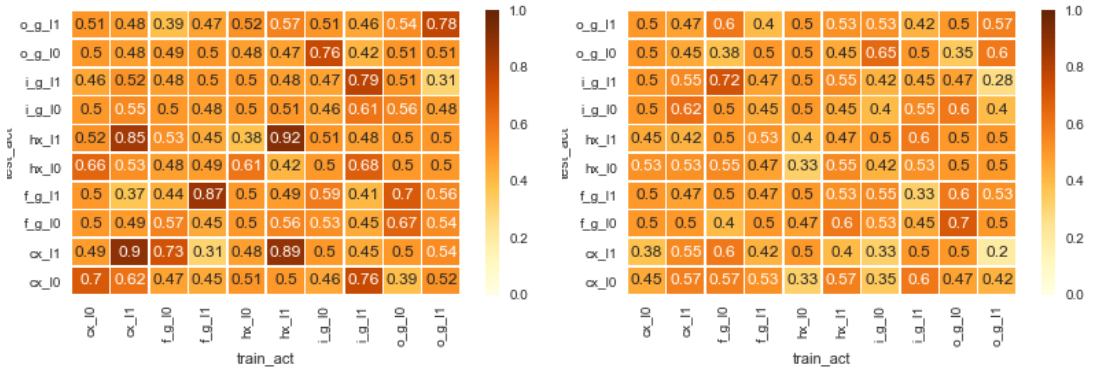


Figure 5.4: The spatial generalization matrices at time step 4. Shown are accuracies of diagnostic classifiers trained on activation data of each component separately (horizontal), and tested on each component separately (vertical). 'Correct' sentences for which the model made the right prediction are shown on the left, 'wrong' sentences on the right.

5.6 Interventions

In the experiments presented above, we used diagnostic classifiers to investigate the way the LSTM performs the verb number prediction task. A concern with diagnostic classifier experiments is the reliability of the information that it picks up. The fact that a high diagnostic classifier accuracy cannot directly be taken as proof that the inferred feature is in fact represented and used by the model, because it is possible that the information was computed post hoc by the diagnostic classifier and was not actually used by the model itself.

In a recent paper, Hewitt and Liang (2019) propose to use *control tasks* that require a diagnostic classifier to predict information of which the modeller knows it is not represented by the model (for instance because it is random). The accuracy of the diagnostic classifiers on such a control task provides a baseline for how difficult it is to extract specific types of labels from activations post hoc and can thus be used to detect potential false positives in diagnostic classifier experiments. If the control diagnostic classifier has a high accuracy, this indicates that the diagnostic classifier experiment itself is unreliable, as the information may have been inferred by the diagnostic classifier rather than being represented by the model.

However, also using control tasks should be considered a post hoc method. Also in this case, a high accuracy on a control task does not irrefutably demonstrate that the diagnostic classifier inferred information that was not used by the model. It merely demonstrates that the diagnostic classifier *could have* inferred the information if it was not there. In this section, I describe a different approach to detect the reliability of a diagnostic classifier, which uses diagnostic classifiers to *intervene* in the model's behaviour. If we can purposely change the behaviour of

the model by using the diagnostic classifier to change specific information, this is a clear demonstration that the information learned by the classifier is causally involved in the behaviour of the model.[4]

5.6.1 Intervention procedure

We use the same data for our intervention experiment as we used for the experiments described in the previous section: a corpus of sentences with the subject at time step 0, one attractor 3 time steps after (at time step 3) and the main verb at time step 6 (*WD-K0-L5-M0-A3*). We train four diagnostic classifiers to predict the number of the sentence from the hidden layer activations and memory cell activations for both layers, respectively.

We then use the trained diagnostic classifiers to actively influence the course of processing by the LSTM. We start processing sentences from the Gulordava et al. (2018) corpus, but after processing the subject of a sentence – the point where we discovered information is stored in a corrupted way for *wrong* sentences – we halt the LSTM's processing, extract the hidden activation and the activation of the memory cell, and apply the trained diagnostic classifier to predict whether the main verb in the sentence is singular or plural. We then slightly adapt the activations of the hidden layer and memory cell activations of the model based on the error that is defined by the difference between the predicted label and the correct label for this particular sentence. We compute the gradients of this error with respect to the activations of the network, and we modify the activations using the delta-rule (we empirically decided on $\eta = 0.5$). In other words, we change the activations such that the prediction of the diagnostic classifier is slightly closer to the gold label. After adapting the activations, we continue to process the rest of the sentence given the adapted activations.

5.6.2 New diagnostic classifier accuracies

In Figure 5.5, I plot the accuracies of diagnostic classifiers trained on different components of the LSTM when we apply them on activations resulting from sentences processed with the above-described intervention. Trivially, the intervention increases the accuracy of the diagnostic classifiers for the hidden activation and memory cell of the network at time step 1, the subject time step. More interestingly, this effect persists while the processing of the sentence proceeds – in some cases it grows even stronger – and thus in fact *changes* how the LSTM processes the sentence. This effect is not only visible in the components on which the intervention is done, but also displays in the gate-values, that are not directly

[4] Arguably, in our case also the difference between diagnostic classifier accuracy for *correct* and *wrong* sentences suggests that the diagnostic classifiers in fact inferred information also used by the model.

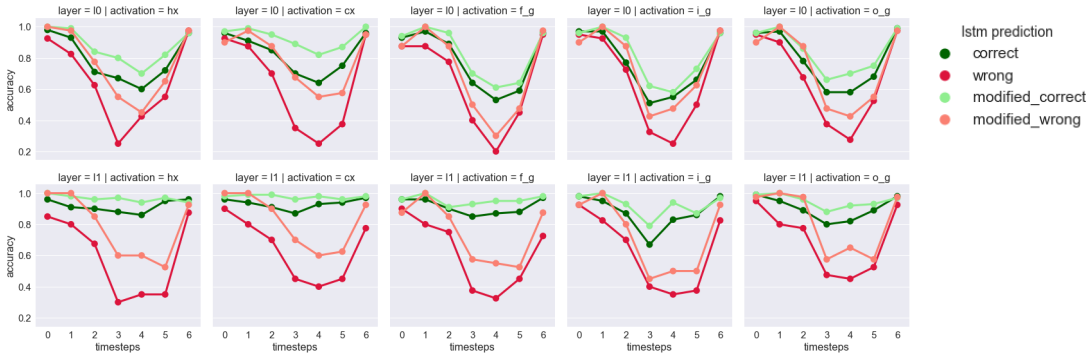


Figure 5.5: Mean accuracies for each component of the LSTM after an intervention of $\mathbf{c_t}$ and $\mathbf{h_t}$ at the subject timestep 0. An attractor and the agreeing verb occur at timestep 3 and 6, respectively.

updated but only changed indirectly through the interventions in the memory cell and hidden activations.

	without intervention	**with intervention**
Original	78.0	85.4
Nonce	70.7	75.6

Table 5.3: Accuracy of the LSTM on the Gulordava et al. (2018) agreement test, with and without an intervention at the subject time step.

5.6.3 NA-task accuracies

To put our interventions to the test, we now assess the predictions made by the LSTM as a consequence of the interventions. First, we confirm that the intervention does not cause strong anomalies in the LSTM, by comparing the perplexity of a small corpus of sentences processed *with* interventions at the subject time step with sentences processed without any interventions. We do not find any strong differences, confirming that the intervention is minor with respect to the overall behaviour of the LSTM. Table 5.4 shows an example sentence from this corpus.

On the number agreement test described by Gulordava et al. (2018) and conducted earlier in this chapter, however, the intervention *does* have a strong

	An	official	estimate	issued	in	2003	suggests	suggest
Original		-11.05	-8.43	-8.47	-1.24	-3.95	-5.75	**-5.70**
Intervention		-11.05	-8.43	-8.47	-1.27	-3.97	**-5.69**	-6.44

Table 5.4: Example sentence as processed by the neural language model of Gulordava et al. (2018), without and with our intervention. Numbers indicate perplexities per word.

effect, as can be seen in Table 5.3: the accuracy of predicting the correct verb number increases from 78.1 to 85.4 and from 70.7 to 75.6 for original and nonce sentences, respectively.

These results provide evidence that diagnostic classifiers, at least in this case, are able to pick up features that are actually used by the LSTM, rather than relying on idiosyncrasies in the high dimensional spaces that happen to be correlated with the predicted labels. Furthermore, they illustrate how diagnostic classifiers can be used to actively change the course of processing in a recurrent neural network, and with this opens a path that moves from merely *analysing* to actively *influencing* black box neural models.

5.7 Conclusion

In this chapter, I focused on understanding how an LSTM language model processes subject-verb congruence, using the number agreement task first presented by Linzen et al. (2016) in the version made online by Gulordava et al. (2018). We first replicated their results and then trained *diagnostic classifiers* to discover where and how number information is encoded by the LSTM. I showed that number information is encoded *dynamically* over time, rather than remaining constant.

Using a cognitive neuroscience-inspired method, we then trained different diagnostic classifiers for different time steps, resulting in a *Temporal Generalisation Matrix*, which provides more information about changing representations over time. I found that while number information is stored in very different ways at the beginning and end of a sentence, in between a relatively stable 'deep' representation is maintained. Additionally, we find that for sentences in which the LSTM prefers an incongruent verb over congruent one, the information appears to be stored wrongly already at the beginning of the sentence, far before the verb or any attractor is to appear.

Combining this information, I invert the process of diagnostic classification, using the classifiers to *influence* rather than merely observe. To do so, we process sentences with the language model and, at the point where we found information to be often corrupted, we intervene by (slightly) changing the hidden activations of the network by backpropagating through a trained DC. After this intervention, we continue processing the sentence as normal. I show that this small intervention

has little effect on the overall course of the LSTM, but a very large effect on the verb prediction at the end: the percentage of sentences for which the model prefers the congruent over the incongruent verb rises from 78.1% to 85.4%.

With these results, I show not only that diagnostic classifiers also in this case offer a detailed understanding of where and when information is encoded in a neural model, but also that this information can be used post hoc to change the course of the processing of such a model. As such this experiment demonstrates that the information inferred by the diagnostic classifiers was indeed causally linked to the behaviour of the model.

Chapter 6

Neuron ablation

In the previous chapter, I described a diagnostic classification study in which we investigated a pre-trained language model's ability to capture long-distance subject-verb relationships. These studies uncovered several interesting aspects concerning when and in which components the model stores information required to processes such relationships, but they did not tell us *how* the model processes them. In this chapter, using the same pre-trained language model, I follow up on this question, focusing on the actual mechanisms used by the model.[1] I describe a controlled data set and several ablation studies that we used to pinpoint important units or clusters of units in the model, aided by further diagnostic classifier experiments.[2]

Chapter outline Following the structure of the previous chapters, I first describe the data we used for the study described in this chapter (Section 6.1).

[1] The main results presented in this chapter were confirmed also for different runs and hyper-parameters. I will not further discuss those results.

[2] The work described in this chapter is part of a collaborative project with Facebook AI Research and Neurospin in Paris, to which several researchers contributed. I was one of these researchers, but to avoid giving the impression that the ideas expressed should all be attributed solely to me, I decided to report about this work primarily in the third person. Part of this project was published at NAACL 2019:

> Yair Lakretz, German Kruszewski, Theo Desbordes, Dieuwke Hupkes, Stanislas Dehaene, and Marco Baroni. The emergence of number and syntax units in LSTM language models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019

It was presented there by Yair Lakretz. I took part in the weekly meetings in which we set the course for this paper, conducted several experiments and contributed to the write-up; Many of the analyses presented in this paper were done by Yair Lakretz. In this chapter, I report some results presented in this paper and a few additional experiments. The figures and plots in this chapter are (re)generated by me, sometimes I repeated them for different data sets than we used for the published article. The text in this chapter is written by me.

I then report the results of the model when tested on this data (Section 6.2). In Section 6.3 to 6.5, I discuss the main results of the ablation and diagnostic classification studies. I conclude in Section 6.6.

6.1 Data

Like the previous study I described, also the study described in this chapter makes use of a number of different data sets that are specifically designed to test particular properties of the model.

6.1.1 Linzen data set

One of the data sets used is the Linzen et al. (2016) data set, in the version that was made available by Gulordava et al. (2018). This data set contains naturalistic, corpus-derived number agreement problems, extracted from a large corpus with Wikipedia data.

6.1.2 Synthetic data sets

In addition to this, Lakretz et al. (2019) use a series of synthetic data sets that contain number agreement problems involving different syntactic constructions. The sentences in these data sets differ with respect to the linear distance between subject and verb, the syntactic distance between subject and verb and the type of sentential material that is separating them. In particular, Lakretz et al. construct seven different syntactic templates:

Simple	The `N V` ...	The **boy** *greets* ...
Adv	The `N adv V` ...	The **boy** probably *greets* ...
2Adv	The `N adv1 adv2 V`...	The **boy** most probably *greets*...
CoAdv	The `N adv` and `adv V`	The **boy** openly and nicely *greets* ...
NamePP	The `N prep name V` ...	The **boy** near Pat *greets* ...
NounPP	The `N prep` the `N V` ...	The **boy** near the car *greets* ...
NounPPAdv	The `N prep` the `N adv V` ...	The **boy** near the car kindly *greets*...

Conditions For all of these templates, Lakretz et al. generate 600 sentences by randomly sampling the object/subject nouns, verbs, adverbs, prepositions, proper nouns and position nouns from a pool of options (I report the word lists for every word type in Table 6.1). They ensured that no semantic anomalies arose as a consequence of the random sampling. E.g. sentences like "The athlete under the dog nicely confuses" are never sampled.

For each sentence, the data set contains several both a plural and a singular version (the examples given above are all *singular* examples). For the sentences that contain intervening nouns (the *NounPP* and *NounPPAdv* templates), also

Subject/object nouns	athlete, aunt, boy, carpenter, doctor, farmer, father, friend, girl, guy, kid, lawyer, man, mother, poet, singer, teacher, uncle, victim, woman
Proper nouns	Barbara, Bill, Bob, Jim, John, Linda, Mary, Mike, Pat, Sue
Position nouns	bike, car, cat, chair, desk, dog, table, tree, truck, window
Verbs	admire, approve, avoid, confuse, criticise, discourage, encourage, engage, greet, inspire, know, observe remember, stimulate, understand
Adverbs	carefully, certainly, definitely, deliberately, gently, knowingly, openly, overtly, probably
Adverbs 2	indeed, most, now, quite
Prepositions	above, behind, besides, near, under

Table 6.1: Word lists that we sampled sentences from. Final sampled sentences can be found at online.

the number of the intervening nouns is systematically varied, resulting in two *congruent* and two *incongruent* versions of the sentence. For instance, for the nounPP template, the previously mentioned sentence would be a *congruent* **SS** sentence. All conditions for this sentence are:

SS	*congruent*	The **boy** near the *car greets* ...
PP	*congruent*	The **boys** near the *cars greet* ...
SP	*incongruent*	The **boy** near the *cars greets* ...
PS	*incongruent*	The **boys** near the *car greet* ...

Data sets To generate the data sets for all templates, Lakretz et al. (2019) sample 600 sentences for every condition. For templates that have only one common noun (Simple, Adv, 2Adv, CoAdv and NamePP), there are only two conditions (S and P), those data sets thus have 1200 samples. For templates that contain two nouns (NounPP, NounPPAdv), there are four conditions, of which two congruent (SS and SP) and two incongruent (SP and PS). Those data sets thus consist of 2400 sentences. The exact sampled sentences that we used for our experiments can be found online.[3]

6.2 Task performance

The carefully crafted synthetic stimuli allow to test the model's behaviour on different syntactic templates that pose different challenges to keeping track of subject-verb relationships. In Table 6.2, I report the model's accuracy on all

[3] `https://github.com/FAIRNS/Number_and_syntax_units_in_LSTM_LMs`

different conditions, as well as the model's accuracy on the previously described Linzen data set.[4] The conditions in which the subject and verb are *singular* are depicted in green.

NA task	**Condition**	
Simple	S	100
Simple	P	100
Adv	S	100
Adv	P	99.6
2Adv	S	99.9
2Adv	P	99.3
CoAdv	S	98.7
CoAdv	P	99.3
namePP	SS	99.3
namePP	PS	68.9
nounPP	SS	99.2
nounPP	PP	99.0
nounPP	SP	87.2
nounPP	PS	92.0
nounPPAdv	SS	99.5
nounPPAdv	PP	99.8
nounPPAdv	SP	91.2
nounPPAdv	PS	99.2
Linzen data		93.9

Table 6.2: Model results on all different NA tasks. The first column indicates the syntactic construction, the second column the condition. Singular conditions are coloured green.

Almost all accuracies of the model are high, a promising result for the main purpose of this study, which is to discover how subject-verb relations are handled by the model. The results also confirm that some conditions are easier than others. In particular, the *simple* condition, in which the subject and verb are directly adjacent, is easier than the condition in which the subject and verbs are separated by one or more adverbs. The conditions in which the intervening material contains names or nouns appear the most difficult. Of those conditions, as expected, the incongruent conditions (SP/PS) are more difficult than the congruent conditions (SS/PP).

[4]The setup is identical to the original setup of Linzen et al. (2016) and the setup also used in the previous chapter: a sentence is considered *correct* if the model assigns a higher probability to the correct verb form than to the incorrect one, and incorrect otherwise. Accuracy expresses the ratio between sentences for which the model's prediction was correct and the total number of sentences.

Intriguingly, singular conditions seem more difficult than plural conditions when the subject and verb are further apart: the accuracy on the SP conditions is lower than the accuracy on the PS conditions. A possible explanation for this asymmetry could be that the plural verb form in English is more frequent than the singular one, as it is identical to also other verb forms like the infinitive and first and second person singular verbs. Later, in Chapter 7, we will see more evidence that the model treats singular and plural verbs in a different way.

6.3 Long-distance number units

Lakretz et al. (2019) start by investigating the impact of single units on the model's performance. They do so by considering the effect of lesioning or ablating a unit on the overall model performance on the NA tasks. If the ablation of a particular unit has a strong impact on this performance, this implies that the encoding of the relevant information for this task is encoded in a *local* fashion. If no particular unit causes a drop in performance, subject-verb relations must instead be represented in a more distributed way.

6.3.1 Ablation experiment

The model's units are ablated by setting their activation to zero. As there are 1300 distinct units in the model, this results in 1300 different ablation experiments. I report the results of these experiments in Table 6.3.

Somewhat surprisingly, the ablation studies point to two units that have a strong effect on the model performance: unit 776 and 998. Ablating these units, both units in the second layer of the model, reduces the model's performance by more than 10% in various conditions. There are two remarkable observations to be made about the impact of ablating unit 776 and 998.

First of all, both neurons seem to play an important role in processing long-distance subject-verb relations, but they are less important for short distance relationships. They cause a substantial reduction in performance in the more difficult incongruent conditions, while the model performance on shorter distance conditions (Simple, Adv, 2Adv) remained intact. Secondly, the effect of the two units is conditioned on the grammatical number of the subject. Ablating unit 998 causes a performance reduction only when the subject is singular, whereas 776 impacts plural conditions. Following Lakretz et al. (2019), I will in what follows refer to these two units as the plural and singular units, respectively, or *long-distance units*, collectively.

Interestingly, the effect of the plural unit persists over more conditions than the singular unit. Whereas the plural unit has an impact in congruent ánd incongruent conditions, the singular unit only impacts the *incongruent* cases (the CoAdv forms a surprising exception). Furthermore, ablating the plural unit has a substantial

NA task	Condition	Full Model	Ablated	
			776	**988**
Simple	S	100	-	-
Adv	S	100	-	-
2Adv	S	99.9	-	-
CoAdv	S	98.7	-	82
namePP	SS	99.3	-	-
nounPP	SS	99.2	-	-
nounPP	SP	87.2	-	54.2
nounPPAdv	SS	99.5	-	-
nounPPAdv	SP	91.2	-	54.0
Simple	P	100	-	-
Adv	P	99.6	-	-
2Adv	P	99.3	-	-
CoAdv	P	99.3	79.2	-
namePP	PS	68.9	39.9	-
nounPP	PS	92.0	48.0	-
nounPP	PP	99.0	78.3	-
nounPPAdv	PS	99.2	63.7	-
nounPPAdv	PP	99.8	-	-
Linzen		75.3	93.9	-

Table 6.3: Results of the Lakretz et al. (2019) ablation study. The first two columns indicate the NA task and condition, the second column the full model accuracy and the last two columns the accuracy when unit 776 and 988, respectively, are ablated. Dashes indicate performance reductions less than 10%.

impact on the model's accuracy on the more diverse Linzen stimuli, but ablating the singular unit does not. In the next chapter of this dissertation, we will see that this asymmetry may be caused by a *default* reasoning effect in the model.

Overall, these results suggest a highly local encoding scheme of grammatical number when processing long-distance dependencies, whereas shorter distance dependencies and syntactic structure are likely to be represented in a more distributed fashion.

6.3.2 Singular and plural unit dynamics

Lakretz et al. (2019) plot the cell behaviour for the long-distance units for the nounPP task, the simplest NA-task with a long-distance dependency and an intervening noun. I regenerated this plot (shown in Figure 6.1), following the colour scheme I used before: green for singular, black for plural subjects.

Recap of LSTM dynamics

Both units show an exemplary picture for keeping track of number throughout the long-distance dependency. How they do so, is best understood considering the LSTM memory cell update and output rules:

$$\tilde{c}_t = \tanh(Wx_t + Vh_{t-1} + b) \tag{6.1}$$

$$c_t = i_t \circ \widetilde{c}_t + f_t \circ c_{t-1} \tag{6.2}$$

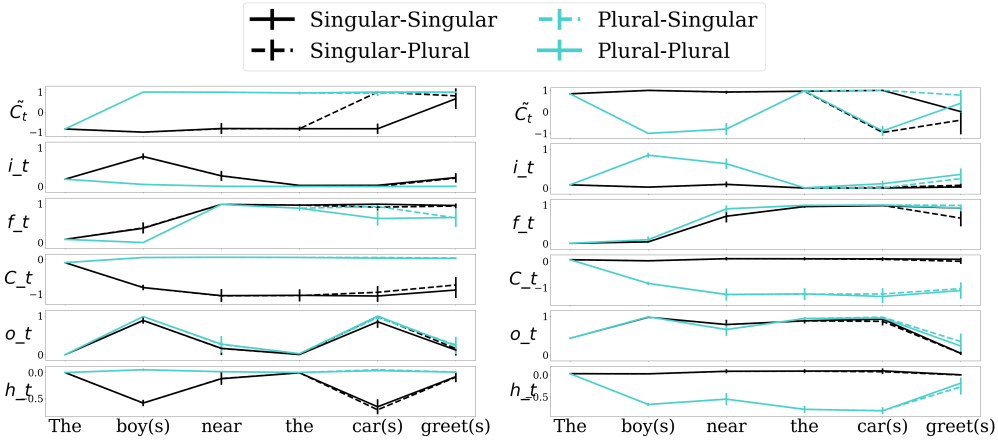
$$h_t = o_t \circ \tanh(c_t), \tag{6.3}$$

As explained in Chapter 2 of this dissertation, the information flow throughout an LSTM model is modulated by three *gates*, whose values are computed by the model itself. At every time step, an LSTM model computes a *candidate memory cell activation* $\widetilde{c}$, which is conditioned on the previous hidden state and the current input (Equation 6.1). The next memory cell value is a weighted sum of this candidate activation and the previous memory cell activation, where the *input* gate control how much of the candidate activation contributes, and the forget gate controls how much the previous memory cell activation contributes[5] (Equation 6.2). The third gate, which is called the *output* gate, determines how much of the memory cell passes on to the hidden state of the model, which is directly connected with the output layer.

Ideal solution

Considering the described LSTM dynamics, an individual cell could represent number through a long-distance relationship as follows:

[5]Somewhat confusingly, when the forget gate takes its maximal value of 1, this means that nothing is forgotten.



(a) Singular unit 988 (b) Plural unit 776

Figure 6.1: The average activation values of different components of both long-distance units throughout the processing of NounPP sentences.

1. First, it encodes the number of the subject in the candidate memory cell activation $\tilde{c}$ at the subject time step;
2. This information then needs to be transported to the memory cell c_t of the network, which it does by:
   a. Opening the input gate ($i_t = 1$), which lets this information pass from $\tilde{c}$ through to the memory cell c_t of the unit.
   b. Closing the forget gate ($f_t = 0$), which clears the previous content of the memory cell.
3. In between the subject and the verb, the model closes off the cell to any input information ($i_t = 0$), and it only retains the information that was stored at the subject time step by completely opening the forget gate ($f_t = 1$);
4. One step before the verb arrives, the output gate of the cell should be opened to let the number information flow to the hidden state of the cell, where it can directly impact the model's prediction.

Behaviour of the long-distance units

As can be seen in Figure 6.1a and 6.1b, both the singular and plural unit almost perfectly follow the described solution:

1. In the top row of Figure 6.1, we see that the cell activations $\tilde{c}_t$ of both unit 988 and unit 766 respond when the subject comes in. This response is different for singular and plural subjects, at that time step, $\tilde{c}_t$ thus encodes whether the just observed subject was singular (black lines) or plural (blue lines).

2. In the second and third row, we see how the units use their gates to copy this just stored information to the memory cell c_t:
   a. In row two, we see that the input gate i_t of the singular unit 988 opens when the subject is singular, thus letting this information pass through to the memory cell c_t. The input gate of the plural unit 776 behaves in opposite way: it opens for plural subjects but stays closed for singular subjects.
   b. In row three, we see that the forget gate of both units is closed, thus clearing the cell from previous information ($f_t = 0$).
3. The number of the subject is then encoded in the cell activation c_t of the units. In subsequent steps, the forget gate f_t opens and stays open while the input gate i_t closes and stays closed, which retains the number information in the memory cell c_t (This can be seen in column four). An interesting deviation from the ideal solution occurs at the time after the subject, where the input gate activation drops but is not exactly zero. Lakretz et al. (2019) propose the speculative explanation that this may be useful to the processing of compound nouns, the number of which is dependent on the second noun.
4. The output gate, as predicted, takes the maximum value right before the verb needs to be predicted (fifth row). By doing so, it lets out the number information about the subject, which can be used by the model for the prediction of the verb in the next time step.

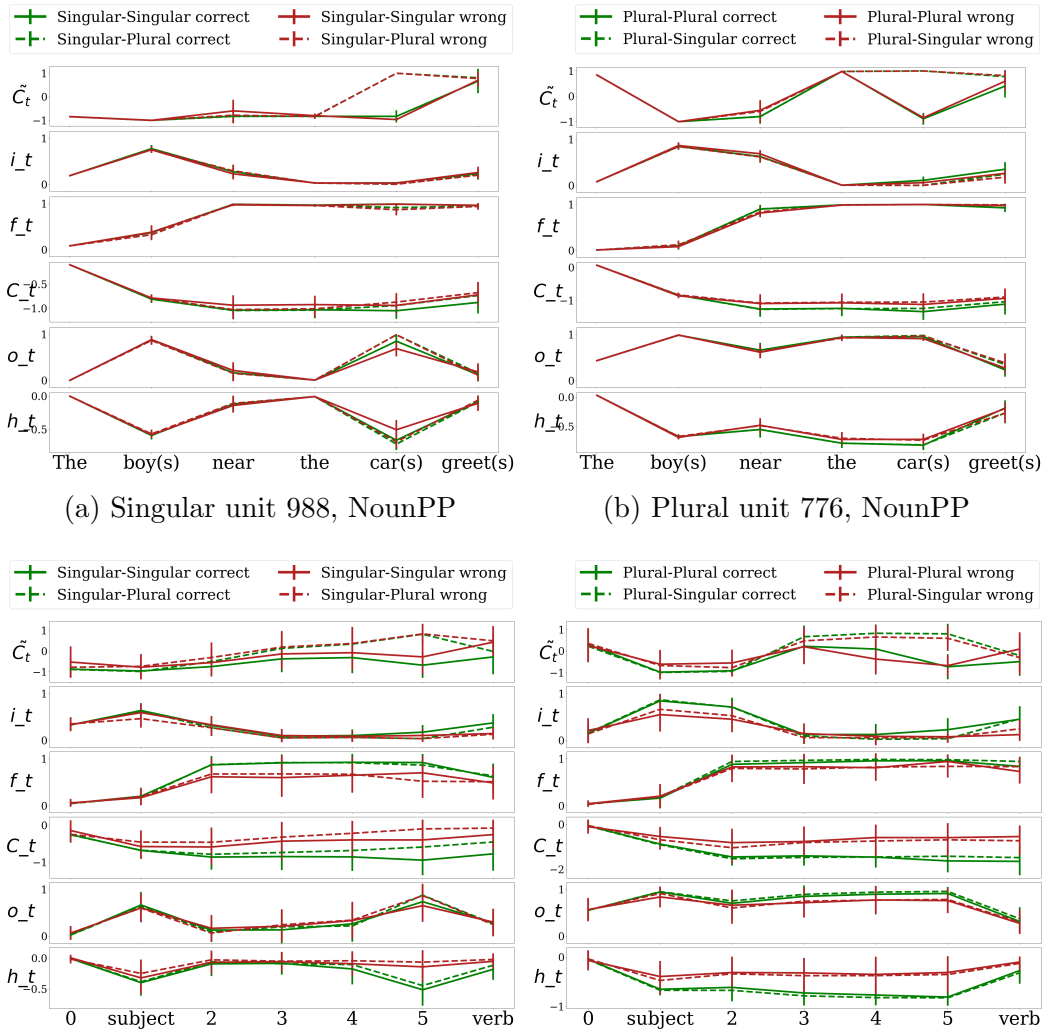
6.3.3 Correctly vs incorrectly processed sentences

Given the importance of the long-distance units for keeping track of number information throughout long-distance dependencies, one might expect a difference in the unit's behaviour depending on whether the model's prediction for the NA-task was correct or wrong. To test this hypothesis, I regenerated the plots we presented in the paper, but I split the data set based on whether the model's preferred the correct verb form at the end of the sentence or not (akin to the *correct* and *wrong* data sets of the previous section.

NounPP

In Figure 6.2a and 6.2b, I show the behaviour of the long-distance units on NounPP sentences correctly and incorrectly processed by the model. Somewhat surprisingly, the behaviour of the units is virtually identical across these two groups and seemingly unrelated to whether the model made a correct or incorrect prediction afterwards. The LR units thus reliably encode the correct grammatical number, but this does not always result in a correct prediction.

In the previous section, we already saw that ablating the long-distance units does not impact the model's accuracy for short-distance condition. There must



(a) Singular unit 988, NounPP (b) Plural unit 776, NounPP

(c) Singular unit 988, WD-K1-L5-M0-A* (d) Plural unit 776, WD-K1-L5-M0-A*

Figure 6.2: The average activation values of different components of the plural and singular unit throughout processing different data sets, split based on whether the model prediction was correct or wrong. The top two plots (a) and (b), show the units' behaviour on the *NounPP* data set, the lower plots (c) and (d) instead consider the corpus with the less controlled Wikipedia stimuli used in the previous chapter.

thus be a second, more distributed mechanism responsible for storing number information in those cases. Furthermore, while ablating the long-distance units reduces the model's performance on the NA-tasks, the performance does not drop to chance level in all cases. This suggests that long-distance number information may also be represented elsewhere in the model, in a more distributed way. A potential hypothesis for incorrect predictions is that conflicting information stored

by either one of these representations is competing with the information stored in the long-distance units.

Wikipedia data

For the Wikipidea-based data set that was used in the previous chapter, which contains more mixed stimuli, there is a slightly larger difference between *correct* and *wrong* sentences (see Figure 6.2c and 6.2d). For both the singular and plural unit, the hidden layer activations at the time step before the verb occurs deviate quite substantially, the main source of which seem to be the forget and output gate. This finding is consistent with the previous chapter, in which we also found a strong difference between correctly and incorrectly processed sentences.

It seems, thus, that there are at least two potential causes for mistakes on long-distance subject-verb relationships. In some cases, such as while processing the mixed Wikipedia stimuli, the long-distance units may not store and release the subject information entirely correctly, potentially resulting in a wrong prediction. In other cases, such as or the controlled NounPP stimuli, the long-distance units do correctly process number information. In such cases, mistakes thus stem from the model's inability to correctly propagate this information to its output layer, likely because it incorrectly deals with conflicting information stored elsewhere. In the next section, I consider again several diagnostic classification experiments to locate more distributed representations of number information that could not be found with the previously described ablation studies.

6.4 Short-distance number units

Ablation studies are an effective tool to find units that on their own have a strong impact on the behaviour of the network. They are less suitable to find units that are part of a cluster that represents information in a more distributed fashion.[6] To locate such units, Lakretz et al. (2019) resort to a diagnostic classification experiment similar to the generalisation through time experiment described in the previous chapter: they train a linear model to predict the number of the subject from the memory cell activations at the *subject* time step and then test the resulting model on all time steps of the incongruent conditions of the NounPP task. Rather than using model accuracy, Lakretz et al. (2019) evaluate the model using Area Under Curve (AUC).

[6]For the model under consideration, ablating all combinations of two units would result in $8.5{\cdot}10^4$ different experiments

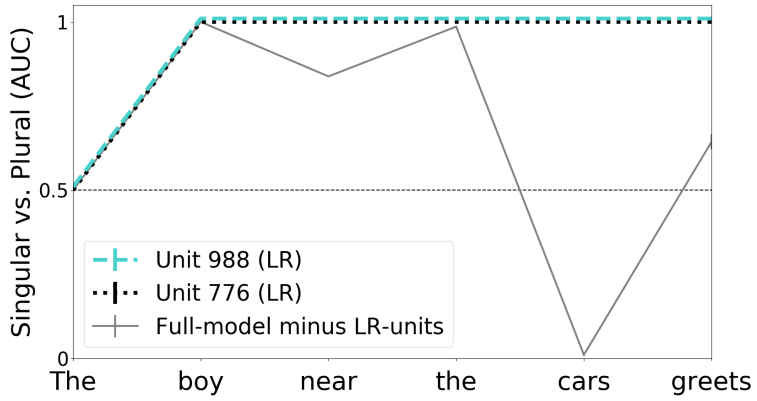


Figure 6.3: AUC for diagnostic classifiers trained on subject time step memory cell activations of unit 988 and unit 766 (green and black dashed lines, respectively) and the rest of the model units (black lines). The x-axis represents the time step that the diagnostic models were tested on.

6.4.1 Short and long-distance number information

Lakretz et al. (2019) train three separate linear models: one for the plural unit, one for the singular unit and one for the rest of the model (without the singular and plural unit). A replication of their plot can be found in Figure 6.3.

The experiments confirm that the grammatical number of the subject can reliably be decoded from the long-distance number units 988 and 776, throughout the whole dependency. The grammatical number of subject can, at the subject time step, also be encoded from the rest of the network. When the trained model is used on activations of the rest of the sentence, however, its accuracy goes down. At the point where the attractor occurs, it consistently predicts the *opposite number*, corresponding to the grammatical number of attractor. This suggests that the information decoded by the linear model is a more short-distance representation of number, which is sensitive to the last encountered noun. The fact that in the ablation experiment no units were found to have a strong impact on short-distance number relations indicates that the representation of short-distance number information is represented in a more distributed rather than highly local fashion.

6.4.2 Ablating short-distance units

Considering the weights of the trained diagnostic classifiers, Lakretz et al. (2019) find 10 units that seemed to play a role in the encoding of short-distance number information. Further ablation experiments, in which either both the SR and LR number units, or only the SR number units were ablated, confirm the role of these

neurons: both experiments result in a significant reduction in task performance on the easier NA tasks, while random equi-sized ablations do not.

6.5 Syntax units

In the previous two sections, we saw two types of number encodings arise: a syntax sensitive encoding, which is extremely local and persists over long-distance relationships, and a more distributed encoding, which appears to encode the number of recently encountered nouns. How the dynamics of these cells are controlled by the rest of the model, however, remains unclear. In other words, we do not know which are the *syntax-aware* units that control the opening and closing of the gates at the relevant moments. At this point in time, I can not present a detailed analysis of the syntactic system controlling the network. But, in the remainder of this chapter, I will discuss a few experiments that are targeted to finding such syntax units.

6.5.1 Tree-depth prediction

To identify syntax units, we do again a diagnostic classification experiment: we train a regularised regression model to predict the *syntactic tree-depth* from the hidden-state activity of all units.[7]

Data To formalise the syntactic tree-depth, we follow the procedure of Nelson et al. (2017). They define the syntactic tree-depth at any point in a sentence as the number of open syntactic nodes in its syntactic parse tree. We generate a set of sentences with various unambiguous syntactic structures and annotate them with their syntactic depth. To decorrelate depth from position, we sample data points from this set that uniformly cover all position-depth combinations within positions 7-12 and depths 3-8. The final data set contains 4033 positions (sampled from 1303 sentences).[8]

Results To train the diagnostic classifiers, we use a nested 5-fold cross-validation procedure, in which we add word frequency as a covariate to the model. This experiment shows that syntactic depth can be decoded from the model ($R^2 = 0.85 \pm 0.009$), and that word frequency had a negligible effect.

[7]For the reader who wonders why we do not run these experiments on the memory cells of the model as well: we are looking for syntax units that can control the gate values of the SR and LR number units. These gate values of are conditioned on the hidden state of the model, not the memory cell.

[8]Like the previous data sets, also this data set is available at `https://github.com/FAIRNS/Number_and_syntax_units_in_LSTM_LMs`

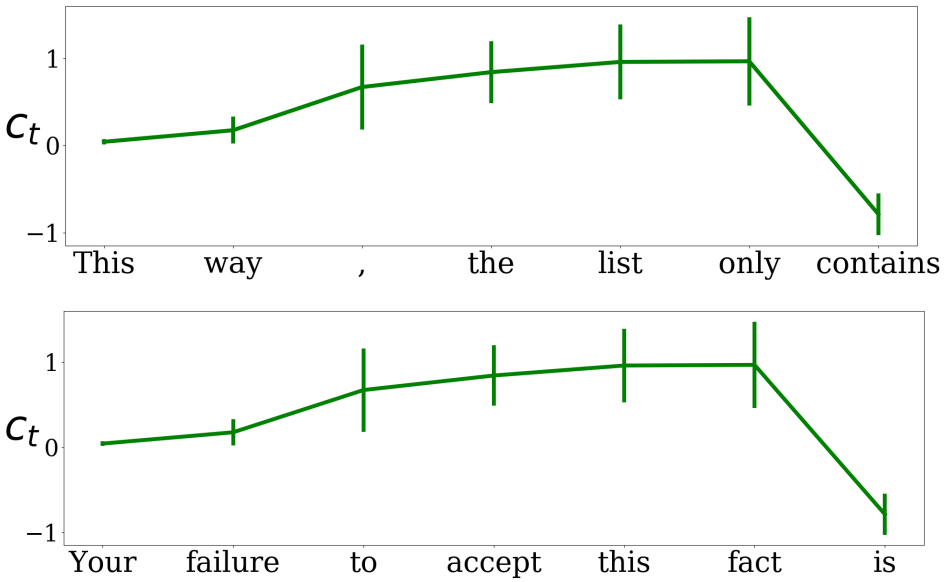


Figure 6.4: Behaviour of unit 1150

6.5.2 Behaviour of syntax units

By analysing the weights in the trained diagnostic classifiers, Lakretz et al. (2019) find a small set of units that have relatively high weights, which they denote as *syntax units*.

Ablation To confirm the role of the syntax units, Lakretz et al. (2019) perform again an ablation experiment. They find that ablating all syntax cells results in a significant performant reduction of all NA-tasks with an interfering noun, compared to random equi-sized ablations.

Behaviour Most of the syntax units do not exhibit a behaviour that is easy to interpret. Unit 1150 forms an exception: the activity of this unit increases throughout the subject-verb dependency and drops abruptly when the verb arrives. In Figure 6.4, I plotted the cell state activation of unit 1150 during two sentences from the corpus used in the previous chapter.

6.6 Conclusion

In this chapter, I described a study that uses neuron ablation and diagnostic classification to understand the mechanisms that an LSTM language model uses to process long-distance subject-verb agreement. For this study, we used a new, controlled data set that contains subject-verb agreement problems where the

subject and verb are separated by different types of sentential material of different lengths.

Like in the diagnostic classification study presented in the previous chapter, this study reveals two different mechanisms to encode number information. A group of *short-distance* units together encode short distance number information, used when the subject and verb are close together in the sentence. *Long-distance* number information, on the other hand, is encoded in a very extremely local fashion, by just two single units. Ablating these units causes substantial drops in the accuracy of the model on sentences that contain long-distance number agreement problems, for singular and plural subjects, respectively. We show how these two units carry singular and plural information over time through use of their gates.

To understand the source of mistakes the model makes on number agreement problems, I also contrasted the behaviour of the long-distance number units on sentences for which the model makes correct and incorrect NA-tasks predictions (similar to the previous chapter). For the mixed Linzen stimuli, we observed a difference between these two cases, in line with the results of the previous chapter. For the controlled NounPP data, however, the long-distance units always reliably encode the number of the grammatical subject, also when the model's verb form prediction is incorrect. This suggests that in such cases the model is confused by other information stored elsewhere in the network, which could either the short-distance number information stored or a different mechanism that stores long-distance number information in a more distributed way. Further uncovering the mechanics of the networks by investigating how the different mechanisms that store number information ion interact with each other could be an interesting direction for future work.

Lastly, with a combination of diagnostic classification and unit ablation, we found a number of *syntax* units, that appear to be involved in the encoding the grammatical structure of sentences. One of these units have high connection weights with the number units; one of them appears to encode the main subject-verb dependency, across various syntactic constructions. There are still many open questions concerning the role of these syntax units. How exactly do they monitor syntactic depth, and what does that have to do with the interaction between the short and long-distance units? And are the same units used across different syntactic dependencies? I would gladly further investigate these issues in the future.

Chapter 7

Generalised contextual decomposition

Both the previous two chapters had a very model-centric focus. They described studies considering which components of the model are representing what, and which mechanisms they were using to do so. In this chapter, I take a somewhat orthogonal approach, which is more data-centric. More precisely, I focus on how the activations elicited by input words flow through the model and causally influence its prediction. To track this activation flow (which I will sometimes refer to as *information flow*), I use a generalised version of a technique called contextual decomposition, which I will explain in the next sections. As in the previous chapter, I use this technique to study how long-distance *subject-verb* relationships are processed by an LSTM language model.[1]

Chapter outline In the next section, I first describe contextual decomposition, as proposed by Murdoch et al. (2018). In Section 7.2, I describe our generalisation: *generalised* contextual decomposition. I provide a precise mathematical description for those that wish to understand the intricacies of contextual decomposition and the motivations for its generalisation. To understand the main results of this chapter, however, a global understanding of what contextual decomposition aims to achieve is sufficient. The mathematical details can thus safely be skipped.

[1]The work described in this chapter is part of the master thesis project of Jaap Jumelet. It was written up as a paper of which I am the last author; this paper was accepted at CoNLL2019 and presented there by the first author Jaap Jumelet:

> Jaap Jumelet, Willem Zuidema, and Dieuwke Hupkes. Analysing neural language models: contextual decomposition reveals default reasoning in number and gender assignment. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 1–11, 2019

The idea to use contextual decomposition for language models came from Jaap Jumelet. We together designed the experiments, with almost daily contact concerning which questions to ask, which data to look at and which experiments to run next. Most of the text in this chapter is non-overlapping with the text of the paper, but incidentally I have copied paragraphs or sentences of the paper that were written by me personally.

Section 7.3 contains a brief description of the model and data we use for our experiments; in Section 7.4 and Section 7.5, I report our results. I conclude in Section 7.6.

7.1 Contextual Decomposition

Contextual Decomposition (CD) is an interpretability method proposed by Murdoch et al. (2018). The aim of this method is to track the causal contributions of individual tokens or phrases to the final prediction of a model, without modifying the underlying architecture or using an additional meta-model such as a diagnostic classifier.

To track the contribution of an input token, Murdoch et al. partition all states and gate values of a model into two parts: a *relevant* part, which contains information stemming from the considered input token, and an *irrelevant part*, which contains information coming from other tokens.[2] For brevity, Murdoch et al. refer to the relevant part of the partition with the letter β and to the irrelevant part with the letter γ, a convention which I follow in the remainder of this chapter.

7.1.1 Separating relevant and irrelevant parts

CD defines how the relevant and irrelevant parts of the hidden states of a model – containing contributions from inside and outside the considered token, respectively – are propagated forward with each forward pass of the model. It does so by computing how the activations corresponding to these parts are transformed through the mathematical equations that define the underlying model.

For some cases, this forward propagation is defined almost trivially, for instance if this operation is a linear sum. If a state z is defined as a weighted sum of two parts a and b – thus $z = w_1a + w_2b$ – the contributions of a and b are straightforwardly defined as w_1a and w_2b, respectively.[3] For other operations, such as multiplications and non-linear activation functions, it is less clear which part of the output should be attributed to which input. An LSTM model, the equations of which I repeat below for convenience, contains many of such 'difficult'

[2] In this chapter, I only consider the case in which the contribution of a single input token is computed, however, the method of Murdoch et al. can be straightforwardly extended to consider multiple, potentially non-consecutive input tokens.

[3] Or, in relative terms: $\frac{w_1a}{z}$ and $\frac{w_2b}{z}$.

operations:

$$f_t = \sigma(W_f x_t + V_f h_{t-1} + b_f) \tag{7.1}$$
$$i_t = \sigma(W_i x_t + V_i h_{t-1} + b_i) \tag{7.2}$$
$$o_t = \sigma(W_o x_t + V_o h_{t-1} + b_o) \tag{7.3}$$
$$\tilde{c}_t = \tanh(W_{\tilde{c}} x_t + V_{\tilde{c}} h_{t-1} + b_{\tilde{c}}) \tag{7.4}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{7.5}$$
$$h_t = o_t \odot \tanh(c_t) \tag{7.6}$$
$$z_t = W_o h_t + b_d \tag{7.7}$$
$$p_t = \text{SoftMax}(z_t) \tag{7.8}$$

In particular, the model *gates* (Equation 7.1-7.3) and candidate activations (Equation 7.4) are wrapped by non-linear activation functions, and the model's memory cell and hidden state (Equation 7.5 and 7.6, respectively) are a result of multiplications between multiple components that are themselves also partitioned in *relevant* and *irrelevant* parts.

I now describe how Murdoch et al. (2018) define the partitioning into β and γ of each of the network components described in the equations above, given the partitioning of the components that they are a function of. I start with the easiest of these cases: the output logits (Equation 7.7), which are formed by a simple linear sum, and work my way down to the more difficult cases. Relevant and irrelevant parts of a partition are always defined with respect to a particular *relevant* input token, which I sometimes leave implicit for brevity. I denote the relevant and irrelevant partitions of a particular model component x at time step t by β_t^x and γ_t^x, respectively.

7.1.2 Output logits z_t

The output logits z_t of the model are computed by taking a weighted sum of the last hidden layer state h_t of the model and the decoder bias or intercept b_d (Equation 7.7). Given the partition of this hidden state into a β_t^h and γ_t^h part, the partitioning of z_t into β_t^z and γ_t^z is thus straightforwardly defined as:

$$z_t = W_o h_t + b_d \tag{7.9}$$
$$= W_o \beta_t^h + W_o \gamma_t^h + b_d \tag{7.10}$$
$$= \beta_t^z + \gamma_t^z + b_d \tag{7.11}$$

where β_t^z provides the final quantitative score of the contribution of the relevant token to the logit.

7.1.3 Hidden state h_t

The hidden state h_t of an LSTM model at time step t is computed by applying a non-linear activation function to the memory cell state c_t and multiplying the outcome with the output gate o_t (Equation 7.6). Rewriting this in terms of β_t^c and γ_t^c gives:

$$o_t \odot \tanh(c_t) = o_t \odot \tanh(\beta_t^c + \gamma_t^c) \tag{7.12}$$

Shapley values

To compute how the two partitions β_t^c and γ_t^c of the memory cell contribute through the tanh function that wraps them, Murdoch et al. (2018) make use of a concept known from cooperative game theory: *Shapley values* (Shapley, 1953). Given a cooperative game, the Shapley values reflect how the total gain of the game should be distributed over the players of this game. Seeing the tanh in Equation 7.6 as the game, and two partitions β_t^c and γ_t^c as the players to this game, the Shapley values $\mathcal{S}_{\beta_t^c}$ and $\mathcal{S}_{\gamma_t^c}$ of β^c and γ^c can thus be seen as their contributions to the gain of the game, which is given by $\tanh(\beta^c + \gamma^c)$:

$$\tanh(\beta_t^c + \gamma_t^c) = \mathcal{S}_{\beta_t^c} + \mathcal{S}_{\gamma_t^c} \tag{7.13}$$

Computing Shapley values

The first step in the computation of the Shapley value of a 'player' y_i considering the 'game' $\tanh(y_1 + y_2 + \cdots + y_n)$, is computing all possible permutations Π_Y of $Y = \{y_1, y_2, \cdots, y_n\}$ and considering all prefixes of those permutations up until y_i.

E.g. when $Y = \{y_1, y_2, y_3\}$, there are six such permutations:

$$\begin{array}{lll} y_1y_2y_3 & y_2y_1y_3 & y_3y_1y_2 \\ y_1y_3y_2 & y_2y_3y_1 & y_3y_2y_1 \end{array}$$

All these six permutations have a prefix that stops at y_i. For instance, for y_1 those prefixes are:

$$\begin{array}{lll} \{\} & \{y_2\} & \{y_3\} \\ \{\} & \{y_2, y_3\} & \{y_3, y_2\} \end{array}$$

For all these subsets U_{y_1}, we compute the gain of adding y_1 to this subset. For example, if we consider subset $\{y_3, y_2\}$, this gain is:

$$G(y_1, \{y_3, y_4\}) = \tanh(y_3 + y_4 + y_1) - \tanh(y_3 + y_4)$$

The Shapley value of y_i, which I denote with $\mathcal{S}_{y_i}$, is the mean of the gains for all previously defined ordered subsets Π_{y_i}:

$$\mathcal{S}_{y_i} = \frac{1}{|Y|!} \sum_{U_{y_i} \in \Pi_{y_i}} G(y_i, U) \tag{7.14}$$

Note that from a computational perspective, many of the permutations yield the same value. It is thus not necessary to compute them all.[4]

Shapley values of β_t^c

To compute the Shapley values of β_t^c and γ_t^c in Equation 7.13, only two subsets need to be considered, respectively:

$$S_{\beta_t^c} = \frac{1}{2}\Big(\big(\tanh(\gamma_t^c + \beta_t^c) - \tanh(\gamma_t^c)\big) + \tanh(\beta_t^c)\Big) \tag{7.15}$$

$$S_{\gamma_t^c} = \frac{1}{2}\Big(\big(\tanh(\beta_t^c + \gamma_t^c) - \tanh(\beta_t^c)\big) + \tanh(\gamma_t^c)\Big) \tag{7.16}$$

The decomposition of h_t

Based on this assignment of contributions of the tanh, Murdoch et al. (2018) define the decomposition of h_t into β_t^c and γ_t^c as follows:

$$\begin{aligned} h_t &= o_t \odot \tanh(c_t) && (7.17)\\ &= o_t \odot \tanh(\beta_t^c + \gamma_t^c) && (7.18)\\ &= o_t \odot (\mathcal{S}_{\beta_t^c} + \mathcal{S}_{\gamma_t^c}) && (7.19)\\ &= o_t \odot \mathcal{S}_{\beta_t^c} + o_t \odot \mathcal{S}_{\gamma_t^c} && (7.20)\\ &= \beta_t^h + \gamma_t^h && (7.21) \end{aligned}$$

Note that Murdoch et al. choose to not decompose the output gate, which could be done following a similar procedure. They report that this decomposition did not empirically improve results. In our experiments, we reverted that decision and decomposed also the output gate. Later, in subsection 7.2.2, I expand upon this decision.

[4]NB: From a mathematical perspective, Shapley values are most straight-forwardly defined in terms of prefixes of permutations. Another in terms of permutations way of understanding Shapley values is to think about the contribution of a player as a weighted average of its contribution to all possible configurations of players that can be formed with a subset of the total number of players. To assure that the total of all contributions from the individual components sums up to the actual total contribution, all possible configurations are multiplied with a number that related to the size of the configuration, resulting in their weighing factor. For every configuration, this multiplication number is exactly equal to the number of prefixes that we defined above.

7.1.4 Memory cell c_t

The memory cell of the LSTM model is a sum of two products that both contain two terms that are partitioned into relevant parts β and irrelevant parts γ, and a part stemming from the bias terms of the model, which I denote $\mathcal{S}_b$:

$$\begin{aligned} c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && (7.22) \\ &= (\beta_t^f + \gamma_t^f + \mathcal{S}_{b_f}) \odot (\beta_{t-1}^c + \gamma_{t-1}^c + \mathcal{S}_{b_c}) \\ &\quad + (\beta_t^i + \gamma_t^i + \mathcal{S}_{b_i}) \odot (\beta_t^{\tilde{c}} + \gamma_t^{\tilde{c}} + \mathcal{S}_{b_{\tilde{c}}}) && (7.23) \end{aligned}$$

Selecting interactions This cross-product (Equation 7.23) expresses c_t as a sum of 18 *interactions* between the β and γ and $\mathcal{S}_b$ terms that are already computed. Of these interactions, Murdoch et al. (2018) label all products containing β-terms and bias terms (6 in total) as relevant for the next step; all other interactions end up in the γ_t^c. Thus:

$$\beta_t^c = \beta_t^f \beta_{t-1}^c + \beta_t^f \mathcal{S}_{b_c} + \mathcal{S}_{b_f} \beta_{t-1}^c + \beta_t^i \beta_t^{\tilde{c}} + \beta_t^i \mathcal{S}_{b_{\tilde{c}}} + \mathcal{S}_{b_i} \beta_t^{\tilde{c}} \tag{7.24}$$

and

$$\begin{aligned} \gamma_t^c = {} & \beta_t^f \gamma_{t-1}^c + \gamma_t^f \beta_{t-1}^c + \gamma_t^f \gamma_{t-1}^c + \gamma_t^f \mathcal{S}_{b_c} + \mathcal{S}_{b_f} \gamma_{t-1}^c + \mathcal{S}_{b_f} \mathcal{S}_{b_c} \\ & + \beta_t^i \gamma_t^{\tilde{c}} \; + \; \gamma_t^i \beta_t^{\tilde{c}} \; + \; \gamma_t^i \gamma_t^{\tilde{c}} \; + \gamma_t^i \mathcal{S}_{b_{\tilde{c}}} + \; \mathcal{S}_{b_i} \gamma_t^{\tilde{c}} \; + \mathcal{S}_{b_i} \mathcal{S}_{b_{\tilde{c}}} \end{aligned} \tag{7.25}$$

Murdoch et al. (2018) present the distribution of interactions over β and γ terms as an obvious choice. However, later in this chapter we will see that which interactions are considered relevant has important implications for the type of research questions that can be answered with contextual decomposition. We therefore propose to relax this assumption and propose an alternative, more general version of contextual decomposition, in which it can be more flexibly chosen which interactions are considered relevant.

Interaction sets In what follows, for brevity I sometimes collectively refer to all interactions containing only β terms with the notation β-β; the interactions between β and bias terms with β-*bias*; the interactions between the β's of model states c and $\tilde{c}$ and the γ terms of the gates with β_s-γ_g and the interactions between the γ's of c and $\tilde{c}$ with β_g-γ_s.

7.1.5 Gates f_t, o_t and i_t, and candidate activation $\tilde{c}$

The last decomposition left is the decomposition of the gate values. For those, Murdoch et al. (2018) use again Shapley values. For instance, for the forget gate,

this gives:

$$\begin{aligned} f_t &= \sigma(W_f x_t + V_f h_{t-1} + b_f) \\ &= \sigma(W_f x_t + V_f(\beta^h_{t-1} + \gamma^h_{t-1}) + b_f) \\ &= \mathcal{S}_{W_f x_t} + \mathcal{S}_{V_f \beta^h_{t-1}} + \mathcal{S}_{V_f \gamma^h_{t-1}} + \mathcal{S}_{b_f} \end{aligned} \tag{7.26}$$

If x_t is a relevant input token (for which the contribution is to be computed), it is put into the β part of the partition and is lumped together with the other β part *before* computing its Shapley value. In that case, β^f_t and γ^f_t are thus:

$$\beta^f_t = \mathcal{S}_{W_f x_t + V_f \beta^h_{t-1}} \tag{7.27}$$

$$\gamma^f_t = \mathcal{S}_{V_f \gamma^h_{t-1}} \tag{7.28}$$

For any other token, the contribution of x_t is instead irrelevant. In that case, β^f_t contains only one term, and the term containing x_t is put in the γ partition:

$$\beta^f_t = \mathcal{S}_{V_f \beta^h_{t-1}} \tag{7.29}$$

$$\gamma^f_t = \mathcal{S}_{W_f x_t + V_f \gamma^h_{t-1}} \tag{7.30}$$

The output gate o_t and input gate i_t can be decomposed analogously.

7.1.6 Shapley approximation

Computing the exact Shapley values for a particular input word requires keeping track of all the components in all β and γ terms separately, before summing them up to the final β and γ contribution. As the number of such terms grows exponentially with the length of the sentence, and the computational complexity of Shapley values grows exponentially with the number of terms, this is not computationally feasible.

Computing Shapley values of sums Murdoch et al. solve this computational issue by computing the Shapley value of the sums of relevant and irrelevant components, respectively, instead of taking the sum of the Shapley values. The resulting contributions are thus not the sum of all the contributions of relevant or irrelevant components, but instead reflect the contribution of all these components considered jointly.

To illustrate this simplification, I invite the reader to reconsider the step from Equation 7.18 to Equation 7.19. In this step, the contributions of β^c_t and γ^t_c components are assigned by computing the Shapley value of their sum, rather than considering the many parts they are made up from (as defined in Equation 7.24 and 7.25.

Another example concerns the difference between Equation 7.26, and Equation 7.27 and 7.30: a full decomposition would require first computing the contributions of the four components in Equation 7.26 – requiring to consider 15 configurations in total – and then summing them; Instead Murdoch et al. (2018) approximate the sum $S_{W_f x_t} + S_{V_f \beta^h_{t-1}}$ by first summing the two terms and computing the Shapley value $\mathcal{S}_{W_f x_t + V_f \beta^h_{t-1}}$ of their joint contribution (7 configurations).

It is important to note that a consequence of the approximation described above is that the sum of the contributions of all input tokens does not sum up to the total logit anymore. I discuss this issue in more detail in Section 7.4.

Intercepts decompositions Another simplification step done by Murdoch et al. to facilitate the Shapley approximation concerns the number of permutations they consider for their Shapley computation. As explained before, a full Shapley computation for a particular element requires considering all possible permutations of elements and averaging over the gains of adding the element of interest to the prefixes of these permutations up until the element. To reduce the number of prefixes for which these values have to be computed, Murdoch et al., use only permutations that have the intercept term at the first position and exclude all others. Later in this chapter, in Section 7.2.2, I discuss the impact of this decision.

7.2 Generalised contextual decomposition

Murdoch et al. (2018) empirically show that their setup works well for the analysis of a model that performs sentiment analysis. With CD, they can reliably identify the sentiment of different input words and subphrases. However, CD contains several choices that can have a substantial impact on the resulting conclusions. These choices concern not only their choice of approximation, but also – more importantly – which *interactions* within the model they consider relevant. In this section, I take a closer look at such choices and propose *generalised contextual decomposition* (GCD), which generalises CD to be useful in more scenarios.

The two main differences between GCD and CD are:

1. GCD allows for flexibly choosing which interaction sets are considered relevant, depending on the exact research question asked with a particular experiment.

2. GCD differs in some technical aspects from CD, in particular, it uses better Shapley approximations and decomposes *all* components of the model, including the output gate.

In the next sections, I discuss these changes, starting from the implications of choosing different interaction sets (subsection 7.2.1) and then continuing with a discussion of the technical 'fixes' (subsection 7.2.2).

7.2.1 Choosing interaction sets

As can be seen in Equation 7.24, Murdoch et al.'s relevant interaction set includes only interactions between β and bias terms (β-β and β-bias). They present this division of interactions between relevant and irrelevant as an obvious choice, but I argue that, depending on the question that a researcher seeks to answer, some of the disregarded interactions may in fact be important. Consider, for instance, the verb prediction in the number agreement tasks that I considered in the previous two chapters. While the verb *form* depends only on the subject, *when* this information should surface depends on the material in between. In Murdoch et al.'s setup, this information is disregarded with the irrelevant considered β-γ interactions.

To address this isue, in GCD we allow a more flexible assignment of relevant and irrelevant that can thus be varied from experiment to experiment. In our own experiments, we consider three ways of defining these interaction sets, which I will explain below.

Our default interaction set `IN`

The interaction set `IN` is designed to solve exactly the problem mentioned before: the contribution of a particular input token does not only depend on the token itself, but also on its interactions with other input tokens. This becomes even more concrete considering the long-distance number units described in the previous chapter. These number units reliably store information about the subject number in long-distance dependencies, but their gating behaviour is primarily controlled by different units: the syntax units. While the initial activation of this syntax unit may still be attributed to the subject, its decrease of activation when the subject information should surface (see Figure 6.4) depends on the interactions with intervening sentential material. If these interactions are disregarded, as in the setup of Murdoch et al., the opening of the output gates of the number units will be considered irrelevant, and the number information stored in the LR units will thus also be put in the irrelevant partition of the hidden layer activations. While we know that the long-distance number units are crucial for processing the long-distance relationships, their contribution would thus not be counted if al β-γ are considered irrelevant.

In our default `IN` interaction set, we therefore include the interactions between γ parts of the gate with the β parts of the (candidate) memory cell activations ($\gamma_t^f \beta_{t-1}^c$ and $\gamma_t^i \beta_{t-1}^{\tilde{c}}$, collectively referred to as β_s-γ_g). In this interaction set we furthermore follow the addition of the CD authors in a follow-up paper (Singh et al., 2019), to consider also the interactions between the bias terms at the time step where the input word is the word for which the information flow is computed.

Intercept interaction sets `Intercept*` and ¬`Intercept`

Flexibly assigning interaction sets allows us to also easily test the impact of the bias terms of the network (the intercepts of the gates and candidate activation equations). We devise two interaction sets to study their influence on the model prediction.

¬`Intercept` In the first interaction set, we leave out all interactions with the intercepts of the network. We call this interaction set ¬`Intercept`. In Equation 7.24, this means excluding both $\beta_t^f \mathcal{S}_{b_c}$ and $\mathcal{S}_{b_i}\beta_t^{\tilde{c}}$ (b-b), which will be put in the γ part of the c_t partition.

`Intercept*` We also consider an interaction set that includes *only* intercept interactions. For this interaction set, the initial model state is put into the β partition, x_t is always put in the irrelevant γ partitions. The interaction set is otherwise identical to `IN`.

7.2.2 Technical fixes

In the previous section, in which I explained CD, I pointed out several simplifications and approximations implemented by Murdoch et al. to simplify the computational complexity of CD. Some of these simplifications, I argue, may have an undesirable impact in the language modelling scenario that we consider here; we therefore decided to revert them. Below I list the two main 'technical fixes' of CD that we did to arrive at GCD.

Subsets considered in Shapley approximation

The first technical fix considers the approximation of the Shapley values of the gate contributions. To reduce the complexity of this approximation, Murdoch et al. restrict the number of subsets that they average over to compute the Shapley values that define the gate partitioning. In particular, they consider only (ordered) subsets in which the bias is the first term. In their article, they report that they find improvements with this configuration. However, this protocol magnifies the contribution of the bias term (see Figure 7.1 for a worked-out example that illustrates this). As the role of the bias term is one of the important points of focus in our experiments, we therefore decide to remove this simplification, and, with a slight loss of computational efficiency, consider all possible configurations instead.

The decomposition of the output gate

Murdoch et al. did not decompose the output gates of the model, they report that they empirically observed that decomposing this gate did not lead to improved

Shapley values, full vs bias-first decomposition

In this example, I consider the computation of the contributions of β, γ and the bias term b to a gate value, described by:

$$g = \tanh(\beta + \gamma + b)$$

Full decomposition

The Shapley value of β, γ and b is computed by, for all possible orderings of terms, considering their subset up until that component, and then computing the gain of adding this component. For instance, for the bias term b:

$$\begin{aligned}\mathcal{S}_b = {}^1\!/_6\big(&\tanh(\beta+\gamma+b) - \tanh(\beta+\gamma)\\ &+ \tanh(\gamma+\beta+b) - \tanh(\gamma+\beta)\\ &+ \tanh(\gamma+b) - \tanh(\gamma)\\ &+ \tanh(\beta+b) - \tanh(\beta)\\ &+ \tanh(b)\\ &+ \tanh(b)\big)\end{aligned}$$

Similarly, the contribution for β is:

$$\begin{aligned}\mathcal{S}_\beta = {}^1\!/_6\big(&\tanh(b+\gamma+\beta) - \tanh(b+\gamma)\\ &+ \tanh(\gamma+b+\beta) - \tanh(\gamma+b)\\ &+ \tanh(\gamma+\beta) - \tanh(\gamma)\\ &+ \tanh(b+\beta) - \tanh(b)\\ &+ \tanh(\beta)\\ &+ \tanh(\beta)\big)\end{aligned}$$

Fixed-bias decomposition

When only permutations are considered of which the bias b is the first term, the computed contribution of the bias does not depend anymore on terms containing β and γ:

$$\mathcal{S}_b = \frac{1}{2}\Big(\tanh(\mathrm{b}) + \tanh(\mathrm{b})\Big)$$

Also for the contribution of β fewer terms are taken into account:

$$\begin{aligned}\mathcal{S}_\beta = {}^1\!/_2\big(&\tanh(b+\gamma+\beta) - \tanh(b+\gamma)\\ &+ \tanh(b+\beta) - \tanh(b)\big)\end{aligned}$$

As can be seen in the two computations above, when only permutations are taken into account that have the bias term b at the first position, the contribution of the bias term defaults to $\tanh(b)$. Unless the other terms are 0, the magnitude of this contribution will be larger than the gain of adding b to the other terms before computing the tanh.
For instance, consider $\beta = \gamma = b = 0.4$. Using the full Shapley decomposition, the contributions of those three components will be equal: $\mathcal{S}_\beta = \mathcal{S}_\gamma = \mathcal{S}_b = 0.28$. When the fixed-bias decomposition is used, $\mathcal{S}_b = 0.38$ and the remaining gain is divided by β and γ: $\mathcal{S}_\beta = \mathcal{S}_\gamma = 0.23$.

Figure 7.1: Murdoch et al. (2018) do not fully decompose the gate values; instead, they only consider permutations of the input components that have b at the first position. Generally, this magnifies the effect of the bias term, which is why in GCD we decided to revert this decision.

results. I have no empirical evidence against this decision, but for completeness we decompose all gates in GCD, including the output gate. We decompose the output gate as described in subsection 7.1.5.

7.3 Model and data

I now present a series of experiments conducted with GCD. Both the model and data for these experiments are identical to the ones used in the previous chapter. I provide a brief description for convenience of the reader.

Model We investigate the pre-trained model of Gulordava et al. (2018), which is a 2-layer LSTM model with 650 units in both the hidden and embedding layers.[5]

Data For our experiments we use the synthetic data sets introduced in the previous chapter, focusing in particular on the NounPP data set. To remind the reader, this data set contains NA problems where the grammatical subject and main verb of the sentence are separated by a prepositional phrase containing a noun-phrase. The number of both the subject and the intervening noun are varied systematically, resulting in four different *conditions*:

SS	*congruent*	The **boy** near the *car knows* ...
PP	*congruent*	The **boys** near the *cars know* ...
SP	*incongruent*	The **boy** near the *cars knows* ...
PS	*incongruent*	The **boys** near the *car know* ...

We do not regenerate the sentences in this data set, but use the exact same set used also in the previous chapter.

7.4 Token contributions

In our first experiment we consider the contributions of all tokens preceding the verb to the prediction of the verb, using our `IN` interaction set. This allows us to trace which tokens in the sentence the model used to come to its prediction.

[5]Most of the results reported below are also confirmed for the state-of-the-art recurrent language model provided online by Jozefowicz et al. (2016), I will not report those results in this chapter.

7.4.1 Decomposition matrix

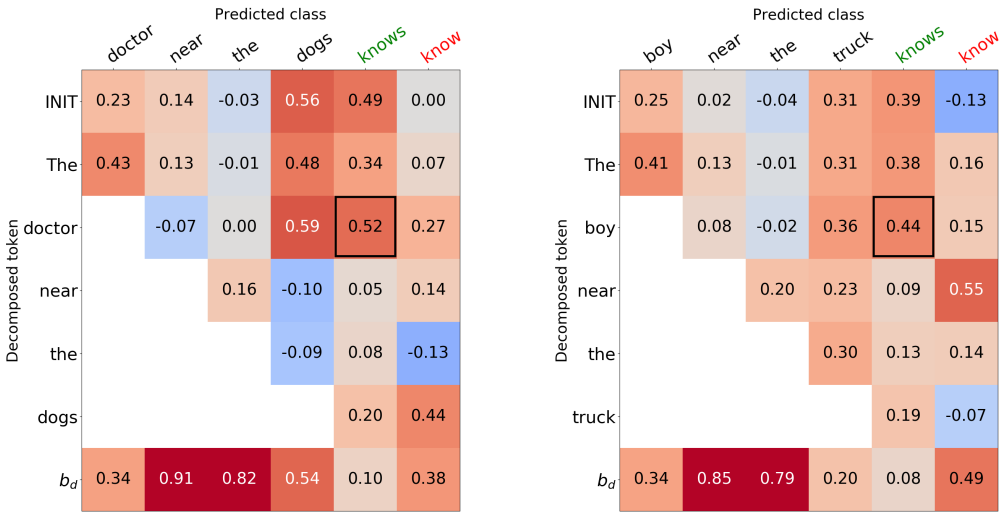
To visualise our results, we create a *decomposition matrix*, the rows of which represent the input tokens; the columns the predicted tokens. Every cell (i,j) of this matrix thus represents the relative contribution of an input token x_i to the logit of an output y_j. Aside from the input tokens, the rows of the decomposition matrix also contain the contributions of the *initial hidden state* and the *decoder bias* $\mathbf{b_d}$. To transform the contribution scores into *relative* contribution scores, I normalised all token contributions to a particular output token by the logit z_t^i assigned to the relevant output token.[6]

Reading decomposition matrices As the values in the columns of a decomposition matrix represent contributions to the same output logit, they are directly comparable. The normalisation allows us to also compare relations within entire columns. For instance, a column by column comparison might reveal that token A was more important for the prediction of token Y than token B, whereas token B was more important for the prediction of token X than token A. It is important to note that comparisons between different cells in the same row are non-sensible, because rows have different normalisation factors.

In Figure 7.2, I show two examples of decomposition matrices. From this plot can be seen that contributions can be both positive and negative. A negative contribution indicates that a particular input token pushes the logit for the considered output token down, while a positive input means it makes the logit larger. In what follows, I highlight a few salient points in the decomposition matrix, before continuing to discussing decomposition matrices averaged over many sentences.

Verb form columns For our particular case, we are most interested in the last two columns of the matrix, that represent the contributions of the input tokens to the probability mass of the correct and incorrect verb form of the network. The contribution of the subject to the verb is marked with a black box. For both sentences, we see that the subject contribution is the highest contribution in its column for these sentences, indicating that most of the probability mass in the logit is indeed stemming from the subject. For both sentences, also the information flowing from the initial model state contributes substantially to the logit of the singular verb form, almost as much as the subject. Interestingly, the

[6] As explained in subsection 7.1.6, the sum of all token contributions does not sum up to the complete logit, because the Shapley values are approximated instead of computed exactly. Normalisation could also be done by dividing by the sum of the token contributions, instead of by the total logit, which may sometimes paint a slightly different picture. In this chapter, I decided to show only pictures that are normalised with the logits, for a more elaborate exploration of the impact of different normalisations, I refer to the (yet unpublished) master thesis of Jaap Jumelet.



(a) Decomposition matrix for the sentence *The doctor near the dogs knows*

(b) Decomposition matrix for the sentence *The boy near the truck knows*

Figure 7.2: Decomposition matrices for two different sentences from the NounPP corpus. The contribution values are computed with the `IN` interaction-set, and the columns are divided by the total logit z of the token in the output layer of the model. The first and last row represent the contributions of the initial model state and the decoder bias, respectively.

subject contributes also to the probability mass of the *incorrect* verb form.

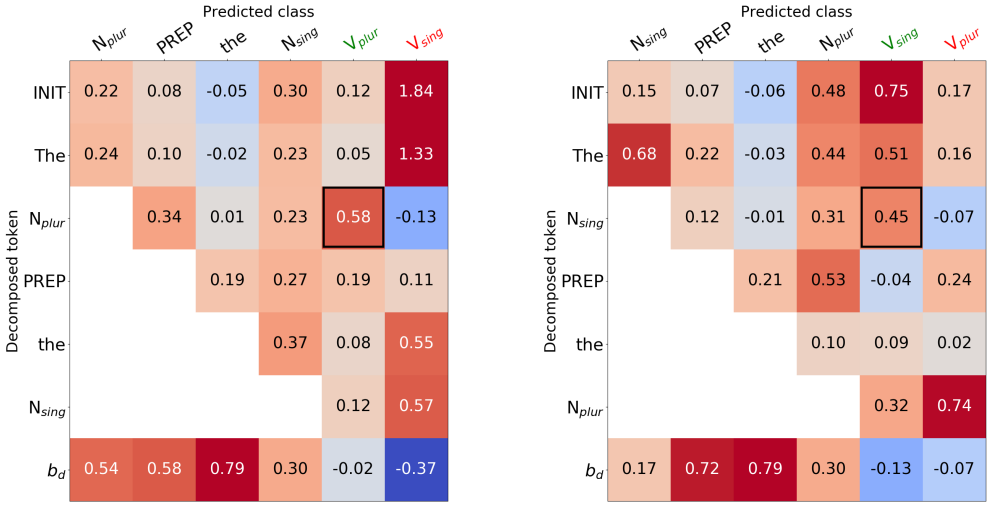
Decoder bias contributions In the last row, we see the contribution of the *decoder bias* b_d (or intercept). This bias is a fixed term, added to the output layer of the network (see Equation 7.7); it does not vary across sentences. The value differences between the values for b_d for the tokens *know* and *knows* thus stem solely from the logit value the network assigned to those tokens: for the sentence in Figure 7.2a, the logit of the token *know* was smaller than it was for the same token in the sentence of Figure 7.2b; The contribution of the decoder bias to the logit in Figure 7.2a is therefore *higher* than the contribution of the decoder bias in Figure 7.2b. Analogously, it can be concluded that the logit for the plural verb form *know* was higher in Figure 7.2a than in Figure 7.2b.

Comparing values It is important to note that the difference in logit values between sentences but also between tokens within one sentence means that value comparisons across columns are not salient without context. For instance, in Figure 7.2a we see that the contribution of the subject to the logit of the singular verb form is twice as big as its contribution to the logit of the plural verb form, but this does not tell us anything about the relative magnitude of this contribution.

If the logit of the plural verb form is much larger than the logit of the singular verb form, the activation flowing from the subject to the plural verb form may still be larger than the activation flowing to the singular form, even though it is relatively speaking smaller. In the rest of the analysis, I therefore stick to within column comparisons.

7.4.2 Average decomposition matrices

After having explained to the reader the salient points in the decomposition matrices of two single sentences, I would now like to move to discussing decomposition matrices that are averaged over multiple sentences. In Figure 7.3a and Figure 7.3b, I show the average of all decomposition matrices for the PS and SP conditions of the NounPP data set, respectively.



(a) NounPP, PS (b) NounPP, SP

Figure 7.3

Noun contributions In both the SP and PS cases, the subject contribution to the main verb of the sentence is highly positive (boxed cells). On average, the subject contribution to the incorrect verb form is negative, something we did not observe when considering the two single sentences in which the verb was the frequent verb *know*. Also in both conditions, the attracting noun contributes to the probability mass of the incorrect verb form, that has the same number as this noun.

Default reasoning The plots show also a remarkable difference between singular and plural sentences. For the plural sentences (left plot), the subject contribution

to the correct verb form is substantially higher than the contributions for all other components, including the initial hidden state and the decoder bias. For the singular condition (right plot), the subject also contributes to the generation of the correct verb form, but the main contributions stem from elements outside of the subject. The highest contribution to the correct verb form comes from the initial state of the network; Also the contribution of the determiner is high.

A similar pattern is apparent in the final column of the plot, which represents the contributions to the logit of the incorrect verb form. While the main contribution to the plural logit stems primarily from the plural attractor noun, the logit of the singular verb form instead receives strong contributions from non-numbered tokens, such as the initial state of the network. These values suggest that predicting a plural verb requires explicit evidence from a plural noun, whereas predicting a singular verb form could be seen as the *default* behaviour of the model. Interestingly, the decoder bias of the model appears to encode a different default: the relative contribution of the decoder to singular verbs is much lower than the contribution to plural verbs, which is consistent with the earlier drawn conclusions that plural verb forms are more frequent than singular verb forms.

7.5 Information ablation

In the previous experiment, we focused on the contributions of different input words to the model's prediction. Now, I focus directly on how the model's prediction is driven by different parts of the input, by doing *interaction ablation*. In other words, I consider how the input tokens are causally linked to the output prediction. To do so, we filter out different information partitions (as specified by GCD) and study how this affects the model's accuracy on the NA tasks.

7.5.1 Subject information

In the first experiment, we consider what happens with the prediction of the model when we filter all information that is not coming from the subject. If the subject is the primary driver for the prediction of the correct verb form, filtering out other information should not affect a model's ability to form the right prediction. On the contrary: it should help the model, as we remove all potentially confusing information that might distract the model. If, instead, the prediction of the verb is not causally linked to the subject, but the model is using a heuristic that requires information from the rest of the sentence (or its initial state), we expect a decrease in accuracy.

The results of this experiment, shown in Table 7.1 (column `IN`), confirm the conclusion drawn in the previous section. For all plural conditions (in back), the accuracy goes up. For plural conditions, filtering information is thus helpful,

			GCD		
Task	**Condition**	FULL	`IN`	`Intercept`*	¬`Intercept`
Simple	S	100 (100)	73.3 (91.3)	97.3 (100)	69.7 (86.3)
Simple	P	100 (100)	100 (100)	32.7 (7.7)	100 (100)
nounPP	SS	99.2 (99.0)	93.0 (99.7)	99.8 (99.8)	72.7 (88.7)
nounPP	SP	87.2 (94.3)	90.3 (99.3)	98.8 (99.8)	60.5 (83.5)
nounPP	PS	92.0 (83.0)	100 (100)	0.0 (0.0)	100 (100)
nounPP	PP	99.0 (94.8)	100 (99.3)	7.0 (0.5)	99.8 (100)
namePP	SS	99.3 (99.9)	97.7 (91.3)	99.4 (100)	76.2 (90.9)
namePP	PS	68.9 (54.6)	98.3 (98.2)	1.3 (0.0)	99.9 (99.9)

Table 7.1: Accuracies of the model when information flow is selectively pruned. The FULL column indicates model accuracies without pruning. `IN` denotes the condition in which only information from the subject, using the `IN` interaction set, is let through. `Intercept`* indicates accuracies where only the information of the model intercepts is considered. For ¬`Intercept`, instead, all intercept interactions are filtered out. Singular conditions are coloured green. $(\cdot)$ indicates model accuracies of scores without decoder bias, i.e. Dh_t vs $Dh_t + b_d$.

indicating that the prediction of the plural verb is indeed causally linked to the plural subject of the sentence. For singular verbs, almost all accuracies instead go down, indicating that the model was using information from outside the subject to come to the right prediction.

7.5.2 The role of the intercepts

In a second experiment, we focus on the role of the model intercepts (b_i, b_o, b_f and $b_{\tilde{c}}$, which are likely to play a role in the model's default behaviour. In particular, we consider two different setups: one in which we consider *only* the information coming from the intercepts (`Intercept*`) and one where instead all interactions with the intercepts are filtered out (¬`Intercept`).[7]

The results of both experiment, shown in the last two columns of Table 7.1, show that the previously found default effect of the model is to a strong extent encoded in the model intercepts. When only information coming from the intercepts and initial state is considered – all activation flowing from the input embeddings is put in the γ part of the partitions – the accuracy of the plural conditions drops in most cases to almost 0, while the singular accuracies stay close to 1. In the discongruent nounPP condition SP, the model accuracy considering only intercept information is even higher than the full model accuracy. Conversely, when no intercept interactions are taken into account, the accuracy of singular conditions

[7]For a complete description of the interactions included in this set, I refer the reader back to Subsection 7.2.1.

suffers, while the accuracy of plural conditions increases, or stays the same.

7.5.3 The decoder bias

For all previously described experiments, we computed also the model's accuracy without the decoder bias b_d, shown between brackets in all columns of Table 7.1. The results confirm the previous observation that the decoder intercepts pushes towards plural predictions. Removing the decoder bias generally *increases* the accuracy for singular conditions while *decreasing* the accuracy for plural conditions. The decrease in accuracy for the plural conditions is less strong in congruent cases, or when information favouring singular predictions is filtered out (the `in` and ¬`Intercept` columns, suggesting that the plural decoder bias may serve as compensation for the overall singular bias of the network (or the other way around).

7.6 Conclusion

In this chapter, I described contextual decomposition (CD), an interpretability technique that provides a procedure for tracking the impact that activations flowing from input tokens have on the predictions of a model. I criticised several impactful choices in this technique and then introduced a more general version – generalised contextual decomposition (GCD) – that addresses some of the criticisms and provides a more flexible setup that can be adapted based on the modeller's research questions.

I described a series of experiments with GCD concerning subject-verb agreement in a pre-trained neural language model. With these experiments, I illustrated how GCD can be used to track the contributions of different input tokens to the predicted logits of specific output tokens, to study interactions between input tokens and the model's intercepts, and to investigate the causal effect of input tokens and/or model components on the model's prediction.

With these experiments, we uncovered a default effect in the model: the prediction of plural verbs is strongly causally linked to the grammatical subject of the sentence, while singular verb forms do not require explicit evidence from singular nouns. This default appears to be encoded in the learned initial state (h_0, c_0) and the intercept terms b_i, b_f, b_o and $b_{\tilde{c}}$ of the model. Interestingly, the decoder bias b_d of the model encodes a different default. On average, this bias term is higher for plural than for singular verbs, in line with the frequency difference of these verb forms.

An important advantage of GCD with respect to ablation or diagnostic classification is that it can also be used for phenomena where a behavioural setup with a right and wrong answer can not be easily constructed. For instance, to study *negative polarity items* or *anaphora resolution*. For such cases, it is important

what information a model-based a particular prediction on, rather than what this prediction is. Consider, for instance, the case in which a model processes the sentence prefix '*The man asked his niece if . . .*'. Both *he* and *she* are acceptable continuations for this prefix and comparing their probabilities thus does not provide information on whether the model based its prediction on the correct referent. With GCD, this is easily addressed, because it can be computed what are the contributions of the two nouns to the logits of the tokens *he* and *she*.[8] This opens up possibilities a new range of linguistic phenomena that can be investigated, but it also carves out a role for GCD to study important topics such as gender bias in models.

[8]In Jumelet et al. (2019), we presented a study considering anaphora resolution of both unambiguously and stereotypically gendered nouns, I refer the interested reader to Section 4.2 and Section 6 of this article.

Part Three

Guiding models

It is well known that a finite set of input-output pairs can be described in many different ways (Angluin and Smith, 1983; Gold et al., 1967). In the space of all potential solutions, one extreme is to simply memorise all pairs, which does not offer any means to extrapolate to new inputs. On the other end of the spectrum, we find solutions that abide by the principle of minimal description length, which provides much stronger generalisation capacity (Hutter, 2004; Rissanen, 1978). Recurrent neural networks can in theory represent many different solutions on this spectrum. In the previous chapters, we learned that such networks, when trained with back-propagation and gradient descent, can capture interesting aspects of structure and hierarchy. But, at the same time, they are not yet at the point of being plausible approximators of human behaviour. In particular, when it comes to explicit compositional rule learning, models often miss the intended compositional solution and model the training data by using a series of heuristics or shortcuts instead.

In the third and last part of this dissertation, I investigate whether it is possible to *change* the types of solutions that models learn by changing their learning signal. More specifically, I explore the hypothesis that one reason that models find for humans undesirable solutions to their task is that they segment the input in a way that is different from the segmentation intended by the modeller. As a consequence, models may be unable to distinguish salient from non-salient patterns and end up memorising patterns that may seem reasonable given the training data, but are not when considering the underlying system. For instance, taking a linguistic perspective, in the sentence *the man with the hat walks in the park*, we say that *walks in the park* a salient and reusable subsequence, whereas *hat walks in* is not, regardless of how often it may have occurred in a corpus.

To test this hypothesis, I investigate what happens when a model is given explicit information about the 'correct' segmentation. Can we guide a model to exhibit more compositional behaviour without introducing an additional component to it that takes care of structure? If so, what is the impact on the way that its parameter space is organised?

Chapter 8

Attentive Guidance

As a little exercise, imagine teaching a child how to read. I assume that you do not imagine giving her increasingly long sequences of words and letters (inputs) and ask her to vocalise them (the target). Instead, you guide her through the process. You draw her attention by pointing at individual letters, tell her how to pronounce them. You show her how together letters form a word. In other words, you inform her about the individual components of the task of reading, and you tell her how these can be combined.

When we train neural networks, we usually do not provide them this type of information. Instead, we do give them a large set of input sequences and ask them to generate the corresponding output sequences. In this chapter, I first investigate if giving a recurrent model a learning signal that contains more information about how the input should be segmented might change the type of solutions it finds. To give this signal, I will use a model with an *attention mechanism* (see Section 2.2.4).[1] Then, I provide a thorough exploration of the impact of this changed learning signal on the way that the learned solutions are implemented by the network, using many of the techniques proposed and discussed earlier in this dissertation. I investigate how the parameter space is structured, how individual neurons and gates behave and how distributed the implementation of the solution is.[2]

[1]The first part of this chapter is based on the following paper:

> Dieuwke Hupkes, Anand Singh, Kris Korrel, German Kruszewski, and Elia Bruni. Learning compositionally through attentive guidance. In *International Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*, 2019c

The study described in this paper was part of the master thesis of Anand Kumar Singh, who I supervised together with Elia Bruni. Anand Singh provided the initial implementation of the technique we called *attentive guidance*. I reimplemented it in a different code base to confirm the results and run further experiments. The published article was written mostly by mean, there is a large overlap between the text of this article and the current chapter.

[2]The second part of this chapter is based on a June project that I, together with Elia Bruni, ran with 7 master AI students. Elia Bruni and I formulated the experiments, which were

Chapter outline Following my previous chapters, I again start with a description of the data set used for this study. In subsequent sections (8.3 and 8.4) I describe attentive guidance (AG) and the experiments we did to understand its impact on the behaviour of the model. In Section 8.5, 8.6 and 8.7, I describe a comparison between the solutions learned by models trained with and without AG, in terms of the organisation of their parameter space and the function of (groups) of neurons. I conclude in Section 8.8.

8.1 Data

For the investigation described in this chapter, I consider the *lookup table task* introduced by Liška et al. (2018), which I already discussed earlier in this dissertation, in Chapter 2. The atomic operations in this task are very simple, but correctly generalising requires understanding that the inputs are compositional sequences of such operations, which should be sequentially composed.

Liška et al. (2018) showed that vanilla recurrent models rarely correctly infer this compositional structure, even when they receive explicit information concerning the "intermediate outputs" of the operations. This clear failure provides an excellent test bed to explore the impact of changing the learning signal..

8.1.1 Task description

As explained in Section 2.3, the lookup-table task consists in computing the meaning of sequences of lookup tables, which are defined as bijective mappings from the domain of binary string of length L onto itself. For instance, the lookup table `t1` may be defined as:

$$\begin{aligned}
&\texttt{t1}\ 00 \rightarrow 01\\
&\texttt{t1}\ 01 \rightarrow 11\\
&\texttt{t1}\ 10 \rightarrow 10\\
&\texttt{t1}\ 11 \rightarrow 00
\end{aligned}$$

A series of such lookup tables is – together with their input – presented to a neural network model, and the model is then asked to output the outcome of this computation. For instance, a potential input sequence could be `000 t1 t2`; the

conducted and further filled in by the students, whom we met twice a week. The work was published at the BlackboxNLP workshop held at ACL 2019:

> Joris Baan, Jana Leible, Mitja Nikolaus, David Rau, Dennis Ulmer, Tim Baumgärtner, Dieuwke Hupkes, and Elia Bruni. On the realization of compositionality in neural networks. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 127–137, Florence, Italy, 2019. ACL

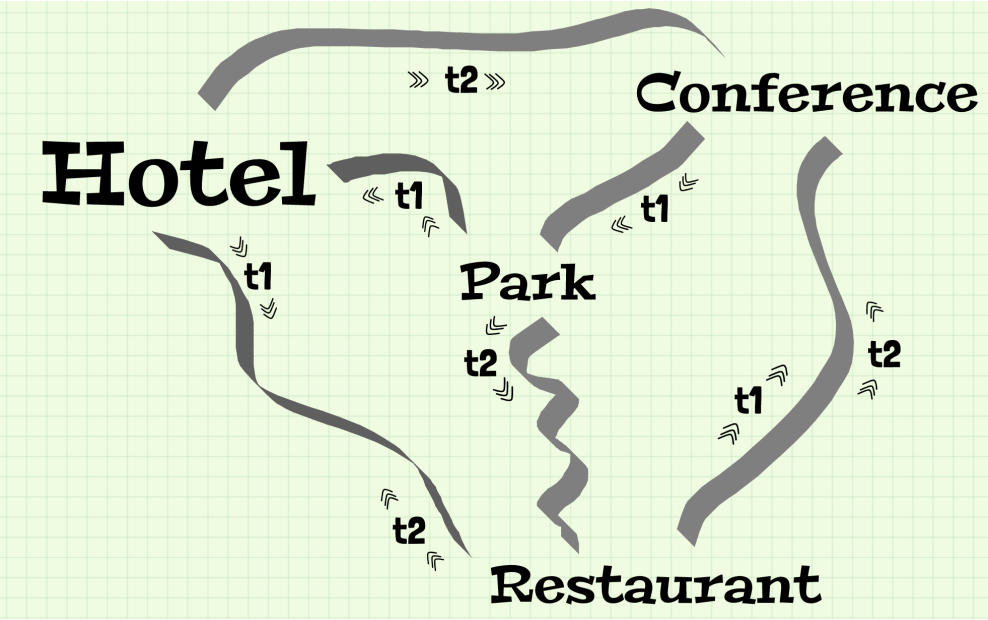


Figure 8.1: Repetition of Figure 2.1, with an example matching the lookup table setup.

correct response would involve first looking up `000 t1` and then applying `t2` to its outcome.[3] To encourage the model to interpret the individual parts of the inputs, the model is asked to output also the intermediate steps of the computation.

Because applying the lookup tables themselves requires nothing more than simple memorisation, the difficulty of the task resides solely in inferring the compositional structure from the input-output sequences. The task performance of the network is thus directly linked to the extent to which the solution a network inferred is compositional, without being conflated by other factors.[4] Liška et al. (2018) show that only very rarely a vanilla LSTM model converges to a solution that generalises to the test set, while the vast majority of trained networks does not exhibit compositional behaviour.

8.1.2 Data splits

Following Liška et al. (2018), we use eight randomly generated 3-bit atomic lookup tables and the 64 possible length-two compositions of these atomic tables. Some examples can be found in Figure 8.2. Contrary to Liška et al. (2018), we use several different splits, that vary in terms of how many and which types of sequences are held out.

[3]Note that we use *polish* notation, to facilitate an incremental computation of the outcome.

[4]At the same time, it also strips away other aspects of compositionality that may be relevant and interesting. For a more elaborate discussion of this issue, I refer back to Chapter 4.

	Input	Target	AG target
Atomic	`000 t1`	`000 011`	`0 1`
	`001 t2`	`001 000`	`0 1`
	...	...	
Composed	`001 t2 t1`	`001 000 011`	`0 1 2`
	`110 t2 t3`	`110 001 000`	`0 1 2`
	...	...	

Figure 8.2: Examples of 3-bit lookup tables and a length-two composition. The order of presentation follows *Polish* notation, allowing the encoder to process the input incrementally rather than having to wait until the very last input symbol. The AG target is a sequence of indices that represent to which symbols in the input sequence the decoder should attend.

Heldout inputs The easiest setting is the *heldout inputs* setting. In this setting, for each of the 64 compositions in the data set, two inputs are held out. E.g., for the composition `t3 t4`, the sequences `t3 t4 01` and `t3 t4 11` could be in the test set, while all the others are in the training data. This setting matches the one presented by Liška et al. (2018).

Heldout compositions In the *heldout composition* condition we hold out *compositions* of tables, rather than random inputs. Of the 64 potential compositions, we randomly remove eight.

Heldout tables The last condition is the *heldout tables* condition, in which we test to what extent models can generalise to compositions with tables that occurred atomically in the training set (e.g. `000 t7`), but never in combination with any other table. For this condition, we remove al composed sequences containing two randomly chosen lookup tables (that I will call `t7` and `t8`).

8.2 Model

For the present study, we use an encoder-decoder model (see Section 2.2.1) similar to the model I used in Chapter 3. Contrary to that model, the model used here has a *sequential* decoder; furthermore, it has an *attention mechanism* (Section 2.2.4).

8.2.1 Architecture

I included a graphical depiction of the model in Figure 8.3. The model first uses a recurrent layer to encode the input (left upper side of the figure). As explained in Section 2.2.2, it requires an *embedding* layer to transform the discrete

one-hot vectors that represent the input words (`000`, `t3` and `t1` in the figure) into continuous representations, which is left implicit in the picture.

The final encoder representation is then used to initialise the decoder part of the model (top right). At every decoding step, the model computes an attention weight vector, which indicates which encoder representations are the most relevant for the current generation step. This weight vector is used to generate a weighted average of these encoder representations, which is often called a *context vector*. This context vector is combined with another vector which represents the embedded previous output of the decoder and then given as an input to the decoder, which generates the next output. The first input token to the decoder is always the *start-of-sequence* token `<SOS>`.

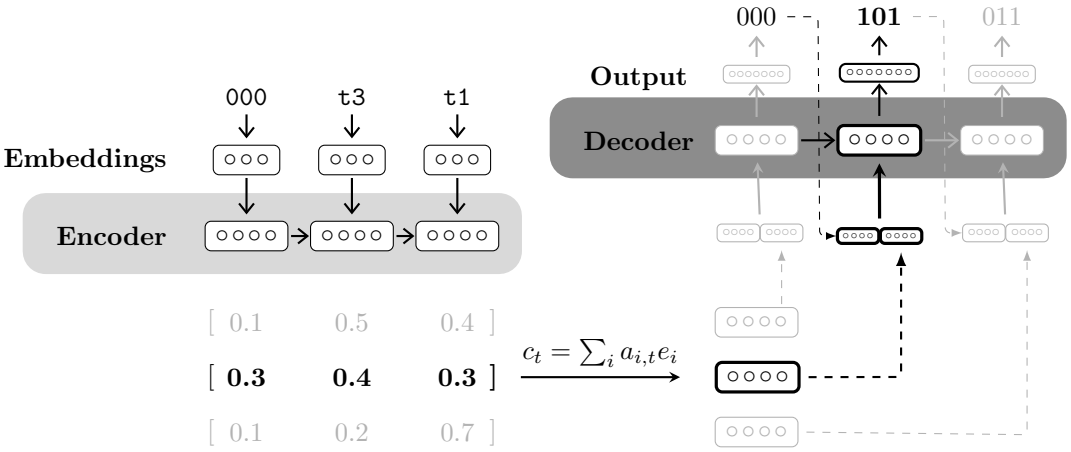


Figure 8.3: The model used for the lookup table task consists of a recurrent encoder and decoder and an attention mechanism. The encoder model first embeds the input words and then creates a series of representations $e_0, \ldots, e_n$ for the input sequence. The decoder, initialised with the final encoder state, then generates an output sequence, making use of an *attention mechanism*, with which it can look back at the encoder representations of every time step. At every time step, the decoder computes an attention weight vector, indicating which encoder representations are the most relevant for the current generation step. This weight vector is used to generate a context vector c_t, which is a weighted average of the encode representations e_i. This context vector is combined with the embedded vector that represents the previous output and given as input to the decoder which then generates the next output.

8.2.2 Attention mechanism

To compute the attention vector a_t, the model has an additional set of weights. These weights, which parametrise a two-layer perceptron are used to compare the current decoder hidden state with all encoder hidden states, resulting in a score

for every encoder hidden state. The attention vector a_t is formed by normalising these weights such that they sum to 1:

$$a_{i,t} = \frac{MLP(e_i, d_t)}{\sum_j a_{j,t}} \tag{8.1}$$

There are several possible methods to then integrate the context vector, constructed using the weights described above, with the embedded previous word. In our experiments, we try both concatenation and multiplication.

8.2.3 Learning

All parameters of the model, including the MLP that computes the attention vectors, are learned end to end by the model. As a loss function, we use categorical cross-entropy:

$$\mathcal{L} = \frac{1}{T}\left(\sum_{t=1}^{T}\sum_{i=1}^{N} - y_{i,t} \log \hat{y}_{i,t}\right) \tag{8.2}$$

where y_t is the target one-hot vector at time step t and $\hat{y}_t$ the softmax distribution generated by the model at time step t.

8.3 Attentive guidance

The learning signal described in the previous section provides information about the correct mapping between input and output but does not provide much information about what the nature of this mapping should be. A model might thus represent the mapping from inputs to outputs present in the training data in a way that does not generalise to the test data. For instance, reconsidering the toy world from Chapter 2 (see Figure 8.1), a model might memorise the composed path from hotel to conference to park as a whole and may then not be able to go to the park from the conference site if the latter was reached from the restaurant rather than the hotel. I now investigate if a model can be helped to robustly learn the desired solution by guiding its attention component.

Intuitively, we tell the model – during training – which components are to be considered at which point in time, similar to how we might also guide the attention of a human learner. Importantly, this signal is only provided during training, at inference time the model receives no different information than a vanilla model.

8.3.1 Attentive guidance as a loss

Concretely speaking, attentive guidance (AG) is provided to a model by adding an extra loss term to its objective function during training. This loss term represents

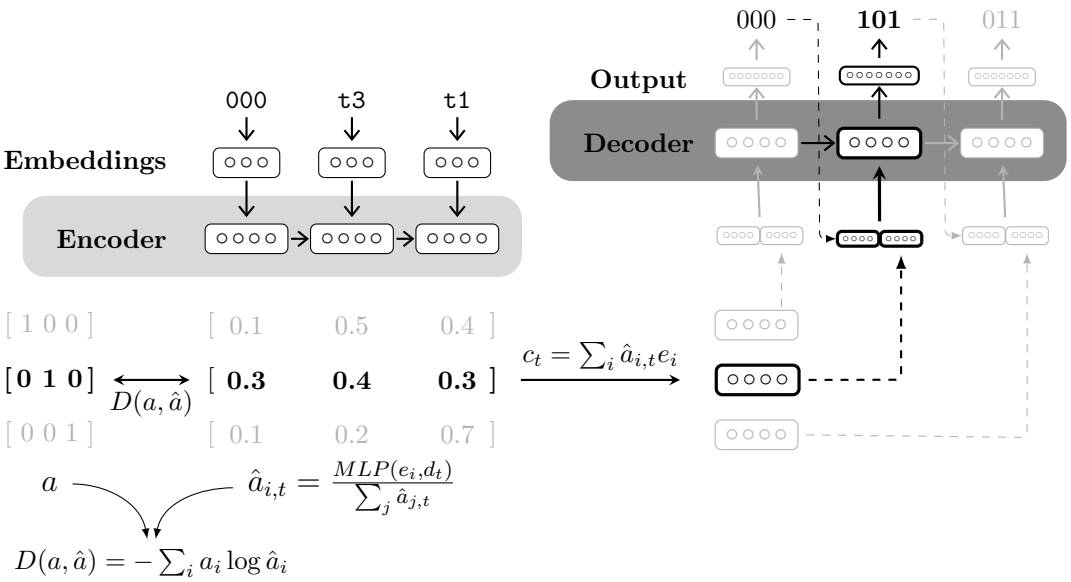


Figure 8.4: At each time step t, the model computes an alignment weight vector $\hat{a}_t$ based on the current decoder state h_t and all encoder states. During training, this alignment vector is compared with the target alignment vector a_t, which indicates which encoder state is relevant at the current computation step. E.g, at the second computation step, the model should apply the lookup table `t3`; it should therefore assign a high weight to the second encoder hidden state representing this lookup table, while it is not necessary to look at the *next* lookup table that should be applied. To teach the model this behaviour, we add a loss term to the training objective that expresses the cross-entropy between $\hat{a}$ and the AG target a.

the difference between the model's computed attention and the target attention pattern:

$$\mathcal{L}_{\text{AG}} = \frac{1}{T}\left(\sum_{t=1}^{T}\sum_{i=1}^{N} - a_{i,t} \log \hat{a}_{i,t}\right) \tag{8.3}$$

where T is the length of the target output sequence, N is the length of the input sequence, $\hat{a}_{i,t}$ is the decoder-computed attention of input token i at time t and $a_{i,t}$ is its corresponding attention target. The target attention pattern specifies how the input should be attended in a compositional way. We weigh the original loss of the model (Equation 8.1) and the AG loss with their corresponding weighting factors. A schematic of attentive guidance is given in Figure 8.4.[5]

[5]The code and data sets are available at: `https://github.com/i-machine-think/machine` and `https://github.com/i-machine-think/machine-tasks`, respectively.

8.4 Experiments

We train several encoder-decoder models with and without attentive guidance (I refer to them with the names *baseline* and *guided* models, respectively). The two model types are identical from an architectural perspective, but – as described in the previous section – the guided model has the additional training objective to minimise the cross-entropy loss between the calculated attention vectors and a provided target attention vectors in its backward pass.

For both guided and baseline models, we run experiments with different model sizes and try both GRU and LSTM cells. We find that GRU models generally perform better for the lookup-table task than LSTM models, for both guided and baseline models; in the remainder of this chapter, I only report results for GRU models.

8.4.1 Model parameters

We search through embedding and hidden layer sizes {16, 32, 64 and 128} and {32, 64, 128, 256 and 512}, respectively. We run each configuration 3 times for each condition (*baseline* and *guided*) and investigate the development of the accuracy of the resulting models on our three different test sets.

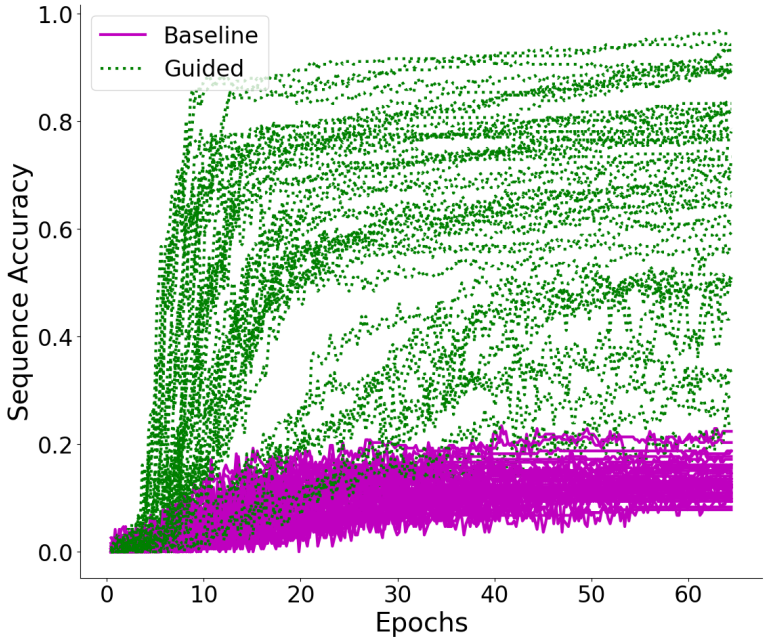


Figure 8.5: Accuracy on the *heldout* tables setup for all models in our grid search. The plot illustrates the across the board difference between the two models: even the worst models trained with attentive guidance (green dotted lines) generalise better than the best baseline models (magenta lines).

Our grid search confirms the findings of Liška et al. (2018) that vanilla recurrent models cannot solve the lookup table task, irrespective of recurrent unit and network size.[6] The baseline performance slightly increases with hidden layer size, but never reaches an average accuracy of higher than 30% across the heldout data, even though all models have a near-perfect performance on the training data. The guided models, on the other hand, appear to require a certain hidden layer size to generalise well, although even the smallest guided models outperform the larger baseline models. To illustrate this, I show the development of the accuracy for the *heldout tables* case for all baseline (magenta) and guided (green) models in Figure 8.5.

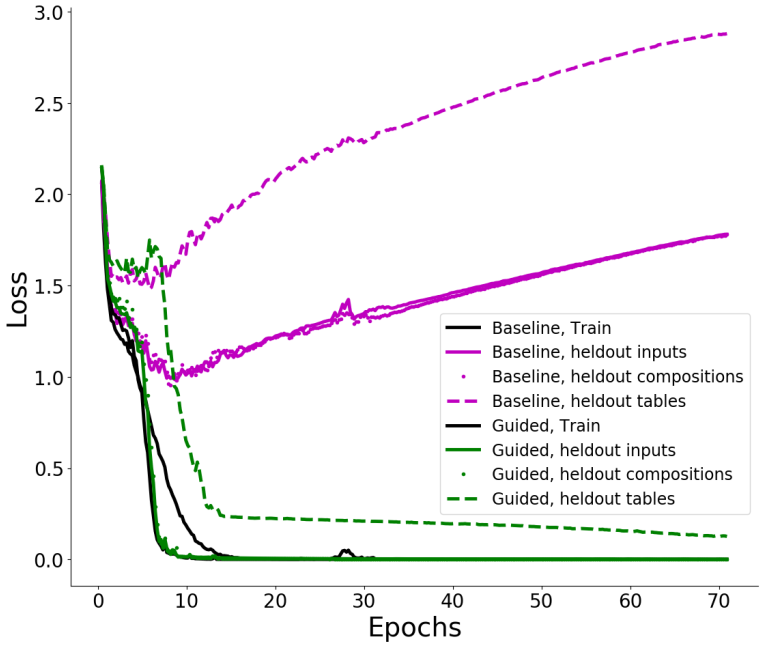


Figure 8.6: Typical task loss development for baseline and guided models on all different test sets (AG loss not depicted). Both the baseline and models trained with attentive guidance converge quite quickly on the training set (black lines). The baseline model strongly overfits the training data, as indicated by the increasing loss for the different test sets (magenta lines). The guided models, on the other hand, do not overfit: their loss on the test sets only decreases as the training progresses (green lines).

Furthermore, in Figure 8.6 we see that all baseline models exhibit overfitting: while their training loss decreases quickly, after an initial decrease, their test loss instead increases. The guided models, on the other hand, do not overfit at all, their test loss decreases steadily and never rises.

[6]Hupkes et al. (2019c) furthermore show that also the precise attention mechanism and whether the attention weight vector is computed before or after the recurrency in the decoder does not have an impact on the model performances.

8.4.2 Evaluation of best configurations

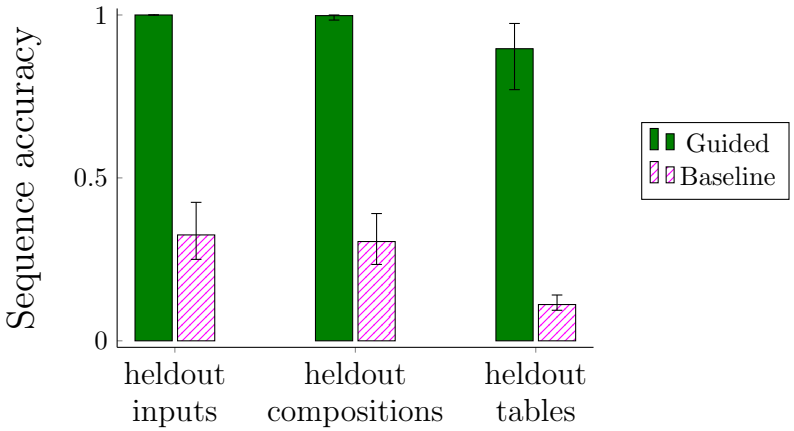


Figure 8.7: Average accuracies on all different test sets for chosen configurations for the lookup-table task. The guided models (green) strongly outperform the baseline models (magenta) in all different test conditions and exhibit a low variance across runs. The error bars indicate the performance of the worst and best-performing model on an individual test set.

For both the baseline and the guided models we pick the configuration that has the best performance across the test sets. We find the best baseline configuration to be a model with an embedding size of 128 and a hidden layer size of 512, whereas the best guided configuration has 16 and 512 nodes for embeddings and hidden layers, respectively. To evaluate the chosen configurations, we generate four new instances of training and testing data, generated with similar criteria, but with different instances. For each configuration and sample, we train two models, using Adam as optimiser (lr=0.001), resulting in 8 different runs for both baseline and guided models. The average accuracies can be found in Figure 8.7.

As expected, all baseline models fail to capture a compositional strategy and perform poorly across all test sets.[7] All guided models, however, achieve a near-perfect generalisation accuracy to the simplest case of generalisation and also generalise well to compositions that are not seen at all during training (*heldout compositions*). For the most difficult case, which requires using tables compositionally that are only seen atomically (*heldout tables*) the generalisation accuracy goes slightly down, and not all models converge to a perfect performance.

Overall, these results provide a clear demonstration that attentive guidance very effectively directs models to more compositional solutions.[8]

[7] It should be observed, however, that the baseline models still perform well above chance level, which lies at 0.2% for this task.

[8] Hupkes et al. (2019c) provide results also for another task, which I chose not to discuss here.

8.5 Compositionality in parameters

In the previous section, we saw that encoder-decoder models can be *guided* to find better generalising solutions for the lookup table task by providing them with a loss signal that indicates which parts of the input are relevant at which stage of processing. As the models trained with and without this signal differ only in terms of the information they received during training, they provide an interesting test case: architecturally the models are identical, but they implement different solutions to the same problem. If the guided models differ in some notable aspects from the baseline models, this could potentially be exploited also in situations where such a learning signal is not available – for instance because no annotation for the 'correct' attention pattern is available, or because it is unclear what this pattern should be.

In the remainder of this chapter, I present a series of explorative experiments in which I aim to understand if the difference in solution can be detected also from the parameter space or the activations of the models.[9] In these experiments, I bring back several of the techniques proposed or used in the previous chapters of this thesis, such as ablation (Chapter 6), visualisation (Chapter 3) and diagnostic classification (Chapter 5).

8.5.1 Weight heat maps

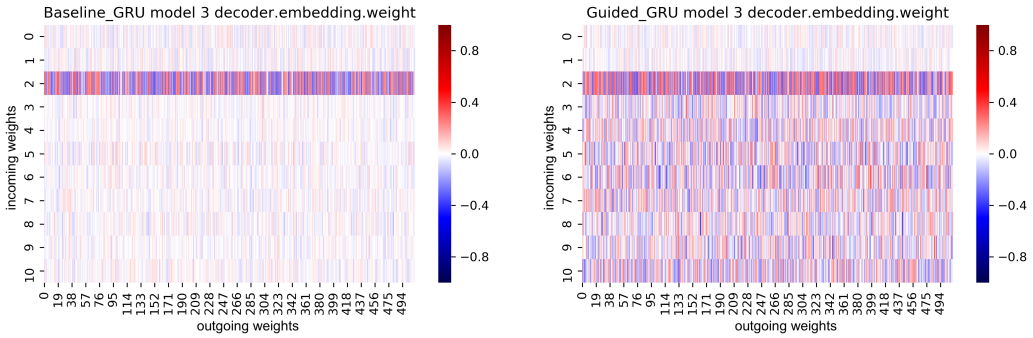
I first consider the learned parameters (or *weights*) of the network. In particular, we look at:

encoder embeddings	Used to embed the inputs to the model
decoder embeddings	Used to embed the *previous* output of the model
W^D_{hz} and W^D_{iz}	Weights from decoder hidden and input to *update* gate
W^D_{hr} and W^D_{ir}	Weights from decoder hidden and input to *reset* gate

We generate weight heat maps in which every cell (x, y) represents the forward connection from a unit y going to another unit x. For instance, Figure 8.8 depicts a visualisation of the decoder embeddings of the baseline and guided models. The 11 nodes on the y-axis represent the 11 output tokens of the model. The first three nodes represent the end of sequence, padding symbol and beginning of sequence symbol, respectively; the last eight the eight model outputs (`000`, `001`, etc). The 512 nodes on the x-axis represent the 512 dimensions of the embedded tokens. Each point (x,y) in this plot thus represents the connection from vocabulary token y to embedding unit x.[10]

[9] All results I report are averages over five models.

[10] Note that we do not normalise reported weights or activations by the activity of the 'pre-synaptic' neurons connected to it. This would be interesting to explore in future research, since a



(a) Baseline (b) Guided

Figure 8.8: Heat map of the decoder embedding weight values. Each row represents the connections of an input neuron (corresponding to one word) to the embedding layer of the decoder.

The decoder embedding visualisation also shows the most striking difference between the guided and baseline models. Generally, the baseline model exhibits small weights for the output tokens, whereas the weights of the guided models are both larger and have a larger variance. Row number two, which is equally strong for both networks, forms an exception. This row represents the embedding for the start of sequence token $<sos>$, which is used to initiate the decoder generation.

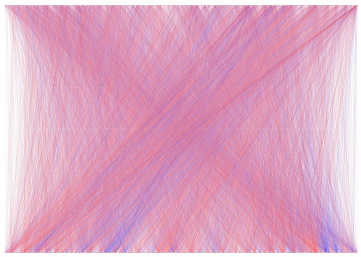
8.5.2 Connection maps

While the heat maps for lower dimensional layers are still somewhat interpretable, heat maps for connections between components of larger dimensionality do not provide much insight. To visualise the connections between such layers, we generate a plot in which we represent the neurons as nodes and the weights as edges; The thickness and colour of an edge represent the magnitude of the weight. We apply a threshold to remove edges that corresponded to weak weights. For this pruning step, we use a threshold of ± 0.2 and ± 0.17 respectively, which corresponds on average (between guided and baseline models) to keeping the strongest one per cent of the weights.

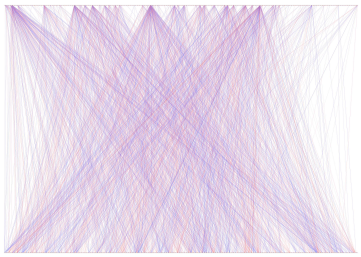
In Figure 8.9, I show a typical picture of the encoder update gate weights W_{hz} and W_{iz}. Every line going up represents a connection weight from a hidden or input unit (i.e. a unit of the layer representing the token embeddings) to the update gate of the encoder.

A striking difference between the baseline and guided models is the distribution of the connections. The connectivity of the baseline model is relatively evenly distributed over the different neurons. The guided model, on the other hand,

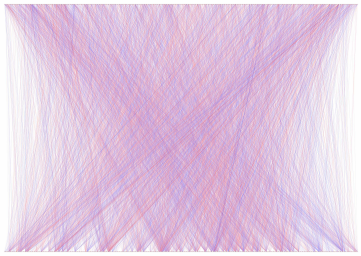
neuron's activation and the importance of its weight is in part dependent on the mean activation of its predecessors.



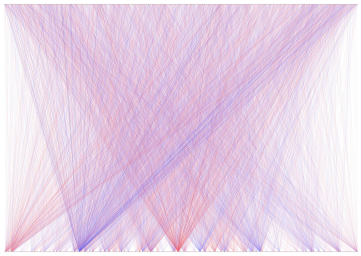
(a) Baseline, update gates W_{hz}.



(b) Guided, update gates W_{hz}.



(c) Baseline, Decoder update gates W_{iz}.



(d) Guided, Decoder update gates W_{iz}.

Figure 8.9: Visualisation of weight matrices W_{hz} (top) of the encoder and W_{iz} (bottom) of the decoder. Weights going from the previous to the next layer are represented by lines going from bottom to the top. The color reflects the weight value, where blue denotes negative, red positive and white zero. Best viewed in color.

has some neurons whose connectivity is strong, and others that do not have many connections. Such neurons that have many strong connections occur in the top layer of the encoder of the AG model in W_{hz} and W_{hr}. Similarly, strongly connected neurons can be found at the bottom layer of the AG's decoder in W_{iz} and W_{ir}. The same pattern holds for the recurrent W_{hh} weights. The finding of highly connected neurons suggests that guided models learn to specialise using fewer but more strongly connected neurons, which could help to learn a more modular solution.

Another interesting phenomenon is the apparent *polarity* of the strongly connected units in the guided models: there are a few distinct units whose connection weights are positive or negative, but not both. Interestingly, when considering the *update* gate weights of the same neurons (not shown), their polarity flips. Neurons that have very strongly positive weights to the update gate have

generally negative weights to the reset gate, and vice versa. A potential explanation for this is that when information from the current hidden state is to be retained, that same part is being reset in the previous hidden state.

8.6 Compositionality in activations

Following the hypothesis suggested by the parameter analysis of the model – that the guided models have more specialised groups of neurons that respond to specific stimuli – I now proceed with analysing the activations of the trained models.

8.6.1 Specialised neurons

First, I present a plot of the activation values of the baseline and guided models over time (Figure 8.10). To generate this plot, we randomly sample 50 neurons of both guided and baseline models and track their activation values emitted over time. Both in the encoder and decoder activations exhibit strong differences between baseline and guided models. Most striking are the encoder activations of the guided models, which confirm the existence of polar neurons that have only positive or only negative activations. In general, the encoder activations of the guided models produce activations in more specific value ranges than the baseline models, which I carefully take as a hint of potential specialisation. Also in the decoder of the guided models, a few polar neurons can be found, although most of the neurons uniformly cover the whole value range during processing.

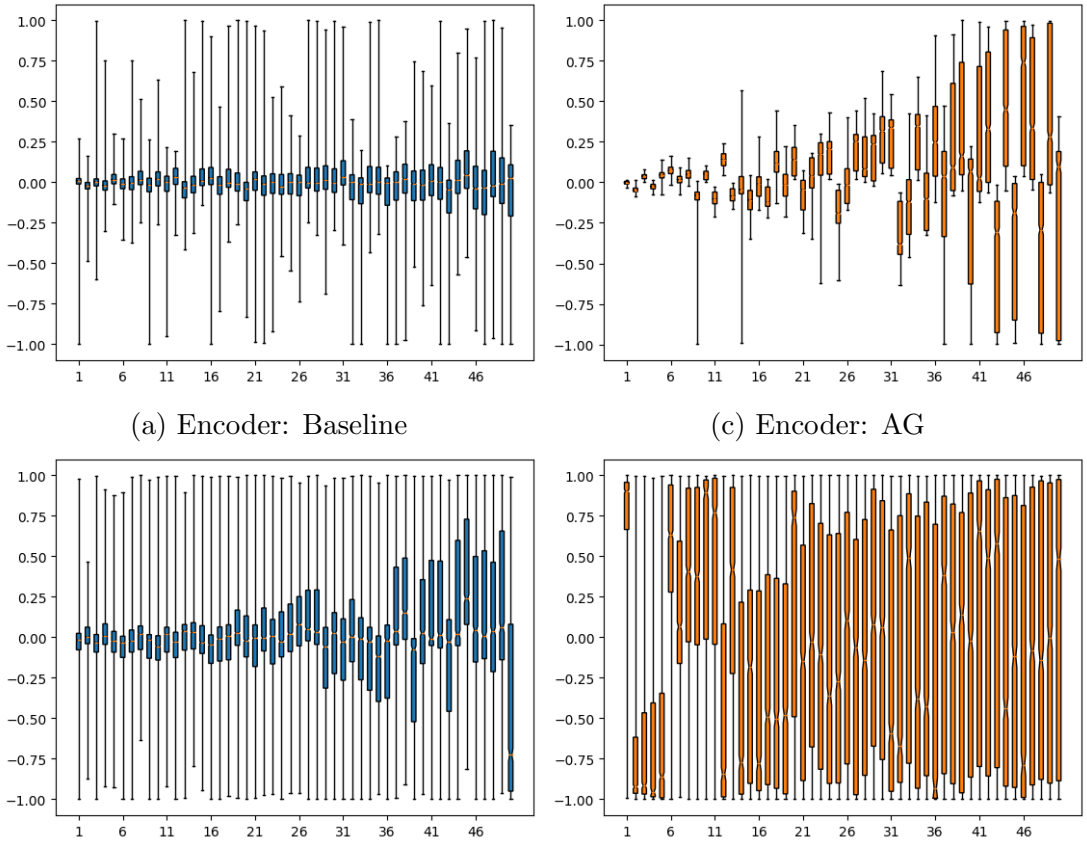
Encoder units

I now present a more target analysis of which encoder neurons are crucial for encoding the relevant lookup tables. To do so, we use diagnostic classification (see Chapter 3).

The encoder processes inputs that consist of sequences of lookup tables and the input to those lookup tables. We train diagnostic classifiers to predict the lookup table from the encoder's activations at the time step where this lookup table was just presented. For example, if the input was `000 t1 t2`, we train the classifier to predict `t1` from the encoder activations of the second time step and to predict `t2` for the activations of the third time step.

To find the units that are involved in encoding the tables, we then use a procedure similar to the method of Dalvi et al. (2019). Starting from an empty set, we keep adding units – taking always the unit with the highest absolute weight in the diagnostic classifier – until the set of added units together have an accuracy of 95% of the accuracy of all units.[11] I show the results in the first row of Table 8.1;

[11] Unlike Dalvi et al. (2019), we do not use any regularisation on the diagnostic classifier to contrast the different degrees to which information is distributed across neurons in the two



(a) Encoder: Baseline

(c) Encoder: AG

(b) Decoder: Baseline

(d) Decoder: AG

Figure 8.10: Distributions of activation values for 50 randomly sampled neurons for baseline (blue) and AG (orange) for both encoders (top) and decoders (bottom). Whiskers show the full range of the distribution. (Best viewed in colour)

All numbers are averaged over the five trained models.

For both baseline and guided models it is possible to predict the tables from the hidden activations with a high accuracy. However, there is a strong difference in the number of neurons required to do so (column three). The baseline models require on average 35 units to make a good prediction, the guided models need only two.

To verify whether the units in the detected groups (which I refer to with the term *functional groups*) are actually important units in the model, we consider the strengths of the connection weights of each of these units. On average, 93% of the units in the functional group of the AG models can be found in the top 5% of the units with the strongest absolute weight values. We conclude that the units of the functional group are highly connected and thus likely to play an essential role in the functionality of the model.

---

model types.

	Model	Full DC Accuracy	#Units	Group accuracy
h_t^{enc}	*Baseline*	0.98	35	0.93
	Guided	1.0	2	0.98
z_t^{dec}	*Baseline*	0.53	52	0.51
	Guided	1.0	22	0.96
r_t^{dec}	*Baseline*	0.52	44	0.50
	Guided	1.0	21	0.96

Table 8.1: Accuracy of the diagnostic classifiers trained to predict the most recent lookup table from the hidden activations (h_t^{enc}) of the encoder and the input and reset gate activations (z_t^{dec} and r_t^{dec}, respectively) of the decoder of both baseline and guided models. Column two shows the accuracy of the diagnostic classifier, column three indicates the number of neurons required to arrive at at least 95% f this accuracy, column four the accuracy of this reduced group of neurons.

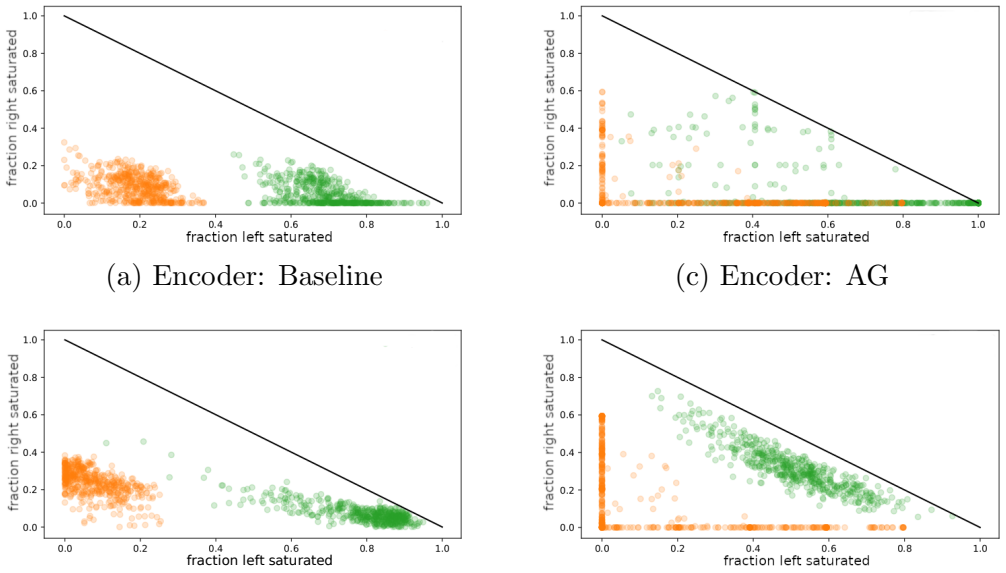
Decoder gates

By analysing the encoder activations, we saw that the lookup tables in the input could be reliably predicted both from the encoder activations of the baseline and guided models. Now, we focus on the decoder activations and test if the lookup tables can also be predicted from the decoder *gates*, which modulate the processing in the decoder. We use the same methodology as in the previous experiment: we train diagnostic classifiers on the update and reset gate of the decoder, respectively, and we then find the minimal number of gate units required to obtain 95% of the full diagnostic classifier accuracy. I show the results in the second and third rows of Table 8.1.

For the decoder gates, a difference between the guided and baseline models already arises for the full diagnostic classifier accuracies. The current lookup table can be perfectly predicted from both the update and reset gate of the guided models; For the baseline models, an accuracy of only around 50% is reached.[12] Also, compared to the encoder hidden layer, much larger functional groups are required to arrive at 95% of the maximal accuracy. Where the lookup table information in the encoder hidden layer of the guided models was inferable from on average just two neurons, more than 20 decoder gate units are needed to infer the same information. The information is thus more distributed over the gates.

This difference could be explained by the fact that the function of the gate differs from the function of the hidden layer. Whereas the hidden layer activations represent and store information, the gates instead use this represented information to modulate the flow of information through the network. Even information that is represented very locally may still have an impact on several other units, which

[12] Accuracy with a majority classifier for the task is 12.5%.



(a) Encoder: Baseline

(c) Encoder: AG

(b) Decoder: Baseline

(d) Decoder: AG

Figure 8.11: Gate activation plots for reset gate r_t and update gate z_t. Best viewed in colour.

would be reflected in a more distributed representation of this information in the gates. Additionally, a gate unit affects only a single hidden unit, while a hidden layer unit can potentially affect all gates in the upcoming time step, making distribution of information across the gates more likely.[13]

8.6.2 Gating behaviour

Both previous experiments point to differences in the gate usage between the guided and baseline models. As a last step in our study of the models' activations, we take a closer look at the gate activations by considering the visualisation method of Karpathy et al. (2015) that I also used in Chapter 3. This method considers, for each gate in the network, the fraction of samples for which it is *closed* (or *left-saturated*, defined as having an activation smaller than 0.1) or *open* (right-saturated,activation greater than 0.9).

Recall, as I explained in Chapter 2, that the reset gate r_t of a GRU model determines to what extent the next *candidate* hidden state depends on the previous hidden state (Equation 8.4). The update gate z_t decides the extrapolation factor

[13]In fact, if we do the same experiments with the *encoder gates*, we observe a very similar pattern. The lookup tables can not be reliably predicted from the baseline encoder gates, but can be from the guided model encoder gates; the minimal number of units required to approximate this accuracy for the guided models is around 20.

between the previous state and the candidate state (Equation 8.5):

$$\tilde{h}_t = \tanh(W x_t + r_t \odot U(h_{t-1}) + b) \tag{8.4}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{8.5}$$

The gate saturation plots indicate how a model uses its gates: Are they frequently open or closed or almost never? Do they have a preference for either one or the other? In our case, the plots (show in Figure 8.11) confirm the difference between the usage of gates of the baseline and guided models that we also found in the previous sections. The guided models appear to make more use of their gates: Reset gate activations generally stick to the axes, indicating that they are sometimes open (or most of the times, depending on the position on the axis) but never closed, or the other way around. The update gate units of the decoder instead are positioned on the diagonal, indicating that they are almost always either open or closed, and almost never anything in between.

The gate units of the baseline units do not show such a behaviour, but they do show a clear distinction between the update and reset gates values. The reset gate units are neither frequently open nor frequently closed, as indicated by their position in the left bottom corner of the plot. Their values are thus usually higher than 0.1 and lower than 0.9. Almost all update gate units are frequently open, indicating that the model does not optimally use the expressivity of the gates.

8.7 Ablation and substitution studies

In this section, I describe two experiments that involve ablation of components or groups of neurons to uncover which parts of the guided models are responsible for the improvement in solution. Are there specific components of the network that are particularly important for the solution? Can the neurons with strong connectivity that we observed in the previous sections perhaps solve the task also without the other neurons?

8.7.1 Component substitution

First, I focus on the contribution of the different parameter sets of the model. To learn about their impact, we extract the entire weight set from a guided (or baseline) model and put it into a trained baseline (or guided) model. Then, keeping the extracted component frozen, we retrain the rest of the parameters of the model with a low learning rate to adapt to the new component. We retrain guided models with a baseline component using the attentive guidance signal and baseline models with a guided component without.

If we retrain a guided model with a frozen baseline component, we expect the performance to drop if that component was important for a compositional solution,

as the network is unable to recover itself. Conversely, if a baseline model with a frozen component extracted from a guided model is retrained without guidance and performs better, that component might contain weights organised in such a way that it forces the baseline model to retrain itself in a more compositional manner.

Setup

We consider eight different components:

- The *entire* encoder and decoder;
- The encoder and encoder weights from input to hidden (W_{ih}^{E} and W_{ih}^{D});
- The encoder and decoder recurrent weights from hidden to hidden (W_{hh}^{E} and W_{hh}^{D});
- The encoder and decoder embedding weights (Emb_E and Emb_D).

For each component, we tune 16 models by retraining the remaining original parts for a maximum of 100 epochs, with a learning rate of 0.001 to allow for limited adjustment.

Results

None of the baseline models with a frozen guided component are able to retrain themselves such that their performance significantly increases. I report the sequence accuracies of the retrained *guided* models with frozen *baseline* components on the new compositions test set in Table 8.2.

For the AG models with frozen baseline components, the encoder embeddings seem irrelevant for a compositional solution: using baseline embeddings result in a similar score. The decoder embeddings, however, do seem to play a role. The retrained accuracy with baseline decoder embeddings is much lower than the score of the original guided model. This seems to be in line with the differences in heat maps shown earlier in Figure 8.8. Replacing the entire encoder results in a 80% drop in accuracy to 0.167.

Interestingly, the recurrent encoder hidden to hidden (W_{hh}) weight matrix can be replaced without as big a drop, and using the baseline input to hidden (W_{ih}) weights improves the accuracy. Finally, replacing the decoder's W_{ih} weights drops the accuracy to around 0.6, but doing the same for the decoder's W_{hh} weights again results in an unexpected increase to almost 0.9. This seems to indicate that the W_{ih} weights of the decoder play an important role in a compositional fit, as the model is unable to recover itself when using baseline decoder W_{ih} weights. The increase in performance after replacing either the encoder's W_{ih} or the decoder's W_{hh} implies that training with AG actually produces suboptimal

Model	Substituted Component	Accuracy
Guided	-	0.82 ± 0.12
Guided	Encoder	0.17 ± 0.11
Guided	Decoder	0.12 ± 0.05
Guided	Emb_E	0.75 ± 0.09
Guided	Emb_D	0.31 ± 0.07
Guided	W_{ih}^E	0.89 ± 0.05
Guided	W_{hh}^E	0.79 ± 0.12
Guided	W_{ih}^D	0.60 ± 0.08
Guided	W_{hh}^D	0.91 ± 0.03

Table 8.2: Sequence accuracy on the new compositions test set. Accuracy is averaged over three models and shown with standard deviation. All retrained models are *guided* models, as indicated in the first column. The second column indicates the baseline component that is taken from a baseline model and frozen during retraining. The third column indicates the accuracy after retraining.

weights for these components. Perhaps the use of a frozen baseline component in a model retrained with AG acts as some kind of regularisation and incentivises the remaining components of the model to become more compositional. Another explanation could be that the AG loss does not provide an appropriate signal for all components, and should thus not be back-propagated to all of them.

8.7.2 Neuron pruning

After showing in Section 8.6.1 that a few strongly connected neurons organised in functional groups carry out specific functions, we want to exhaust this observation and see if the model can still successfully solve the task by using **only** those strongly connected neurons. We remove all weakly connected neurons, keeping only 5% of neurons with the biggest weights of the encoder and decoder of the guided models, respectively. Averaged over all models, distilling the network in this way results in a performance drop to 12.4% sequence accuracy on the new compositions task.

We then retrain the network 20 epochs (in the way specified in Section 8.4), which fully restores the functionality and even yields better performance: 92.5% on average compared to the full network with 82.3%.

The loss in performance that occurs when neurons are removed indicates that some functionality is distributed among weakly connected neurons. However, the fact that their functionality can be taken over by other neurons shows that weakly connected neurons do not play a crucial role. We conclude that most of the neurons do not contribute to the solution and therefore only an extremely small subset of all neurons of the model suffices to solve the task after retraining. Those

neurons exhibit strong weights and are specialised in functional groups. Networks that find a compositional solution seem to rather form a small number of highly specialised neurons than distributing functionality over the whole network.

Independently, similar results were found also by other research groups. Frankle and Carbin (2019) present a study with feed-forward and convolutional networks in which they show that over 90% of a trained model's neurons can be pruned without compromising the accuracy of the model. Frankle and Carbin (2019) argue that these successful sub-networks received a lucky *lottery ticket* in the initialisation procedure, which is not easily found when a network is trained with a small number of neurons from the start.

In a different domain, Voita et al. (2019) show that in transformer models trained to perform machine translation, the vast majority of attention heads can be removed without seriously affecting the model's performance. Their and our results illustrate that while the current initialisation and optimisation techniques have resulted in a gigantic leap forward when it come to the applicability of neural network, there is still much room for improvement.

8.8 Conclusion

In the first five studies I presented in this dissertation, I focused on the analysis of already trained models. In this last chapter, I took a slightly different approach: I tested whether I could interact with the network through its learning signal, to *change* the type of solutions found by the network. Afterwards, I compared the different solutions using many of the techniques discussed in the earlier chapters.

This study started from the assumption that one of the main reasons that a model's solutions sometimes deviates from the desired solution is that the model does not correctly segment the input. If a model memorises a part as a whole that was in fact made up from different parts, this may result in a failure to generalise to a different sequence which is made up from these parts. I tried to address this issue with *Attentive Guidance*, an additional learning signal that – through the attention component of a model – indicates to the model what the salient subsequences are.

First, I showed that this learning signal indeed helps the model, resulting in across the board accuracy increases on the *lookup table task* (Liška et al., 2018), which was specifically designed to test for compositional solutions. This confirms the hypothesis that one of the reasons models do not always generalise correctly stems from the fact that they assume a wrong segmentation of the input. Barrett et al. (2018) showed that attentive guidance can also be successful in scenarios with natural data. Using attentive guidance with a signal coming from human attention estimated from eye-tracking corpora, they obtained a substantial improvement for a number of natural language processing tasks, such as sentiment analysis and grammatical error detection. Further research could focus on identifying other

scenarios in which human attention or other types of human data might aid neural models in finding generalising solutions.

Then, I compared the models trained with and without attentive guidance, to understand if the difference in solution was detectable from the parameter space and neuron behaviour of the trained models. With several experiments – using, among other things, diagnostic classification and ablation – we found that the guided models develop specialised neurons organised in *functional groups*. No such groups were found in the baseline models. Furthermore, the guided models develop *polar* neurons, and they appear to make better use of their gates. Future research could focus on exploiting these findings about modularity and specialisation of neurons to investigate whether similar solutions can be achieved without the explicit use of attentive guidance.[14]

[14]See, e.g. Korrel et al. (2019).

Chapter 9

Discussion and conclusions

The main research goal I formulated at the beginning of this dissertation was to understand if neural networks can be used as explanatory models of natural language processing. On page 2, I described three generic requirements for explanatory models of human processing:

i) **Architectural similarity**: The model should share some relevant aspects of processing with humans;

ii) **Behavioural similarity**: The model should be able to adequately approximate the phenomena we are interested in understanding;

iii) **Model interpretability** We should be able to obtain insight into *how* it implements or processes these phenomena.

In the seven chapters that followed, I argued that recurrent neural networks share some relevant aspects with the human processing system, making them satisfy requirement i), and I investigated if they also satisfy requirement ii) and iii). I looked at processing of different types of hierarchical structure and compositionality – in linguistics regarded important aspects of natural language – considering both *artificial languages* (Chapter 3 and 4) and *natural language* (Chapter 5 through 7). I explored a range of techniques to understand the internal dynamics of neural networks, such as *diagnostic classification* (Chapter 3 and 5), *neuron ablation* (Chapter 6 and 8) and *generalised contextual decomposition* (Chapter 7). I considered how the networks behaviour can be influenced at both training and inference time (Chapter 8 and 5, respectively).

In this last chapter of this dissertation, I recapitulate what I have done in these studies, starting from the three requirements listed above (Section 9.1, 9.2 and 9.3). I also briefly discuss the potential impact of interpretability research on society (Section 9.4) and sketch some potential paths for future research (Section 9.5).

9.1 RNNs as abstractions of human processing

One function of an explanatory model is to help understand something for which a full explanation is not known or available. For an explanatory model to serve that function, it should share some relevant properties with this thing or phenomenon it is a model of. Which properties are relevant depends on the particular aspects that the researcher aims to understand. For instance, to help understanding gravity it may be useful to consider a model in which friction does not exist, but to understand the terminal velocity of objects when you throw them off a high building, including friction is important.

In this dissertation, I chose to primarily focus on *recurrent neural networks*. One reason for this decision was simply that many of the studies I described preceded the invention of *attention-based* transformer-like models (Vaswani et al., 2017) that currently dominate the state of the art in NLP (e.g. Radford et al., 2019; Devlin et al., 2019). However, recurrent models also share some properties with the human language processing system that attention-based models do not. In particular, they are temporally structured and receive their input in an incremental fashion. As such, they might provide insight in the potential mechanics that allow humans to process natural language – incrementally, with a system of interconnected units. It is therefore my conviction that to serve as explanatory models of language processing, recurrent models are more promising candidates than attention-based models.

While attention-based models may not be directly suitable to understand questions about human processing[1], this does not imply they cannot be used as explanatory models of language at all. On the contrary: their successes make them interesting models that can be used to discover more about the (static) structure of language. Many of the experiments described in this dissertation could also be conducted with attention-based models, which would provide a different kind of insight in the nature of the symbolic structure of language. Some of the experiments I described have already been applied to them. For instance, Goldberg (2019) and Wolf (2019) considered how well such models process subject-verb agreement, using tests similar to the ones described in Chapter 5 and Chapter 6.

9.2 Can RNNs represent interesting structure?

The second criterion I formulated concerned a model's ability to represent or approximate the phenomenon that we are interested in. I focused specifically on hierarchical structure and compositionality. Discovering what types of structure RNNs can correctly process under different circumstances has been a central theme in almost all of the chapters presented in this dissertation. These chapters

[1] A potential exception to this may be the recent GTP-2 model (Radford et al., 2019), which – since it is a language model – also receives its input in an incremental fashion.

all show that while models may not always find the solutions we expected or intended, RNNs can certainly capture interesting aspects of hierarchical structure and compositionality. I now briefly recap those findings.

9.2.1 Artificial languages

In Chapter 3 and Chapter 4, I considered artificial languages. Artificial languages provide a clean setup to study the abilities of RNNs to process structure, which allows to isolate specific aspects and facilitates evaluation.

Arithmetic languages In Chapter 3, I experimented with the *arithmetic language*. This language contains sentences that are spelled out arithmetic expressions with the operators `plus` and `minus`, whose meaning is defined by the outcome of the arithmetic expression. Due to the clear symbolic structure of this language, I could formulate hypotheses about how the network may process the sentences.

PCFG SET In Chapter 4, I used the artificial language *PCFG SET* to tease apart different aspects of processing structure, with a particular focus on what it actually means for a network to behave *compositionally*. In this chapter, I laid out five different tests, that considered different aspects and implications of compositional processing. While these tests provided interesting results about RNNs, the development and contextualisation of the tests constituted the most important contribution of this chapter. As an exception, I therefore "tested the tests" not only on RNNs, but also on convolution and attention-based architectures.

Results Both studies showed that recurrent models can learn to correctly predict the meanings of languages with a strongly hierarchical compositional structure. Interestingly, both studies also showed that if a model correctly learns to process structured sentences, this does not necessarily imply that they also processed the structures as intended. For instance, the models trained to predict the meaning of arithmetic expressions did not do so by following a linearised recursive strategy. Instead, they take a cumulative shortcut in which they count the number of embedded minus signs, which does not require keeping track of an elaborate stack.

The study with PCFG SET further illustrated that correct predictions do not entail that the model comes to these predictions by systematically recombining the parts defined by the researcher; they do not entail that the model can deal with something like synonym substitutions, and they do not entail that the *rules* a model learns match the general rules from which the data was generated. The first finding is further strengthened in Chapter 8, in which I show that providing a feedback signal to the model that helps it identify salient subsequences helps the model to correctly process temporally structured inputs.

The deviation of the conclusions drawn from behavioural experiments (what does the model do) and internal inspection (how does the model implement this) sketches an interesting contrast. They show that a model can behave like it understands hierarchical structure but yet not understand some of the components that many assume to be implied by this understanding. This discrepancy raises an important question: what does it mean to correctly process structure? Should the behavioural tests be changed to better match what we believe correct processing should entail? Or should we revise the components we thought were essential?

9.2.2 Natural language

In the second part of this dissertation, in Chapter 5, 6 and 7, I considered language models, trained on natural data (a corpus with English sentences). Following Linzen et al. (2016), I probed their ability to process hierarchical structure by considering their predictions on sentences with long-distance subject-verb relationships. These studies all confirmed results from earlier studies that recurrent models are able to model these relationships well, even if the subject and the verb are separated by quite some intervening sentential material, including multiple *attractor* nouns.

Focusing on one particular pre-trained LSTM model, the studies also uncovered different aspects of the mechanisms that neural language models use to model these relationships. Chapter 5 revealed that the model has two distinct types of number representations: a *shallow* representation – used for short distance agreement, and a *deep* representation that remains more constant when a long-distance dependency is processed. Furthermore, by splitting the corpus based on whether the model correctly or incorrectly processed the long-distance relationship, I showed that incorrect sentences typically go wrong already when the number of the subject is encoded.

Chapter 6, with a more controlled data set, confirmed and further uncovered the two distinct mechanisms that the model uses to process number information. The study described in this chapter showed that long-distance number information is primarily processed by a surprisingly local mechanism, consisting of just two units that are responsible for storing long-distance number information (plural and singular, respectively).[2] Short distance number information, as well as syntactic information, is represented in a more distributed fashion.

Chapter 7, exposes another interesting aspect of how the model processes subject-verb relationships: it uses its intercept terms (also known as bias terms) to perform default reasoning. The prediction of plural verbs is strongly causally linked to the grammatical subject of the sentence, while singular verbs do not require explicit evidence from singular nouns.

[2]The locality of this mechanism implies that the model cannot correctly process subject-verb agreement in specific types of doubly embedded structures. We are currently investigating this, both in models and in humans.

9.3 How can we interpret neural networks?

The last requirement I formulated concerned the *interpretability* of explanatory models. To use a model as explanatory model, it is important that we at least to some extent understand *how* it implements the things we are interested in. A model being able to perform subject-verb agreement on par with humans provides only very limited explanation if we cannot then use this model to identify the mechanisms by which it does so. Without interpretability, such results at best teach us that recurrent models are adequate approximators of the phenomena we are interested in, but even that is difficult to establish if the model is treated as a black box.

In the previous section I already briefly summarised some of the mechanisms that we discovered. Arriving at these findings required the development and use of different interpretability techniques, which I reiterate now.

9.3.1 Diagnostic classification

One of the most prominent techniques proposed and used in this dissertation is *diagnostic classification*. Diagnostic classification can be used to test what kind of information a model computes or represents. Its key idea is to train a *meta* model to predict some hypothesised information from the internal states of the model. For instance, one could predict part-of-speech tags from the internal states of a trained language model. If the meta-model can predict the hypothesised information from the original model's internal states with a high accuracy, this indicates that the hypothesised information is indeed represented in these states, whereas a low accuracy indicates the opposite.

Applicability Due to its simplicity, diagnostic classification is a powerful and versatile tool, which can be used for any type of neural model. It is by now part of the standard toolkit of interpretability research.[3] It has by now been used in many different scenarios, by both me and others. For instance, we used it to investigate how artificial dialogue agents process disfluencies (Hupkes et al., 2018a); how neural language models process negative polarity items (Jumelet and Hupkes, 2018); to what extent different type of sequential model build incremental representations (Ulmer et al., 2019); and what information is encoded in the messages of languages emerged in a referential game (Hupkes et al., 2019b). Others have used diagnostic classification to investigate the internal representations of BERT (e.g. Lin et al., 2019; Tenney et al., 2019a; van Aken et al., 2019); to analyse what type of features neural machine translation models represent (e.g. Belinkov et al., 2017b, 2019),

[3]When we proposed diagnostic classification in 2016 (Veldhoen et al., 2016), independently, also other researchers groups proposed to use similar techniques (Adi et al., 2017; Belinkov et al., 2017a; Gelderloos and Chrupała, 2016; Shi et al., 2016; Ettinger et al., 2016). Diagnostic classifiers are also known by the name *probing classifiers*.

and to find syntax trees in pre-trained word representations (Hewitt and Manning, 2019).

Limitations and difficulties While being a versatile tool to, among other things, investigate a model's internal states, diagnostic classification has also several limitations. Firstly, it requires an hypothesis about what the model might be representing, such as the strategy it might be pursuing to compute the meaning of a sentence in the arithmetic language. This hypothesis-driven nature implies that the technique can only be used to confirm or reject hypotheses that the modeller came up with himself and is not applicable for more open-ended explorations.

Furthermore, not all types of hypotheses can be tested with diagnostic classification. If a complicated (non-linear) meta-model is required to extract the hypothesised information from the internal states of the model – for instance because the model may not use this information in a linear way – it is difficult to assess whether this information was in fact represented by the model or was computed afterwards by the meta-model (a point also made by Saphra and Lopez, 2019b). This issue may be addressed by *comparing* different hypotheses, as done in Chapter 3, rather than focusing on the accuracy of a single diagnostic classifier.[4]

A related issue concerns the extent to which the information extracted by the diagnostic classifier is in fact used by the model. Even for very simple diagnostic classifiers, it may be difficult to assess if there is a causal relationship between the extracted information and the model's behaviour. Intervention studies (Chapter 5) can be a powerful tool to confirm that the information extracted by the diagnostic classifier is in fact causally related to the behaviour of the model. In some cases, if a diagnostic classifier indicates that a set of specific neuron together encode a particular feature, ablation studies may help to confirm such a result (see Chapter 5).

In summary, diagnostic classification is a useful tool when specific hypotheses are available. It may provide different kinds of information concerning when and where information is represented, and whether this representation is sparse or distributed. The results of diagnostic classification studies can be confirmed with a follow-up study that uses a different type of interpretability technique.

9.3.2 Ablation

In Chapter 6 and 8, I described several experiments that used neuron ablation. The idea behind it is simple: turn off the activations of a particular neuron or group of neurons and evaluate what is the impact on the model's behaviour.

[4]Hewitt and Liang (2019) recently proposed to use *control tasks*, that can also be used to calibrate the accuracy of a diagnostic classifier.

Applicability and limitations Neuron ablation can be an effective tool to uncover mechanisms that are encoded in single neurons. It does not require a specific hypothesis, but, as illustrated in Chapter 6, it does require a carefully designed data set to understand what the effects are of ablating a neuron or group of neurons. Given their combinatorial complexity, they are not suitable to find groups of neurons that together have an impact on the model's behaviour. However, they can be used to *confirm* the role of groups of neurons found with other experiments, such as diagnostic classification experiments.

9.3.3 Generalised Contextual Decomposition

The last technique I discussed in this dissertation is *Generalised Contextual Decomposition* (GCD), a generalisation of a technique proposed by Murdoch et al. (2018). GCD can be used to track the contributions of different input tokens or model components to the behaviour of the network and to study the *interactions* between different components and input words.

Applicability An important advantage of GCD is that it does not rely on a model's behaviour, which makes it more applicable in situations where neither diagnostic classification nor ablation studies are of help. For instance, GCD can be used to consider pronoun resolution in sentences such as *The boy told his friend that he liked her* (see Jumelet et al., 2019). For such sentences, it is not possible to determine if the model correctly resolves the pronouns by looking at its word probabilities: what matters is not the probability mass of the words *his* and *her*, but why the model assigned this probability mass to these words. GCD tracks exactly that, and it can thus be used quite generically to understand the causal relationships between the input and the model's behaviour.

Limitations GCD is computationally more complex than diagnostic classification, but much lighter than neuron ablation. It is designed to give a faithful explanation of how information flows through the model. However, to come to this explanation, several non-trivial choices have to be made. The contributions computed with GCD are not exact but an approximation, and it is unclear how much the approximation technique impacts the results.

Furthermore, to define how the information from a particular word or component flows forward, GCD requires to select which interactions with other words and components should be considered. GCD allows to specify this *interaction set* per experiment, but it is not always evident which interaction set best serves the current research question.

Lastly, while GCD is computationally relatively efficient once the interactions are decided and the approximations implemented, the technique does not easily extend to different types of models. Formulating GCD for a GRU model would

still be relatively straightforward, as it only requires adapting the interaction sets to the gates and components of the GRU; defining GCD for transformer-based models, which have many more multiplicative interactions than recurrent models, would be substantially more difficult.

9.4 The societal impact of interpretability

With the increasing ubiquitousness of neural networks in applications used in the public domain, their interpretability has become a topic of increasing importance also for society. Neural networks are used in many situations where it is important to understand what they implement or how they reason, and whether they may unintendedly encode unwanted biases. While societal impact was not my initial motivation to develop interpretability techniques, the topic came up several times during my studies. Most prominently, it arose when we developed generalised contextual decomposition.

In Chapter 7, I reported the results of a study in which we used GCD to analyse how a recurrent model processes subject-verb relationships. In the same project, we also did another study, on which I did not report in Chapter 7 in the interest of brevity. I would like to briefly discuss the findings study here, as an example of how interpretability research can contribute to society.

9.4.1 Pronoun resolution

In our experiments, we considered intra-sentential anaphora resolution, in which a pronoun within a subordinate clause refers to an entity in the main clause. For example: *The boy liked the **girl** because **she** was always nice to him.* In all sentences we used, the pronouns should be resolved based on *gender* information.

For sentences like this, it is not easy to find a setup where there is a right prediction that indicates whether the model correctly or incorrectly identifies the referent of the pronouns. With GCD, however, this can be easily established, by computing to what extent the probability mass of the words ***she*** and *him* are caused by the words ***girl*** and *boy*, respectively.

9.4.2 Corpus

We apply GCD to a corpus with sentences generated using the WinoBias corpus templates created by Zhao et al. (2018). The WinoBias corpus contains sentences with two nouns in the main clause, that are referred back to by two pronouns in a relative clause. As nouns, the WinoBias corpus uses job titles that are gender-neutral, but contain a stereotypical bias towards one gender (doctors and CEOs are *male*, nurses and housekeepers *female*); to not conflate different types

of biases, we replace these titles by entity descriptions that are unambiguously gendered (*king*, *bride*, *father*, etc.).

For every sentence, we create 4 different conditions, based on the gender of the subject and object (FF, FM, MF, and MM). For instance, an example of an MF sentence is: *The father likes the woman because he/she*. We sample 500 sentences per condition.

9.4.3 Default reasoning for gender

For all sentences, we compute and compare the contributions of the male and female nouns to the male and female pronouns. The results show that, like with the singular/plural case, the model uses a default strategy to deal with pronoun resolution. While the predictions of *he* primarily stem from the male nouns and the predictions of *she* are causally related to the female noun, the female connection is much stronger than the male connection. In other words, the model does not need much evidence to predict a male pronoun, but it does require evidence to predict a female pronoun. GCD reveals also the reason for this asymmetry, which lies in the decoder intercepts of the model. These intercepts encode a strong male preference, allowing the model to use this male prediction as a default, similar to how singular verbs are predicted as a default baseline for number prediction.

9.4.4 Biases in society

It is easy imagine scenarios in which asymmetrical treatment of gender in models used in society might prove problematic. Consider for instance the case in which a model is used to automatically match jobs with resumes; An a priori imbalance between male and female applicants would be highly undesirable.

Whether or not the solution of the model makes sense given how and on what data it has been trained, the difference in treatment for male and female referents was certainly not intended. The first step towards improving such issues requires understanding what they are. With this study, we made a first step to do so. Secondly, before we can start to improve such undesired biases, we have to understand what in the model is responsible for them. In case of this particular model, the bias seems to be encoded in the decoder intercepts, which might be easily addressed. Much more research is required to understand how to create models that are not unwantedly biased towards or against particular groups of society, interpretability research will play an important role in this quest.

9.5 What's next?

I have now briefly summarised the research described in this dissertation, mentioned some related work and touched upon the societal implications of interpretability

research. With this work, I believe to have confirmed that RNNs are in fact interesting to consider as explanatory models of human processing. For me, that means that this dissertation should not be seen as an endpoint, but rather as a starting point for many interesting studies to come. Throughout the previous sections, I already mentioned a few directions for future work, such as using interpretability techniques to understand and then remove unwanted biases in models. In this very last section, I would like to sketch two more paths I consider promising for the future.

9.5.1 Improving interpretability techniques

One important component of this dissertation constituted the development and evaluation of different types of interpretability techniques to further our understanding of neural networks. While I believe that I and the field have made substantial progress in this respect, there are still many open questions regarding the interpretability of neural networks. Such questions concern the faithfulness of techniques – to what extent does the information they extract truly reflect the underlying model behaviour; their computational complexity – how feasible is it to apply techniques to larger corpora and different models; and their informativeness – how useful is the information that they give us?

The different techniques I discussed in this dissertation score differently on these different axes. Diagnostic classification is easily applicable and computationally very feasible but requires clear hypotheses. Furthermore, it is not always easy to establish that the diagnostic classifier extracts information that was also used by the model. Neuron ablation does not have the latter problem but is generally not applicable for processes that are encoded more distributedly than in just a single neuron. GCD is both computationally feasible and directly related to the underlying model, but also illustrates how many difficult choices and simplifications need to be made to be able to track how information flows from particular input words. The impact of some of these choices – such as the choice of interactions to include and the influence of the choice of Shapley approximation technique – is difficult to estimate, making it hard to determine the faithfulness of the results.

There are also many interpretability techniques that I did not use or discuss in this thesis, such as layer-wise relevance propagation (LRP, Arras et al., 2017; Bach et al., 2015) and singular vector canonical correlation analysis (SVCCA, Saphra and Lopez, 2019b). While there is some work that systematically compares the results and types of conclusions that can be drawn from some of these techniques (e.g. Murdoch et al., 2018), much work in this area is still to be done.

9.5.2 Actually using RNNs as explanatory models of language processing

One of the main goals I described in the beginning of this dissertation was to understand if RNNs can be used as explanatory models of human language processing. I what followed, I explored if the three requirements I formulated for this to be possible were fulfilled. These studies led me to believe the are.

The work that I did as well as several other recent studies illustrate that RNNs may not represent all we find interesting about structure, but that they do capture many interesting aspects of it. Their performance on end-to-end tasks such as language modelling further shows that they are perhaps not perfect, but definitely adequate models of natural language. These positive results make them interesting alternative models for how humans process natural language. The interpretability techniques developed by me and others in the recent past allow us to investigate *how* RNNs process these structural aspects and get sometimes even very precise descriptions of the mechanisms that they use to do so. These findings can provide us with new hypotheses for how language is processed by humans.

Psycholinguistic experiments

One way in which such hypotheses could be tested is by doing psycholinguistic experiments. For instance, I described in Chapter 6 how LSTM language models use an extremely local mechanism to represent long-distance number information, while short-distance number information is represented in a more distributed fashion. This finding generates very specific predictions about the type of contexts in which interference of these two mechanisms might lead to mistakes, which can then be tested on humans with behavioural experiments.

Similarly, our GCD study pointed out very specific default behaviour in a pre-trained RNN model. How does this relate to known biases in the human processing system? And can we perhaps restart using RNN models to help explain their origins?[5]

Brain data

Another domain in which RNNs (as well as other types of neural models) may serve as hypothesis generators is in the analysis of brain data. The first steps towards the cross-pollination between modern computational linguistics and neuroscience stem from already more than 10 years ago, when (Mitchell et al., 2008) presented a study in which he used word representations from distributional semantics to predict brain activations. More recently, several other groups investigated the relation between representations from neural networks and brain activation data

[5]Concerning the relationship between connectionism and the existence of unbounded recursion in natural language, I personally very much like the account of Christiansen and Chater (1999).

(i.a Anderson et al., 2016; Huth et al., 2016; Wehbe et al., 2014; Abnar et al., 2018; Jain and Huth, 2018), both on the word and sentence level. Many of these studies focus on *if* there are potential links between model activations and brain data or evaluate which type of model best predicts different types of brain data. A complementary approach would be to instead use neural models to generate new hypotheses about how the human brain may implement specific features and test if similar mechanisms can be identified also in brain data. The results presented in this dissertation lay the groundwork to further develop such a line of research.

Bibliography

Samira Abnar, Rasyan Ahmed, Max Mijnheer, and Willem Zuidema. Experiential, distributional and dependency-based word embeddings have complementary roles in decoding brain activity. In *Proceedings of the 8th Workshop on Cognitive Modeling and Computational Linguistics (CMCL 2018)*, pages 57–66, 2018.

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

Andrew James Anderson, Jeffrey R Binder, Leonardo Fernandino, Colin J Humphries, Lisa L Conant, Mario Aguilar, Xixi Wang, Donias Doko, and Rajeev DS Raizada. Predicting neural activity patterns associated with sentences using a neurobiologically motivated model of semantic representation. *Cerebral Cortex*, 27(9):4379–4395, 2016.

Jacob Andreas. Measuring compositionality in representation learning. In *International Conference on Learning Representations (ICLR)*, 2019.

Dana Angluin and Carl H Smith. Inductive inference: Theory and methods. *ACM Computing Surveys (CSUR)*, 15(3):237–269, 1983.

Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. Explaining recurrent neural network predictions in sentiment analysis. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis (WASSA@EMNLP 2017)*, pages 159–168. Association for Computational Linguistics, 2017.

Joris Baan, Jana Leible, Mitja Nikolaus, David Rau, Dennis Ulmer, Tim Baumgärtner, Dieuwke Hupkes, and Elia Bruni. On the realization of compositionality in neural networks. In *Proceedings of the 2019 ACL Workshop*

BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, pages 127–137, Florence, Italy, 2019. ACL.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10 (7):e0130140, 2015.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: What is required and can it be learned? In *International Conference on Learning Representations (ICLR)*, 2018.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.0127, 2018.

Maria Barrett, Joachim Bingel, Nora Hollenstein, Marek Rei, and Anders Søgaard. Sequence classification with human attention. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 302–312, 2018.

Joost Bastings, Marco Baroni, Jason Weston, Kyunghyun Cho, and Douwe Kiela. Jump to better conclusions: Scan both left and right. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 47–55, 2018.

John Batali. Artificial evolution of syntactic aptitude. In *Proceedings from the Sixteenth Annual Conference of the Cognitive Science Society*, pages 27–32. Lawrence Erlbaum Associates Hillsdale, NJ, 1994.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James R. Glass. What do neural machine translation models learn about morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Volume 1: Long Papers*, pages 861–872. Association for Computational Linguistics, 2017a.

Yonatan Belinkov, Lluís Màrquez, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. Evaluating layers of representation in neural machine translation on part-of-speech and semantic tagging tasks. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1–10, 2017b.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James R. Glass. On the linguistic representational power of neural machine translation models. *CoRR*, abs/1911.00317, 2019.

Terra Blevins, Omer Levy, and Luke Zettlemoyer. Deep RNNs encode soft hierarchical syntax. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 14–19, 2018.

Ondrej Bojar, Christian Buck, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno-Yepes, Philipp Koehn, and Julia Kreutzer, editors. *Proceedings of the Second Conference on Machine Translation, WMT 2017*, 2017. Association for Computational Linguistics.

Samuel R Bowman, Christopher D Manning, and Christopher Potts. Tree-structured composition in neural networks without tree-structured architectures. In *Proceedings of the 2015th International Conference on Cognitive Computation: Integrating Neural and Symbolic Approaches-Volume 1583*, pages 37–42. CEUR-WS. org, 2015.

Rudolf Carnap. *Meaning and necessity: a study in semantics and modal logic*. University of Chicago Press, 1947.

François Chollet et al. Keras. `https://github.com/keras-team/keras`, 2015.

Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585, 2015.

Morten H Christiansen and Nick Chater. Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23(2):157–205, 1999.

Junyoung Chung, Caglar Gulehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, Anthony Bau, and James Glass. What is one grain of sand in the desert? analyzing individual neurons in deep nlp models. In *Proceedings of the AAAI Conference on Artificial Intelligence AAAI. Honolulu, Hawaii, USA*, 2019.

Misha Denil, Alban Demiraj, Nal Kalchbrenner, Phil Blunsom, and Nando de Freitas. Modelling, visualising and summarising documents with a single convolutional neural network. *CoRR*, abs/1406.3830, 2014.

Roberto Dessì and Marco Baroni. CNNs found to jump around more skillfully than rnns: Compositional generalization in seq2seq convolutional networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), Short Papers*, pages 3919–3923, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

Jeffrey L Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, 1990.

Jeffrey L Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine learning*, 7(2-3):195–225, 1991.

Allyson Ettinger, Ahmed Elgohary, and Philip Resnik. Probing for semantic evidence of composition by means of simple classification tasks. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP, RepEval@ACL 2016, Berlin, Germany, August 2016*, pages 134–139. Association for Computational Linguistics, 2016.

Jerry A Fodor and Zenon W Pylyshyn. Connectionism and Cognitive Archirecture: A Critical Analysis. *Cognition*, 28(1-2):3–71, 1988.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Alona Fyshe, Gustavo Sudre, Leila Wehbe, Nicole Rafidi, and Tom M Mitchell. The semantics of adjective noun phrases in the human brain. *bioRxiv*, page 089615, 2016.

Jonas Gehring, Michael Auli, David Grangier, and Yann N Dauphin. A convolutional encoder model for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), Long Papers*, volume 1, pages 123–135, 2017a.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning, (ICML)*, pages 1243–1252, 2017b.

Lieke Gelderloos and Grzegorz Chrupała. From phonemes to images: levels of representation in a recurrent neural model of visually-grounded language learning. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1309–1319, 2016.

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

Mario Giulianelli, Jack Harding, Florian Mohnert, Dieuwke Hupkes, and Willem Zuidema. Under the hood: Using diagnostic classifiers to investigate and improve how language models track agreement information. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 240–248. ACL, 2018.

E Mark Gold et al. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

Yoav Goldberg. Assessing BERT's syntactic abilities. *CoRR*, abs/1901.05287, 2019.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. Colorless green recurrent networks dream hierarchically. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 1195–1205, 2018.

Xiaodong He and David Golub. Character-level question answering with attention. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1598–1607, 2016.

John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2733–2743, Hong Kong, China, November 2019. Association for Computational Linguistics.

John Hewitt and Christopher D. Manning. A structural probe for finding syntax in word representations. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4129–4138. Association for Computational Linguistics, 2019.

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

Dieuwke Hupkes and Willem Zuidema. Diagnostic classification and symbolic guidance to understand and improve recurrent neural networks. In *Proceedings of Neural Information Processing Systems – Workshop track*, 2017.

Dieuwke Hupkes and Willem Zuidema. Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure (extended abstract). In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5617–5621, 7 2018.

Dieuwke Hupkes, Sanne Bouwmeester, and Raquel Fernández. Analysing the potential of seq-to-seq models for incremental interpretation in task-oriented dialogue. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 165–174, Brussels, Belgium, November 2018a. Association for Computational Linguistics.

Dieuwke Hupkes, Sara Veldhoen, and Willem Zuidema. Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure. *Journal of Artificial Intelligence Research*, 61:907–926, 2018b.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. The compositionality of neural networks: integrating symbolism and connectionism. *To appear in Journal of Artificial Intelligence Research (JAIR)*, 2019a.

Dieuwke Hupkes, Diana Luna Rodriguez, Edoardo Ponti, and Bruni Elia. Internal and external pressures on language emergence: Least effort, object constancy and frequency. in prep, 2019b.

Dieuwke Hupkes, Anand Singh, Kris Korrel, German Kruszewski, and Elia Bruni. Learning compositionally through attentive guidance. In *International Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*, 2019c.

Edmund Husserl. *Logische Untersuchungen*. Max Niemeyer, 1913.

Alexander G Huth, Tyler Lee, Shinji Nishimoto, Natalia Y Bilenko, An T Vu, and Jack L Gallant. Decoding the semantic content of natural movies from human brain activity. *Frontiers in systems neuroscience*, 10:81, 2016.

Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2004.

Pauline Jacobson. The (dis) organization of the grammar: 25 years. *Linguistics and Philosophy*, 25(5):601–626, 2002.

Shailee Jain and Alexander Huth. Incorporating context into language encoding models for fmri. In *Advances in Neural Information Processing Systems*, pages 6628–6637, 2018.

Theo Janssen. *Foundations and applications of Montague grammar*. Mathematisch Centrum, 1983.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1988–1997, 2017.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016.

Jaap Jumelet and Dieuwke Hupkes. Do language models understand anything? on the ability of LSTMs to understand negative polarity items. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 222–231, 2018.

Jaap Jumelet, Willem Zuidema, and Dieuwke Hupkes. Analysing neural language models: contextual decomposition reveals default reasoning in number and gender assignment. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 1–11, 2019.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. In *Proceedings of the International Conference on Learning Representations 2016*, pages 1–13, 2015.

Jean-Rémi King and Stanislas Dehaene. Characterizing the dynamics of mental representations: the temporal generalization method. *Trends in cognitive sciences*, 18(4):203–210, 2014.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, pages 1–13, 2015.

Dan Klein and Christopher D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04. Association for Computational Linguistics, 2004.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. Opennmt: Open-source toolkit for neural machine translation. In Mohit Bansal and Heng Ji, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), System Demonstrations*, pages 67–72. Association for Computational Linguistics, 2017.

Kris Korrel, Dieuwke Hupkes, Verna Dankers, and Elia Bruni. Transcoding compositionally: using attention to find more generalizable solutions. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, page 1–11, 2019.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *35th International Conference on Machine Learning, ICML 2018*, pages 4487–4499. International Machine Learning Society (IMLS), 2018.

Yair Lakretz, German Kruszewski, Theo Desbordes, Dieuwke Hupkes, Stanislas Dehaene, and Marco Baroni. The emergence of number and syntax units in LSTM language models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019.

Yongjie Lin, Yi Chern Tan, and Robert Frank. Open sesame: Getting inside BERT's linguistic knowledge. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 241–253, 2019.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535, 2016.

Adam Liška, Germán Kruszewski, and Marco Baroni. Memorize or generalize? searching for a compositional RNN in a haystack. *CoRR*, abs/1802.06467, 2018.

João Loula, Marco Baroni, and Brenden M Lake. Rearranging the familiar: Testing compositional generalization in recurrent networks. In *Proceedings of the EMNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 108–114, 2018.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

Gary F Marcus, Steven Pinker, Michael Ullman, Michelle Hollander, T John Rosen, Fei Xu, and Harald Clahsen. Overregularization in language acquisition. *Monographs of the society for research in child development*, pages i–178, 1992.

David Marr. Vision: A computational investigation into the human representation and processing of visual information. *Phenomenology and the Cognitive Sciences*, 8(4):397, 1982.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 24:109–165, 1989.

Tom M Mitchell, Svetlana V Shinkareva, Andrew Carlson, Kai-Min Chang, Vicente L Malave, Robert A Mason, and Marcel Adam Just. Predicting human brain activity associated with the meanings of nouns. *science*, 320(5880): 1191–1195, 2008.

Mathijs Mul and Willem Zuidema. Siamese recurrent networks learn first-order logic reasoning and exhibit zero-shot compositional generalization. *CoRR*, abs/1906.00180, 2019.

W. James Murdoch, Peter J. Liu, and Bin Yu. Beyond word importance: Contextual decomposition to extract interactions from lstms. In *ICLR*, 2018.

Matthew Nelson, Imen El Karoui, Kristof Giber, Xiaofang Yang, Laurent Cohen, Hilda Koopman, Sydney Cash, Lionel Naccache, John Hale, Christophe Pallier, and Stanislas Dehaene. Neurophysiological dynamics of phrase-structure building during sentence processing. *Proceedings of the National Academy of Sciences*, 114(18):E3669–E3678, 2017.

Peter Pagin. Communication and strong compositionality. *Journal of Philosophical Logic*, 32(3):287–322, 2003.

Peter Pagin and Dag Westerståhl. Compositionality i: Definitions and variants. *Philosophy Compass*, 5(3):250–264, 2010.

Barbara Partee. Lexical semantics and compositionality. *An invitation to cognitive science: Language*, 1:311–360, 1995.

Martina Penke. The dual-mechanism debate. In *The Oxford handbook of compositionality*. Oxford University Press, 2012.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

Alessandro Raganato and Jörg Tiedemann. An analysis of encoder representations in transformer-based machine translation. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 287–297, 2018.

J Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–658, 1978.

Paul Rodriguez. Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural computation*, 13(9):2093–118, 2001.

Paul Rodriguez, Janet Wiles, and Jeffrey L Elman. A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40, 1999.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. In David E. Rumelhart, editor, *Parallel distributed processing*. MIT Pr., 1986.

Naomi Saphra and Adam Lopez. Do lstms learn compositionally? 2019a.

Naomi Saphra and Adam Lopez. Understanding learning dynamics of language models with svcca. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3257–3267, 2019b.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations (ICLR)*, 2019.

Lloyd S. Shapley. A value for n-person games. *Contributions to the Theory of Games*, (28):307–317, 1953.

Xing Shi, Inkit Padhi, and Kevin Knight. Does string-based neural MT learn source syntax? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1526–1534, 2016.

Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.

Chandan Singh, W James Murdoch, and Bin Yu. Hierarchical interpretations for neural network predictions. In *ICLR*, 2019.

Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2): 159–216, 1990.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

Zoltan Szabó. The case for compositionality. *The Oxford handbook of compositionality*, 64:80, 2012.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, page 4593–4601, 2019a.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R Bowman, Dipanjan Das, et al. What do you learn from context? Probing for sentence structure in contextualized word representations. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019b.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

Ke Tran, Arianna Bisazza, and Christof Monz. The importance of being recurrent for modeling hierarchical structure. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4731–4736, 2018.

Dennis Ulmer, Dieuwke Hupkes, and Elia Bruni. Assessing incrementality in sequence-to-sequence models. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 209–217, Florence, Italy, August 2019. Association for Computational Linguistics.

Betty van Aken, Benjamin Winter, Alexander Löser, and Felix A. Gers. How does bert answer questions?: A layer-wise analysis of transformer representations. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, pages 1823–1832. ACM, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Sara Veldhoen. Semantic adequacy of compositional distributed representations. Master's thesis, University of Amsterdam, 2015.

Sara Veldhoen, Dieuwke Hupkes, and Willem Zuidema. Diagnostic classifiers: Revealing how neural networks process hierarchical structure. In *Pre-Proceedings of the Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches (CoCo @ NIPS 2016)*, 2016.

Jesse Vig and Yonatan Belinkov. Analyzing the structure of attention in a transformer language model. *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 63–76, 2019.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association*

for Computational Linguistics, pages 5797–5808. Association for Computational Linguistics, 2019.

Leila Wehbe, Brian Murphy, Partha Talukdar, Alona Fyshe, Aaditya Ramdas, and Tom Mitchell. Simultaneously uncovering the patterns of brain regions involved in different story reading subprocesses. *PloS one*, 9(11):e112575, 2014.

Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision rnns for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 740–745, 2018.

Ethan Wilcox, Roger Levy, Takashi Morita, and Richard Futrell. What do RNN language models learn about filler–gap dependencies? In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 211–221, 2018.

Janet Wiles and Jeff Elman. Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of the seventeenth annual conference of the cognitive science society*, number s 482, page 487. Erlbaum Hillsdale, NJ, 1995.

Thomas Wolf. Some additional experiments extending the tech report "assessing BERT's syntactic abilities" by yoav goldberg. Technical report, Technical report, 2019.

Wlodek Zadrozny. From compositional to systematic semantics. *Linguistics and philosophy*, 17(4):329–342, 1994.

Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. Gender bias in coreference resolution: Evaluation and debiasing methods. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 15–20, 2018.

List of my publications

Joris Baan, Jana Leible, Mitja Nikolaus, David Rau, Dennis Ulmer, Tim Baumgärtner, **Dieuwke Hupkes**, and Elia Bruni. On the realization of compositionality in neural networks. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 127–137, Florence, Italy, 2019. Association for Computational Linguistics. *See Chapter 8 of this book.*

Mario Giulianelli, Jack Harding, Florian Mohnert, **Dieuwke Hupkes**, and Willem Zuidema. Under the hood: Using diagnostic classifiers to investigate and improve how language models track agreement information. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 240–248. Association for Computational Linguistics, 2018. *See Chapter 5 of this book.*

Jaap Jumelet and **Dieuwke Hupkes**. Do language models understand anything? on the ability of LSTMs to understand negative polarity items. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 222–231, 2018.

Jaap Jumelet, Willem Zuidema, and **Dieuwke Hupkes**. Analysing neural language models: contextual decomposition reveals default reasoning in number and gender assignment. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 1–11, 2019. *See Chapter 7 of this dissertation.*

Kris Korrel, **Dieuwke Hupkes**, Verna Dankers, and Elia Bruni. Transcoding compositionally: using attention to find more generalizable solutions. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, page 1–11, 2019.

Yair Lakretz, German Kruszewski, Theo Desbordes, **Dieuwke Hupkes**, Stanislas Dehaene, and Marco Baroni. The emergence of number and syntax units in

LSTM language models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019. *See Chapter 6 of this dissertation.*

Dieuwke Hupkes and Willem Zuidema. Diagnostic classification and symbolic guidance to understand and improve recurrent neural networks. In *Proceedings of Neural Information Processing Systems – Workshop track*, 2017. *See Chapter 3 of this dissertation.*

Dieuwke Hupkes and Willem Zuidema. Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure (extended abstract). In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5617–5621. International Joint Conferences on Artificial Intelligence Organization, 7 2018. *See Chapter 3 of this dissertation.*

Dieuwke Hupkes, Sanne Bouwmeester, and Raquel Fernández. Analysing the potential of seq-to-seq models for incremental interpretation in task-oriented dialogue. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 165–174, Brussels, Belgium, November 2018a. Association for Computational Linguistics.

Dieuwke Hupkes, Sara Veldhoen, and Willem Zuidema. Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure. *Journal of Artificial Intelligence Research*, 61:907–926, 2018b. *See Chapter 3 of this dissertation.*

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. The compositionality of neural networks: integrating symbolism and connectionism. *Journal of Artificial Intelligence Research*, 2019a. *See Chapter 4 of this dissertation.*

Dieuwke Hupkes, Diana Luna Rodriguez, Edoardo Ponti, and Bruni Elia. Internal and external pressures on language emergence: Least effort, object constancy and frequency. in prep, 2019b.

Dieuwke Hupkes, Anand Singh, Kris Korrel, German Kruszewski, and Elia Bruni. Learning compositionally through attentive guidance. In *International Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*, 2019c. *See Chapter 8 of this dissertation.*

Dennis Ulmer, **Dieuwke Hupkes**, and Elia Bruni. Assessing incrementality in sequence-to-sequence models. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 209–217, Florence, Italy, August 2019. Association for Computational Linguistics.

Sara Veldhoen, **Dieuwke Hupkes**, and Willem Zuidema. Diagnostic classifiers: Revealing how neural networks process hierarchical structure. In *Pre-Proceedings of the Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches (CoCo @ NIPS 2016)*, 2016. *See chapter 3 of this dissertation.*

Samenvatting

Artificiële neurale netwerken zijn opmerkenswaardig goed geworden als modellen voor verschillende natuurlijke taalverwerkingstaken. In dit proefschrift onderzoek ik of dit soort modellen daardoor ook gebruikt kunnen worden om meer over natuurlijke taal te leren. Ik focus specifiek op hiërarchische compositionaliteit in *recurrente neurale netwerken* (RNN). Net zoals het menselijke taalverwerkingssysteem verwerken deze modellen binnenkomende signalen op een *incrementele* wijze, hun *temporele* structuur in acht nemend. Ik stel twee vragen:

a) Zijn RNN modellen in staat om hiërarchisch compositionele structuren correct te verwerken (**gedragsmatige gelijkenis**)?

b) Hoe kunnen we inzicht verkrijgen in de manier waarop ze dat doen (**interpreteerbaarheid van modellen**)?

Ik behandel deze vragen in zes hoofdstukken, die onderverdeeld zijn in drie delen. In deel één beschouw ik *kunstmatige talen*, wat mij in staat stelt om de verwerking van talige structuren in isolatie te bestuderen. In dit gedeelte introduceer ik ook *diagnostische classificatie* – een interpreteerbaarheidstechniek die een belangrijke rol speelt in dit proefschrift – en bezin wat het voor een model betekent om hiërarchische compositionaliteit te kunnen verwerken.

In deel twee bestudeer ik *taalmodellen* die getraind zijn op naturalistische data (Engelse zinnen). Eerder onderzoek wees uit dat dit soort modellen in staat zijn om syntaxgevoelige onderwerp-gezegde relaties te bevatten. Ik onderzoek hoe ze dat doen. Ik presenteer een gedetailleerde analyse van de interne dynamiek van modellen, waarvoor ik *diagnostische classificatie*, *neuron ablatie* en *gegeneraliseerde contextuele decompositie* gebruik.

Ten slotte onderzoek ik in het laatste gedeelte van dit proefschrift of de oplossing die een model vindt gestuurd kan worden door een aangepast leersignaal. Ik gebruik de technieken die eerder in het proefschrift geïntroduceerd zijn om de impact van dit aangepaste leersignaal in kaart te brengen.

Samenvattend presenteer ik in dit proefschrift verscheidene analyses die de geschiktheid van RNN modellen aangaande hierarchische syntactische structuur betreffen, evenals verschillende technieken die gebruikt kunnen worden om deze moeilijk interpreerbare modellen te begrijpen. De resultaten schetsen een positief beeld, dat suggeert dat RNN modellen wel degelijk nuttig kunnen zijn als verklarende modellen van natuurlijke taalverwerking.

Abstract

Artificial neural networks have become remarkably successful on many natural language processing tasks. In this dissertation, I explore if these successes make them useful as *explanatory models* of human language processing. I focus in particular on *hierarchical compositionality* and recurrent neural networks (RNNs), which share with the human processing system the property that they process language *temporally* and *incrementally*. I consider two questions:

i) Are RNNs in fact capable of processing hierarchical compositional structures (**behavioural similarity**)?

ii) How can we obtain insight in how they do so (**model interpretability**)?

I address these questions in six chapters, divided into three parts. In part 1, I consider *artificial languages*, which provide a clean setup in which processing of structure can be studied in isolation. In this part, I also introduce *diagnostic classification* – an interpretability technique that plays an important role in this dissertation – and reflect upon what it means for a model to be able to process hierarchical compositionality.

In part two, I consider *language models* trained on naturalistic data (English sentences). Such models have been shown to capture syntax-sensitive long-distance subject-verb relationships. I investigate *how* they do so. I present detailed analyses of their inner dynamics, using *diagnostic classification*, *neuron ablation* and *generalised contextual decomposition*.

Lastly, in the final part of this dissertation, I consider if a model's solution can be changed through an adapted learning signal. I use several of the previously introduced techniques to analyse the impact of adding this learning signal.

In summary, In this dissertation I present many different analyses concerning the abilities of RNNs to process hierarchical structure, as well as several techniques to open these blackbox models. Overall, the results sketch a positive picture of the usefulness of such models as explanatory models of processing languages with hierarchical compositional semantics.

Titles in the ILLC Dissertation Series:

ILLC DS-2009-01: **Jakub Szymanik**
Quantifiers in TIME and SPACE. Computational Complexity of Generalized Quantifiers in Natural Language

ILLC DS-2009-02: **Hartmut Fitz**
Neural Syntax

ILLC DS-2009-03: **Brian Thomas Semmes**
A Game for the Borel Functions

ILLC DS-2009-04: **Sara L. Uckelman**
Modalities in Medieval Logic

ILLC DS-2009-05: **Andreas Witzel**
Knowledge and Games: Theory and Implementation

ILLC DS-2009-06: **Chantal Bax**
Subjectivity after Wittgenstein. Wittgenstein's embodied and embedded subject and the debate about the death of man.

ILLC DS-2009-07: **Kata Balogh**
Theme with Variations. A Context-based Analysis of Focus

ILLC DS-2009-08: **Tomohiro Hoshi**
Epistemic Dynamics and Protocol Information

ILLC DS-2009-09: **Olivia Ladinig**
Temporal expectations and their violations

ILLC DS-2009-10: **Tikitu de Jager**
"Now that you mention it, I wonder...": Awareness, Attention, Assumption

ILLC DS-2009-11: **Michael Franke**
Signal to Act: Game Theory in Pragmatics

ILLC DS-2009-12: **Joel Uckelman**
More Than the Sum of Its Parts: Compact Preference Representation Over Combinatorial Domains

ILLC DS-2009-13: **Stefan Bold**
Cardinals as Ultrapowers. A Canonical Measure Analysis under the Axiom of Determinacy.

ILLC DS-2010-01: **Reut Tsarfaty**
Relational-Realizational Parsing

ILLC DS-2010-02: **Jonathan Zvesper**
Playing with Information

ILLC DS-2010-03: **Cédric Dégremont**
The Temporal Mind. Observations on the logic of belief change in interactive systems

ILLC DS-2010-04: **Daisuke Ikegami**
Games in Set Theory and Logic

ILLC DS-2010-05: **Jarmo Kontinen**
Coherence and Complexity in Fragments of Dependence Logic

ILLC DS-2010-06: **Yanjing Wang**
Epistemic Modelling and Protocol Dynamics

ILLC DS-2010-07: **Marc Staudacher**
Use theories of meaning between conventions and social norms

ILLC DS-2010-08: **Amélie Gheerbrant**
Fixed-Point Logics on Trees

ILLC DS-2010-09: **Gaëlle Fontaine**
Modal Fixpoint Logic: Some Model Theoretic Questions

ILLC DS-2010-10: **Jacob Vosmaer**
Logic, Algebra and Topology. Investigations into canonical extensions, duality theory and point-free topology.

ILLC DS-2010-11: **Nina Gierasimczuk**
Knowing One's Limits. Logical Analysis of Inductive Inference

ILLC DS-2010-12: **Martin Mose Bentzen**
Stit, Iit, and Deontic Logic for Action Types

ILLC DS-2011-01: **Wouter M. Koolen**
Combining Strategies Efficiently: High-Quality Decisions from Conflicting Advice

ILLC DS-2011-02: **Fernando Raymundo Velazquez-Quesada**
Small steps in dynamics of information

ILLC DS-2011-03: **Marijn Koolen**
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

ILLC DS-2011-04: **Junte Zhang**
System Evaluation of Archival Description and Access

ILLC DS-2011-05: **Lauri Keskinen**
Characterizing All Models in Infinite Cardinalities

ILLC DS-2011-06: **Rianne Kaptein**
Effective Focused Retrieval by Exploiting Query Context and Document Structure

ILLC DS-2011-07: **Jop Briët**
Grothendieck Inequalities, Nonlocal Games and Optimization

ILLC DS-2011-08: **Stefan Minica**
Dynamic Logic of Questions

ILLC DS-2011-09: **Raul Andres Leal**
Modalities Through the Looking Glass: A study on coalgebraic modal logic and their applications

ILLC DS-2011-10: **Lena Kurzen**
Complexity in Interaction

ILLC DS-2011-11: **Gideon Borensztajn**
The neural basis of structure in language

ILLC DS-2012-01: **Federico Sangati**
Decomposing and Regenerating Syntactic Trees

ILLC DS-2012-02: **Markos Mylonakis**
Learning the Latent Structure of Translation

ILLC DS-2012-03: **Edgar José Andrade Lotero**
Models of Language: Towards a practice-based account of information in natural language

ILLC DS-2012-04: **Yurii Khomskii**
Regularity Properties and Definability in the Real Number Continuum: idealized forcing, polarized partitions, Hausdorff gaps and mad families in the projective hierarchy.

ILLC DS-2012-05: **David García Soriano**
Query-Efficient Computation in Property Testing and Learning Theory

ILLC DS-2012-06: **Dimitris Gakis**
Contextual Metaphilosophy - The Case of Wittgenstein

ILLC DS-2012-07: **Pietro Galliani**
The Dynamics of Imperfect Information

ILLC DS-2012-08: **Umberto Grandi**
Binary Aggregation with Integrity Constraints

ILLC DS-2012-09: **Wesley Halcrow Holliday**
Knowing What Follows: Epistemic Closure and Epistemic Logic

ILLC DS-2012-10: **Jeremy Meyers**
Locations, Bodies, and Sets: A model theoretic investigation into nominalistic mereologies

ILLC DS-2012-11: **Floor Sietsma**
Logics of Communication and Knowledge

ILLC DS-2012-12: **Joris Dormans**
Engineering emergence: applied theory for game design

ILLC DS-2013-01: **Simon Pauw**
Size Matters: Grounding Quantifiers in Spatial Perception

ILLC DS-2013-02: **Virginie Fiutek**
Playing with Knowledge and Belief

ILLC DS-2013-03: **Giannicola Scarpa**
Quantum entanglement in non-local games, graph parameters and zero-error information theory

ILLC DS-2014-01: **Machiel Keestra**
Sculpting the Space of Actions. Explaining Human Action by Integrating Intentions and Mechanisms

ILLC DS-2014-02: **Thomas Icard**
The Algorithmic Mind: A Study of Inference in Action

ILLC DS-2014-03: **Harald A. Bastiaanse**
Very, Many, Small, Penguins

ILLC DS-2014-04: **Ben Rodenhäuser**
A Matter of Trust: Dynamic Attitudes in Epistemic Logic

ILLC DS-2015-01: **María Inés Crespo**
Affecting Meaning. Subjectivity and evaluativity in gradable adjectives.

ILLC DS-2015-02: **Mathias Winther Madsen**
The Kid, the Clerk, and the Gambler - Critical Studies in Statistics and Cognitive Science

ILLC DS-2015-03: **Shengyang Zhong**
Orthogonality and Quantum Geometry: Towards a Relational Reconstruction of Quantum Theory

ILLC DS-2015-04: **Sumit Sourabh**
Correspondence and Canonicity in Non-Classical Logic

ILLC DS-2015-05: **Facundo Carreiro**
Fragments of Fixpoint Logics: Automata and Expressiveness

ILLC DS-2016-01: **Ivano A. Ciardelli**
Questions in Logic

ILLC DS-2016-02: **Zoé Christoff**
Dynamic Logics of Networks: Information Flow and the Spread of Opinion

ILLC DS-2016-03: **Fleur Leonie Bouwer**
What do we need to hear a beat? The influence of attention, musical abilities, and accents on the perception of metrical rhythm

ILLC DS-2016-04: **Johannes Marti**
Interpreting Linguistic Behavior with Possible World Models

ILLC DS-2016-05: **Phong Lê**
Learning Vector Representations for Sentences - The Recursive Deep Learning Approach

ILLC DS-2016-06: **Gideon Maillette de Buy Wenniger**
Aligning the Foundations of Hierarchical Statistical Machine Translation

ILLC DS-2016-07: **Andreas van Cranenburgh**
Rich Statistical Parsing and Literary Language

ILLC DS-2016-08: **Florian Speelman**
Position-based Quantum Cryptography and Catalytic Computation

ILLC DS-2016-09: **Teresa Piovesan**
Quantum entanglement: insights via graph parameters and conic optimization

ILLC DS-2016-10: **Paula Henk**
Nonstandard Provability for Peano Arithmetic. A Modal Perspective

ILLC DS-2017-01: **Paolo Galeazzi**
Play Without Regret

ILLC DS-2017-02: **Riccardo Pinosio**
The Logic of Kant's Temporal Continuum

ILLC DS-2017-03: **Matthijs Westera**
Exhaustivity and intonation: a unified theory

ILLC DS-2017-04: **Giovanni Cinà**
Categories for the working modal logician

ILLC DS-2017-05: **Shane Noah Steinert-Threlkeld**
Communication and Computation: New Questions About Compositionality

ILLC DS-2017-06: **Peter Hawke**
The Problem of Epistemic Relevance

ILLC DS-2017-07: **Aybüke Özgün**
Evidence in Epistemic Logic: A Topological Perspective

ILLC DS-2017-08: **Raquel Garrido Alhama**
Computational Modelling of Artificial Language Learning: Retention, Recognition & Recurrence

ILLC DS-2017-09: **Miloš Stanojević**
Permutation Forests for Modeling Word Order in Machine Translation

ILLC DS-2018-01: **Berit Janssen**
Retained or Lost in Transmission? Analyzing and Predicting Stability in Dutch Folk Songs

ILLC DS-2018-02: **Hugo Huurdeman**
Supporting the Complex Dynamics of the Information Seeking Process

ILLC DS-2018-03: **Corina Koolen**
Reading beyond the female: The relationship between perception of author gender and literary quality

ILLC DS-2018-04: **Jelle Bruineberg**
Anticipating Affordances: Intentionality in self-organizing brain-body-environment systems

ILLC DS-2018-05: **Joachim Daiber**
Typologically Robust Statistical Machine Translation: Understanding and Exploiting Differences and Similarities Between Languages in Machine Translation

ILLC DS-2018-06: **Thomas Brochhagen**
Signaling under Uncertainty

ILLC DS-2018-07: **Julian Schlöder**
Assertion and Rejection

ILLC DS-2018-08: **Srinivasan Arunachalam**
Quantum Algorithms and Learning Theory

ILLC DS-2018-09: **Hugo de Holanda Cunha Nobrega**
Games for functions: Baire classes, Weihrauch degrees, transfinite computations, and ranks

ILLC DS-2018-10: **Chenwei Shi**
Reason to Believe

ILLC DS-2018-11: **Malvin Gattinger**
New Directions in Model Checking Dynamic Epistemic Logic

ILLC DS-2018-12: **Julia Ilin**
Filtration Revisited: Lattices of Stable Non-Classical Logics

ILLC DS-2018-13: **Jeroen Zuiddam**
Algebraic complexity, asymptotic spectra and entanglement polytopes

ILLC DS-2019-01: **Carlos Vaquero**
What Makes A Performer Unique? Idiosyncrasies and commonalities in expressive music performance

ILLC DS-2019-02: **Jort Bergfeld**
Quantum logics for expressing and proving the correctness of quantum programs

ILLC DS-2019-03: **András Gilyén**
Quantum Singular Value Transformation & Its Algorithmic Applications

ILLC DS-2019-04: **Lorenzo Galeotti**
The theory of the generalised real numbers and other topics in logic

ILLC DS-2019-05: **Nadine Theiler**
Taking a unified perspective: Resolutions and highlighting in the semantics of attitudes and particles

ILLC DS-2019-06: **Peter T.S. van der Gulik**
Considerations in Evolutionary Biochemistry

ILLC DS-2019-07: **Frederik Möllerström Lauridsen**
Cuts and Completions: Algebraic aspects of structural proof theory

ILLC DS-2020-01: **Mostafa Dehghani**
Learning with Imperfect Supervision for Language Understanding

ILLC DS-2020-02: **Koen Groenland**
Quantum protocols for few-qubit devices

ILLC DS-2020-03: **Jouke Witteveen**
Parameterized Analysis of Complexity

ILLC DS-2020-04: **Joran van Apeldoorn**
A Quantum View on Convex Optimization

ILLC DS-2020-05: **Tom Bannink**
Quantum and stochastic processes