University of Amsterdam

MSc Artificial Intelligence
Master Thesis

From Sequence to Attention

Search for a Compositional Bias in Sequence-to-Sequence Models

by
Kristian Korrel
10381937

November, 2018

36 ECTS
March 2018 - November 2018

Supervisors:
dr. Elia Bruni
Dieuwke Hupkes MSc

Assessor:
dr. Elia Bruni

Institute for Logic,
Language and Computation

Abstract

Recent progress in deep learning has sparked a great, renewed interest in the field of artificial intelligence. This is in part because of achieved superhuman performance on several problems, and great versatility. A trained deep learning model, however, can typically only be applied in a very narrow domain as they only excel on test data that is drawn from the same distribution as the training data. This is exemplified by research on adversarial examples that shows how deep learning models respond on valid and perturbed data. However, even when test data comes from a significantly different distribution than the train data, it may be valid in a compositional sense. Recent research on systematic compositionality has provided evidence that deep learning models generally lack a compositional understanding of the domains that they are trained on.

Compositionality is a feat that is often attributed to humans that allows quick few-shot learning and easy generalization to new domains and problem instances. Such an understanding is also crucial in natural language. In short, the principle of semantic compositionality means that the semantic meaning of a complex expression can be explained by the meaning of its constituents and the manner in which they are combined.

In this thesis we show that although deep learning models are potentially capable of having such an understanding, they typically do not converge on such an solution with regular training techniques. We propose two new techniques that aim to induce compositional understanding in sequence-to-sequence networks with attention mechanisms. Both are founded on the hypothesis that a salient, informative attention pattern helps in finding such a bias and in countering the use of spurious patterns in the data. The first of these methods, *Attentive Guidance*, guides a model in finding correct alignments between input and output sequences. It is a minor extension to existing sequence-to-sequence models and is intended to confirm the aforementioned hypothesis. The second method, the *sequence-to-attention* architecture, involves a more rigorous overhaul of the sequence-to-sequence model with the intention to further explore and exploit this hypothesis. We use existing data sets to show that both methods perform better on tasks that are assumed to correlate with systematic compositionality.

Acknowledgments

First and foremost I would like to extensively thank Elia Bruni and Dieuwke Hupkes who have been my supervisors, advisors and motivators throughout the whole process of writing this thesis. With passion they have helped me focusing my attention on the right research directions and have proposed numerous ideas to think about and work on. I wholeheartedly thank them for all the energy they have put in sparking new ideas, providing feedback on my writing, their interest and help in my personal development and academic career, getting me in touch with the researchers at FAIR Paris and the organization of regular lab meetings.

These lab meetings with fellow graduate students have been a great forum for providing feedback on each others work, exchanging ideas and pointing out interesting research. I thank Germán Kruszewski for his supervision and for providing us with related research. Furthermore I thank Bart Bussmann, Krsto Proroković, Mathijs Mul, Rezka Leonandya, Yann Dubois and Anand Kumar Singh for their contributions in this and wish them the best of luck in their future careers.

In special I want to thank Anand Kumar Singh and Yann Dubois, with whom I worked more intensively, for their honest interest in my research, the fruitful conversations we had, the insights they have given me and the pleasant collaborations.

All participant of the *Project AI* I want to congratulate on their great contributions and efforts and I thank them for the insights they have provided.

Finally I thank my family and friends. Not so much for their technical contributions, but ever more for their love, support and for keeping me sane.

Contents

Chapter 1

Introduction

This thesis summarizes our search for extensions and adaptations of sequence-to-sequence models in order for them to converge on solutions that exhibit more systematic compositionality. We start with an introduction in which we quickly describe some aspects of the current status of deep learning in this aspect; What has been accomplished with this paradigm so far and where it fails. We point out that deep learning models generally lack a compositional understanding of the domains they are trained on, and argue that we should search for new types of architectures and training methods that allow models to more easily converge on solutions with more compositional understanding. Our search for such methods is finally summarized as a set of research questions and contributions.

1.1 Successes and Failures of Deep Learning

In the last couple of years we have seen great leaps of success in the world of Artificial Intelligence (AI). New techniques have solved a great number of longstanding problems or have caused massive performance improvements without much need for expert domain knowledge. This has led to AI-powered technologies becoming increasingly commonplace for businesses, households and personal devices. Most of these successes can be attributed to the subfield of *deep learning*. This is a class of machine learning algorithms which apply, in succession, a multitude of non-linear transformations for feature extraction. Because of their deep nature, they allow for hierarchical representation learning (Goodfellow et al., 2016). The application and development of deep learning models have seen great advancements caused by increases in hardware performance, easier software tools, availability of large-scale labeled data sets, and the use of the ever popular learning technique: loss backpropagation.

The Artificial Neural Networks (ANNs) that are trained using this technique have solved long-standing problems in computer vision (Krizhevsky et al., 2012), speech recognition (Hinton et al., 2012a), and natural language processing (Bahdanau et al., 2014; Wu et al., 2016). Increasingly, deep learning is becoming the de facto technique in the field of AI. The versatility of these systems, and the speed at which ANNs can be constructed and trained makes that they are deployed for a wide variety of domains and problems. Where such problems were historically

tackled by experts with domain knowledge and hand-crafted rules and feature extractors, deep learning methods, as illustrated by the iconic statement of Frederick Jelinek: *"Every time I fire a linguist, the performance of our speech recognition system goes up."* [1,2]

The empirical successes of deep learning models in the first and second decade of the 21st century were preceded by some more theoretical results which show the capabilities of ANNs. Hornik (1991) has shown that single-layer ANNs of appropriate size can approximate any continuous function. These models are therefore *universal function approximators*. In this work, we focus on one particular type of ANNs, namely the Recurrent Neural Network (RNN). Siegelmann and Sontag (1992) have shown that these can be Turing complete with only 886 neurons, meaning that they can simulate any Turing machine and can therefore compute any computable function. Both Hornik's and Siegelmann and Sontag's results are concerned with the representational and computational powers of ANNs, but not with how these models can learn. Given enough expressive power, an RNN would thus be able to represent any function or algorithm, but how do we make it learn the particular algorithm we want?

Most famous deep learning research centers around defining previously unsolved problems and showing how a novel ANN architecture can solve this. There seems to be less interest in how exactly the model constructs a solution to the posed problem, or guiding the model in this search intelligently. One of the problems of deep learning is the interpretability of trained models and their solutions. Although several interpretation and explanation techniques have been tried, mostly in the field of computer vision, ANNs remain for a large part black boxes. It is hard to inspect the inner workings of these models, to intuitively understand how they solve tasks, and what kind of solution they converge on. Even more, we have little control over the type of solutions they converge on. Few have tried to explicitly guide an ANN into finding a specific solution to a problem. These models are mainly *trained by example*, but are given little or no indication about the manner in which they should solve the problem. This can result in models that fit the training examples perfectly, but can't generalize to new test examples.

The work of Zhang et al. (2016) shows empirically that, when a feed-forward ANN has enough parameters relative to the problem size and enough training iterations, it is able to represent and learn any arbitrary input-output mapping. Even functions where the input and output examples are sampled independently. They show this with a classification task where for each input example, they associate a random target label, or in another experiment, for each class label they associate a set of random noise inputs. In both cases, a large enough model is able to fit the training set perfectly. Thus in the extreme case, where ANNs are provided with superfluous parameters and training time, they will find any spurious pattern available in the provided discrete training set to individually map each input to its respective output, similar to a lookup table. This provides, of course, little means to generalize to data outside of the training set. This phenomenon is a form of overfitting. As shown, ANNs are prone to overfitting for randomly sampled data where there is no other mechanism to learn the input-output mapping but to memorize each example individually. However, to some extent this also holds for data that could be explained with an underlying algorithm or function. In the extreme case, an ANN can thus learn a lookup table for the input-output pairs in the provided training data, which generally is not the solution we want it to converge on.

[1] Although there is little debate about whether Frederick has made such a quote, the exact wording remains unknown.

[2] This quote dates before the prevalence of deep learning algorithms, and is thus more concerned with other statistical methods. However it applies to deep learning avant la lettre.

In the extreme case, an overfitted model will have perfect performance on the examples it was trained on, but show undefined behavior on other data. This is related to their sensitivity to adversarial examples. Most image classification models can be easily fooled by small perturbations in the input data. Szegedy et al. (2013) show that when you take an image that the model can classify correctly, and perturb this image so slightly that the changes are barely noticeable by the human eye, they can create an image that the model will misclassify with high confidence. Several research has also shown that RNNs are typically not capable of generalizing to longer lengths than the sequences they were trained on (Cho et al., 2014a; Lake and Baroni, 2018). These are examples of cases in which the test data is drawn from a different distribution than the training data. The capability of generalizing to such out-of-distribution data is often associated with terms as *compositionality*, *systematicity* and *productivity* (Fodor and Pylyshyn, 1988; Marcus, 1998). Recently, multiple authors have constructed training and test sets that should test for the *compositional understanding* of learning agents and the systematic application of functions (Liška et al., 2018; Lake and Baroni, 2018; Weber et al., 2018; Loula et al., 2018; Johnson et al., 2017; Yang et al., 2018). They test standard RNNs with standard learning techniques on these tasks and conclude that the typical solution that these models converge upon do not have a real compositional understanding. In this thesis we argue that this is a useful and important quality, and we focus on increasing this understanding in similar models.

1.2 The Need of Compositional Understanding

Let us first give a quick intuition about compositional understanding with an example. A more detailed elaboration is found in Section 2.1. Lake et al. (2015) mention the difference between humans and deep learning models on the ability to quickly learn new concepts. For example, we humans are able to see cars and other vehicles not only as a single physical entity, but on multiple levels as a hierarchy of connected entities; A car consists of windows, pedals, wheels, a roof, et cetera. These in turn are constructed of bolts, screws, glass and other materials. Not only is there a hierarchy of parts, the way in which they are combined also make the car; If all parts were to be assembled in a completely random way, we could barely call it a car anymore, even when it has all the parts. It is partly this compositional understanding - which is observed in humans (Fodor and Pylyshyn, 1988; Fodor and Lepore, 2002; Minsky, 1988) - that allows for quick learning and generalization. When a new vehicle like the Segway (Nguyen et al., 2004) is introduced, we humans have little problem with figuring out how such a vehicle is constructed and how to classify one. We quickly recognize individual parts like wheels and a steering wheel since we are already familiar with them in some form. The manner in which they are combined define the concept of the Segway. Subsequently, after observing only one example, one could recognize Segways of various forms and sizes. Current deep learning models, however, would have to be retrained on large data sets in order to confidently classify Segways. This is not only inefficient in data and compute time, it greatly inhibits the generalizability and practical use of such systems.

We have earlier stated that current deep learning methods are capable of representing any input-output mapping and are prone to overfitting. Similar results are again shown empirically in the domain of the *lookup tables task* by Liška et al. (2018), which we explain more thoroughly in Section 4.1. In short, the task consists of sequentially applying functions, namely lookup tables, on some input bitstring. In one particular experiment, Liška et al. devised an abnormal training and test set for this domain. In this, they train an RNN on unary table compositions, where each

table has a consistent semantic value, e.g., t_2 always refers to the second table and t_5 always to the fifth, as expected. However, this semantic mapping only holds in unary compositions. The training set also consists of longer compositions of tables. In these longer compositions, they consistently shuffle the semantic meaning of the symbols, e.g., in the context of the binary compositions $t_2\ t_6$ and $t_5\ t_2$, the symbol t_2 always refers to the fifth table (and thus t_5 would refer to any other of the tables). This change in training data does not affect the overall performance curve. They conclude that the models do not learn that the individual symbol t_2 has an intrinsic semantic meaning and that the models learns rules on how to apply this function in longer compositions, but rather learns only the semantic meaning of the entire input sequence. In other words, the model would not apply t_2 and t_6 in sequence, but instead treats $t_2\ t_6$ as one single function. One could argue that this is not a compositional solution as the model does not learn the meaning of constituent parts and how they are combined, but learns only the meaning of the entire expression as is. Although the provided training data is arguably of high enough quality and quantity to teach a human the semantic meaning of all individual input symbols, and also the semantic meaning of a composition of such symbols in a sequence, an ANN trained with regular backpropagation generally does not find a similar solution. By partially hand-crafting the weights of an RNN to represent a finite-state automaton, Liška et al. show however that this is not a problem of representational powers of RNNs, but rather a problem of learning the correct function.

The aforementioned problem might again be seen as a case of overfitting, a problem widely discussed in machine learning research. Many solutions have been proposed to counter-attack this phenomenon. In general, these methods try to reduce the problem of converging on a lookup function by reducing the expressive power of the model. Ideally this could be used to find the right balance between enough expressive power to find a solution to the problem, but not enough expressive power to memorize the entire the training set. Maybe the simplest of methods is reducing the number of learnable, free parameters in the model. Other notable methods include weight regularization (Krogh and Hertz, 1992), Dropout (Hinton et al., 2012b) and data augmentation (Taylor and Nitschke, 2017). All of these methods have shown great improvements in tackling the problem of overfitting in general, but have not been shown to be successful in helping to find a compositional solution.

We argue that the problem shown by Liška et al. (2018) and others involves more than this classical interpretation of overfitting and could be attacked differently. Statistical approaches like Dropout and data augmentation can greatly improve the robustness of a model against adversarial attacks (Guo et al., 2017). They can also prevent the model from honing in on spurious pattern in the training data. However, these regularization methods approach the problem from a statistical view. Most regularization techniques try to "spread the workload"; Instead of a small portion of the neurons activating on extremely specific input patterns, they try accomplish more robustness. To some extent, this forces the model to utilize all aspects of the inputs it receives and to add more redundancy in the model. We believe, however, that this does not necessarily rigorously change the type of solutions the models converge on. More specifically, there is still little incentive for the network to find a compositional solution.

We hope that the reader is convinced of the need of a compositional understanding in learning agents as this allows for easier few-shot learning, understanding of natural language, productivity, less training data, and more efficient learning by recombining already learned functions. We hypothesize that the attention mechanism (Bahdanau et al., 2014; Luong et al., 2015) in sequence-to-sequence networks (Cho et al., 2014b; Sutskever et al., 2014) could play an important role in

inducing this bias. We therefore develop and test two techniques that assess to which degree this hypothesis is correct.

1.3 Research Questions and Contributions

In this thesis we aim to induce systematic compositionality in sequence-to-sequence networks. To objectively assess this, a way to quantify such an understanding must first exist. Our first research question thus concerns the search for effective ways to quantify the amount of compositional understanding in models. When such methods are defined, the second research question asks whether we are able to improve on standard sequence-to-sequence models in compositional understanding. Specifically, we want to research to which degree a well-controlled, informative and sparse attention mechanism in sequence-to-sequence models helps in forming systematic compositional behavior.

Our contributions are as follows. First, we provide an extension to the training of sequence-to-sequence models, to be called *Attentive Guidance*. This guides the model into finding informative and sparse attention patterns. Attentive Guidance is observed to improve compositional understanding, but requires training data to be additionally annotated with attention patterns.
The second contributions is the development and testing of the *sequence-to-attention* architecture. This is a modification of the standard sequence-to-sequence network that is designed to rely fully on the attention mechanism of the decoder. With the sequence-to-attention model, similar results are obtained as with Attentive Guidance. However, the design of the model makes that it can reach similar compositional understanding, without the need for annotated attention patterns.
Both methods are used to assess whether an informative attention mechanism can aid in compositional generalization. As a secondary benefit, they provide intuitive insights in how models solve certain tasks and thus contribute to more interpretable AI.

Work on Attentive Guidance is also published separately (Hupkes et al., 2018a), and was joint work with fellow student Anand Kumar Singh, who came up with the original idea and implementation, and my supervisors Elia Bruni, Dieuwke Hupkes and Germán Kruszewski. I personally helped extensively on porting the implementation to a new codebase, which allowed us to extensively test the technique but also make changes to it. The experiments of the symbol rewriting task (Section 4.2) were entirely my work, while the experiments on the lookup table task (Section 4.1) were performed in collaboration with Anand. I also contributed to the writing in Hupkes et al. (2018a). All text in this thesis, including Chapter 5 (Attentive Guidance), is written by me.

The following of this thesis is structured in the following way. We first provide some background knowledge on concepts like *compositional understanding* and the types of architectures we will work with in Chapter 2. This is followed by a small chapter on relevant related work. Next, in Chapter 4, we describe the test sets that we will use to assess the amount of compositional understanding. Attentive Guidance and the Seq2Attn network are described in separate chapters, each with their own introduction, method section, results and conclusions (Chapters 5 and 6). Finally we conclude our findings of both methods and give recommendations for future work in Chapter 7.

Chapter 2

Background

In this chapter we will give a more detailed explanation of concepts that will be used throughout the rest of this document. The main aim of this thesis is to induce a bias towards more compositional understanding in sequence-to-sequence models. We therefore first provide an interpretation of the concept of *compositional understanding* and the benefits it provides for efficient learning and representation. Later we will provide a more technical explanation of *sequence-to-sequence models* for those that are less familiar with this branch of deep learning.

2.1 Compositional Understanding

In this thesis, we search for new deep learning architectures and learning mechanisms in order to arrive at models with more compositional understanding than usually found. It is therefore crucial that we first grasp what a *compositional understanding* entails. In this section, we will try to expound this to all readers and motivate how this quality is beneficial for learning agents to possess. We will review concepts like *compositionality*, *systematicity* and *productivity*, of which Fodor and Pylyshyn (1988) discussed their implementation and use of in the human brain.[1]

Compositionality Let us start by recalling what the principle of semantic compositionality entails. This term is often used in linguistics and mathematics and is, in that context, the principle that the semantic meaning of a complex expression is a function only of the meanings of its syntactic parts together with the manner in which these parts were combined. This is also called Frege's Principle (Pelletier, 1994).[2] To understand a sentence like *"Alice sees Bob."*, we humans use the principle of compositionality. Let's break this principle down in two.

Firstly we must conclude that the sentence should not be looked at as an atomic concept. In-

[1] Ironically, they try to make formal arguments about why Connectionist models are not a viable explanation of the human mind at a cognitive level. It must be noted that although they invalidate the idea that Connectionist models can explain the human mind on a cognitive level, such models could still be used as an underlying neural structure on which a Classical architecture is implemented.

[2] Although called Frege's Principle, it is unknown whether Gottlob Frege stood by it (Pelletier, 2001).

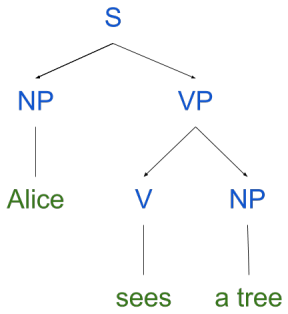


Figure 2.1: Semantic parse tree of *"Alice sees a tree."*

stead, it syntactically consists of multiple constituents or smaller parts that can have an intrinsic meaning on their own. If we would view each sentence atomically, there would be no way to induce the meaning of a sentence like *"Alice sees Bob."* when we would already be familiar with a sentence like *"Bob sees Alice."*, just as you cannot induce the meaning of *apple* from the meaning of *car*. We must thus identify the individual words in this sentence, which would be *Alice*, *sees* and *Bob*.[3] Instead of treating the expression as one, we look at the meaning of all individual words, we look at the ordering of the words and from that we induce the meaning of the sentence. This allows us to reuse this knowledge to understand or construct similar sentences.

Secondly we must also conclude that we must be able to understand the non-trivial manner in which the constituent parts are combined. In language this of course holds a close relation to the semantic parsing of a sentence. The parse tree of a sentence dictates to a large degree how the meaning of the individual words are to be combined to form a semantic understanding of the entire sentence. In language and most other domains, the composition function is not as trivial as summing up or averaging the meaning of all constituents in an expression. Ironically, in deep learning, the meaning of an expression is sometimes naively estimated by averaging the vectorized representations of its constituents. The context in which the constituents appear may dictate their individual meaning and the meaning of the entire sentence.

Systematicity Systematicity is very related to the principle of compositionality. Fodor and Pylyshyn mention that, in the context of linguistics, it entails that the ability to understand or produce some sentences is intrinsically connected to the ability to understand or produce other sentences. Systematicity might be interpreted as the sequential application of certain rules in order to end up in a similar state. Let's consider the simple example of *"Alice sees a tree."*. By systematically applying a set of rules, we can create a semantic parse tree of this (Fig. 2.1). By parsing sentences in a systematic manner, we can easily infer the semantic meaning of similar sentences like *"Alice sees a house."* and *"Bob sees a tree."* or any of the other countless combination, given that we already know the intrinsic meaning of these physical objects.

This has other implications as well. From sentences like *"Alice sees a jarb."* and *"Jarb sees a tree."*, direct inference can be done about the semantic meaning of *jarb*; Whether it is a living being, or an object, and whether it is visible by the human eye. Going a step further, when

[3]It would also be possible to go one level deeper and look at how words and sentences are constructed by characters (in text) or phones (in speech).

understanding that *jarb* is a singular noun, new sentences like *"Carol holds two purple jarbs in her hand."* can be produced (or composed) systematically, in which *jarb* is both syntactically changed (pluralized) and combined with other constituents (purple) to form a more complex expression and understanding.

As Fodor and Pylyshyn point out, the systematicity of natural language justifies that, when learning a language, we do not memorize exhaustive phrase books with all phrases that could possible be uttered in this new language. Instead we learn the semantic meaning of a set of words, and a much smaller set of rules by which sentences can be constructed. In this respect, systematicity is again closely related to the concept of productivity.

Productivity Productivity in linguistics may be seen as the limitless ability to produce new, syntactically well-formed sentences. Or, as explained from a different angle, it is the ability to assess, of a newly seen sentence, whether it is formed correctly and follows the right syntactical rules (Chomsky, 2006). Emphasis should be placed on the fact that natural language, in theory, supports the ability to construct unbounded sentences. A trivial example being the explicit listing of all existing natural numbers. Such an infinite generative or understanding process can of course also prove useful in other domains. Given a pen and long enough paper, any human (or a finite state machine for that matter) could solve the binary addition of two arbitrarily long sequence. Similarly to the concepts of compositionality and systematicity, an agent that possesses the understanding of productivity is able to produce or understand expressions of arbitrary length because it understands the underlying generative and explanatory rules of the language, instead of memorizing each single example sentence.

In the following of this thesis we will use the term *compositional understanding* as a more broad term of the ability to understand, process or produce expressions that require one or more of the above explained concepts. To once more illustrate the productive possibilities and efficient learning that a compositional understanding provides, let's consider the following example.

Imagine a parent teaching its child to follow navigational directions (in New York City) in order to get to school and to the grocery store. An example instruction could be *"Turn right, walk for one block, turn left, walk for two blocks."* to get to the grocery store, while the parent would utter *"Turn right, walk for one block, turn left, walk for two blocks, turn left, walk for one block."* to get to school. One can imagine that if the child distills the semantic meaning of *turn left*, *turn right* and *walk for x block(s)* and knows how to count, it could theoretically apply an instruction of infinite length (productivity).
The above ability is also enabled by the systematic application and understanding of such sentences (systematicity). In addition, the same principle also allows the child to apply this method in Paris or any other city in the world. If the child has thus distilled the correct meaning of the individual parts and knows how to combine them, it can generalize this solution to greatly varying instances.
Compositionality comes into play at two different levels. Firstly, it is required to understand how the meaning should be derived from a well-formed instruction. This includes understanding all the atomic instructions, and understanding that these instructions are to be applied in sequential order. Next, it it also necessary to understand the meaning of composed instructions. If the child is asked to go to the grocery store and to school, it would be wise to combine this into one trip. It could go to the grocery store, then turn left and walk for one block in order to arrive at school. A compositional understanding would furthermore enable extremely efficient learning and adaption. Let's assume that the family has moved to Paris and has to follow navigational

Training set					
Input			Output		
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
1	1	0	1	1	0

(a)

Original test set					
Input			Output		
0	0	1	0	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	1

(b)

Alternative test set					
Input			Output		
0	0	1	0	0	0
0	1	1	0	1	0
1	0	1	1	0	0
1	1	1	1	1	0

(c)

Table 2.1: Training set and possible test sets of a bitstring copy task. Note that only the final bit is different between the training and the original test set. Courtesy of Marcus (1998)

instructions in French. From only a few examples like *"Tourne à droite, marche sur un pâté de maisons, tourne à gauche, marche pour deux pâtés de maisons."* to get to school, it could associate *à droite* with *right*, *à gauche* with *left*, et cetera. By learning the similar semantic meanings of these symbols, it would be able to reuse all navigational knowledge it has gained, even when the instruction is provided in French.

We hope that the above example provides an insight in the usefulness of a compositional understanding in learning agents to produce or understand complex expressions, and to learn efficiently. In order to generalize to unseen cities, routes and instructions, the agent must be able to understand the complex instructions on multiple levels, it should understand the overall navigational instruction, as well as the meaning of individual words and the phones of letters and syllables. This allows to reuse knowledge in new situations, but also effective learning; As the compositional structure of navigational instructions in French are similar to those in English, these do not have to be relearned from scratch.

Arguably, such understanding has so far only been minimally shown in deep learning models, which might not be too surprising. Marcus (1998) provides an example that shows how a compositional understanding and the current learning techniques in deep learning can be at conflict. Let's consider the (arguably) simple task of directly copying a bitstring. Tables 2.1a and 2.1b show an example of a training and test set for this task. The training set includes only even binary numbers where the final bit is always zero. The test set contains almost identical bitstrings, with the only difference that the final bit is set to one.

Marcus found humans, after seeing only the training data, to consistently infer the *sameness* relation between the input and output. They therefore generalize perfectly to the test set. Trained deep learning models, on the other hand, would consistently miss out on this. E.g., for the test example [1 1 1], models would generally output [1 1 0], ignoring the sameness relation of the first few bits, and consistently outputting zero for the final bit, as this policy was optimal for the training set. With *optimal* we mean here optimality in the context of loss minimization. The models are trained using backpropagation of a certain loss function. One could imagine that in a different definition of optimality, e.g., in one that takes into consideration the number of parameters or minimum description length, a policy in which *all* input bits are simply copied might be more optimal. Note however that from a mathematical perspective, the policy chosen by the deep learning models is perfectly valid. By a maximum likelihood estimation of only the training set, the estimated conditional probability of the final bit being 1 is zero. If the original task was changed to be *"Copy the first two bits and then output 0"*, the task would have the same training set (Table 2.1a), but would now use an alternative test set (Table 2.1c) on which deep

learning models would beat humans.

Given the evidence of only the training examples, one thus cannot say which of the two policies is optimal in terms of generalization capacities. This is only evident when one knows on what test set it will be tested on. One could also say that a learning agent, be it a human or ANN, that is only trained on a training set *by example*, cannot always infer *how* the task should be solved. The training data shows which task should be solved, or technically only instantiations of this task, but provides no explicit means of inferring how this should be done. The difference between humans and deep learning models on this and similar tasks are thus a matter of prior bias. With simple loss minimization on a limited training set, a deep learning model will not know on what data it will be tested on, and can therefore not know what the optimal policy will be. This is a bias that has to be inserted into the model *"from above"*. In this thesis we test two method to add such a bias; One of these changes the learning objective, and one changes the architecture of the model.

2.2 Sequence to Sequence Models

In this section we give a global introduction to sequence-to-sequence models and attention mechanisms.

2.2.1 Introduction to Encoder-Decoder Models

In modern-day deep learning, three common types of models could be distinguished that are typically aimed at different kinds of data representations. Of these, the fully-connected feed-forward neural network could be considered the most basic one. This type of networks is often used for non-sequential data of which the order in which the data is presented is not relevant. For data where the ordering is important, e.g., because of spatial or spatiotemporal structure, a Convolutional Neural Network (CNN) is often used. The most common use of this is for image and video data (Girshick, 2015; Zhu and Ramanan, 2012; Ilg et al., 2017). Because language is assumed to have a lot a local structure as well, multiple successful attempts have been made to use CNNs in the domain of (natural) language (Gehring et al., 2017; Kim, 2014). However, language is also often interpreted as a sequence, for which a different type of neural network can be deployed.

For time-dependent or sequential data-like language, stock prices or (malicious) network traffic over time (Radford et al., 2018), the number of data points in a sequence is often not known in advance and can vary. For this type of sequential data the Recurrent Neural Network (RNN) cell has widely been applied in the past years. The typical property of this cell is that it can be applied over each item in the sequence recurrently. The same operation is applied over each token in the sequence and it has thus shared parameters over time. The actual implementation, however, can vary, and we can distinguish three widely known implementations, which we will further discuss in Section 2.2.3.

Such an RNN cell can be employed in a model differently depending on the task. For the task of text classification, an RNN cell may be used to "read in" the sentence on character or word-level (Liu et al., 2016). The RNN is expected to accumulate and process all relevant information

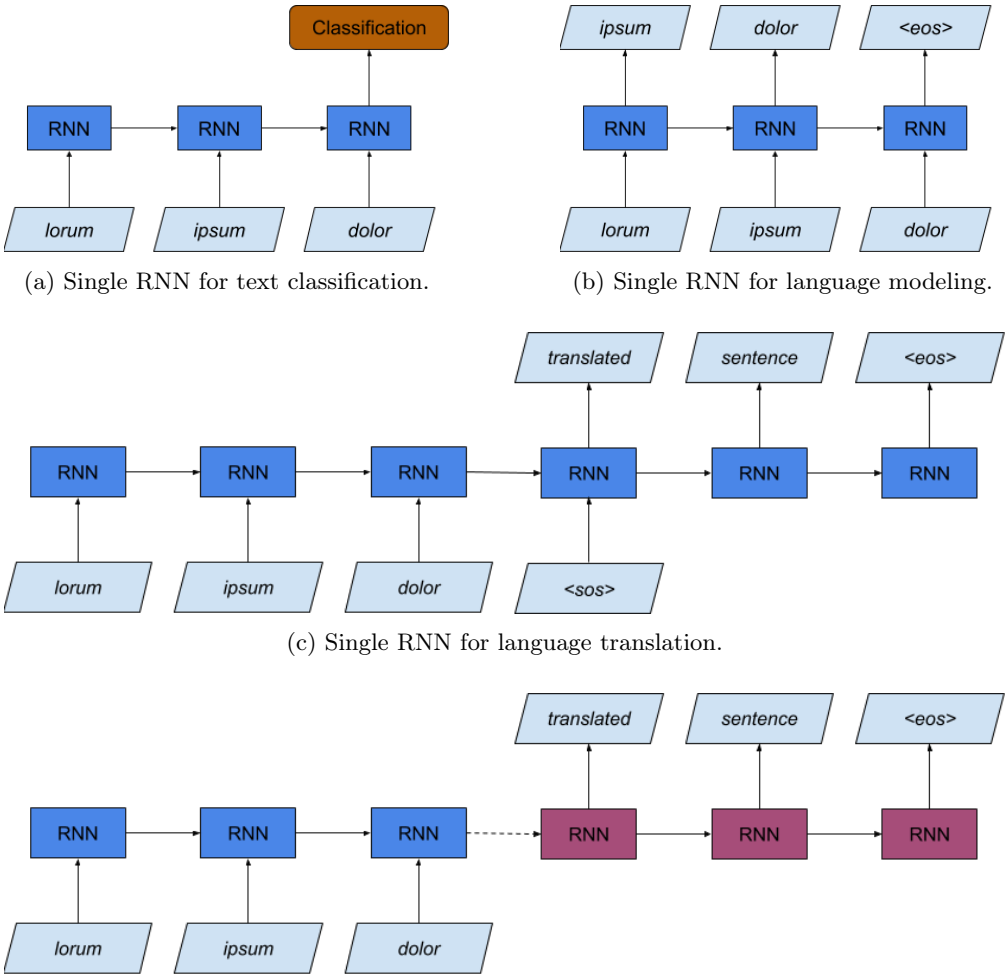
via its recurrent layer, and all evidence for a correct classification should thus be stored in the final cell's hidden state. Based on the final RNN hidden state, one or more feed-forward layers may be used to do the classification (Fig. 2.2a). RNNs can also be used for language modeling. For this, the predicted output character can be used as input to the next RNN cell (Fig. 2.2b). Graves (2013) has shown that by probabilistically sampling the output character, such a model can generate a wide variety of texts. The tasks of sentence classification and language modeling could be considered very related to language translation. The successes of RNNs in the former fields has thus also sparked interest in the use of RNNs for Neural Machine Translation (NMT). For translation, it is generally assumed that the entire context of the source sentence has to be known before it can be translated into a target language. When using a single RNN for a task like NMT, the output sequence modeling is therefore often delayed until the entire input sentence has been processed. After that, a special Start Of Sequence (SOS) input symbol initiates the process of output modeling (Fig. 2.2c). However, recent research has shown the use of only a single RNN for this task to be suboptimal.

Cho et al. (2014b) and Sutskever et al. (2014) have introduced the encoder-decoder models for sequence-to-sequence tasks like NMT. In these types of models, two separate RNNs are used for encoding the input sequence into a single state, and for decoding this state and modeling the output sequence respectively (Fig. 2.2d). The encoder reads in the input sequence as a regular RNN model and accumulates and processes all relevant information. The last hidden state is expected to have encoded all information relevant for the decoder. The decoder is another RNN with independent parameters, of which the first hidden state is initialized with the final hidden state of the encoder. This decoder then models the output sequence. The encoder-decoder architecture increases the number of parameters and allows for specialized RNNs for the two separate tasks. In order for the encoder and decoder to work in different manifolds or to have different hidden sizes, an additional transformation may be applied on the final hidden state of the encoder before it is used to initialized the decoder.

Intuitively, one might quickly think that the encoder-decoder architecture has quite a (literal) bottleneck. The decoder - which should model the target sentence - is conditioned on solely the encoded vector produced by the encoder. Increasing the RNN size of the encoder might increase the theoretical information that can be passed from the encoder to the decoder, but might introduce overfitting behavior and increase memory and computational requirements. Furthermore, current RNN implementations - including even the LSTM and GRU cells (Section 2.2.3) - can have problems with retaining long-term dependencies. For a longer input sequence, it is thus hard for the encoder to retain information over a longer period of time steps, resulting in an encoding with missing information about the beginning of the source sentence. Recently, Luong et al. (2015) and Bahdanau et al. (2014) have introduced implementations of attention mechanisms. These mechanisms allow the decoder to access and recall information stored in hidden states of not only the final encoder state, but all encoder states. In the next section we will examine the implementation of these attention mechanisms more thoroughly.

2.2.2 Attention Mechanisms

The effective use of sequence-to-sequence networks has greatly increased by the introduction of attention mechanisms (Luong et al., 2015; Bahdanau et al., 2014). In attention-augmented networks, instead of fully relying on the final hidden state of the encoder, the decoder additionally receives information from other hidden states of the encoder. The decoder is equipped with an



(a) Single RNN for text classification.

(b) Single RNN for language modeling.

(c) Single RNN for language translation.

(d) Encoder-decoder architecture for language modeling, using two separate RNNs.

Figure 2.2: RNN cells used in different configurations for different tasks. Note that these are basic schematics and that actual implementation might significantly differ.

ability to look at the entire history of the encoder, allowing specialized information retrieval and more effective use of the internal memories of both the encoder and decoder.

In this thesis we will only work with global, content-based attention as used by both Bahdanau et al. and Luong et al.. This means that all encoder states can be attended to (global) and that this attention is based on the contents of hidden states of the encoder and decoder (content-based). Luong et al. also describe global, location-based attention, in which the decoder computes the locations of the encoder states that are attended to as a function of only its own hidden state. Global attention, however, might be computationally inefficient or impractical for tasks with long input sequences, as all encoder states are attended to. They therefore also describe a local attention method in which only a subset of the encoder states can be attended to at one time. In CNNs, similar attention mechanism have also been introduced (Xu et al., 2015; Gregor et al., 2015). We view these other attention mechanisms as irrelevant and impractical for our current research. However, they might be considered for future work.

Before providing an explanation of this global, content-based attention, we first address a technical detail. Vanilla RNN and GRU cells have only one output vector. LSTM cells, on the other hand, have a memory state $\mathbf{c}_t$ and output vector $\mathbf{h}_t$, which are both used in the recurrency of the network. When we refer to the *(hidden) state* of an encoder or decoder, we generally refer to $\mathbf{h}_t$ in the case of LSTMs. However, we hypothesize that the exact choice is minimally relevant for performance. One could also use $\mathbf{c}_t$ or a concatenation of both.

We describe the used attention mechanism in line with the work of Luong et al.. As we are using content-based attention, the hidden states of the decoder and encoder are used to compute the alignment. This means that for each decoder step $t \in \{1, \ldots, M\}$ we compute a score for each encoder step $s \in \{1, \ldots, N\}$, with N and M being the lengths of the input and output sequences respectively. This scoring is done with a scoring or alignment function. Multiple alignment functions have been used in earlier work, some with and some without learnable parameters. In this thesis we will use the following alignment functions.

$$\text{score}(\mathbf{h}_t, \mathbf{h}_s) = \begin{cases} \mathbf{h}_s^\intercal \mathbf{h}_t & \text{(dot)} \\ \boldsymbol{v}_a^\intercal [\mathbf{h}_s; \mathbf{h}_t] & \text{(concat)} \\ \boldsymbol{v}_a^\intercal \text{ReLU}(\boldsymbol{W}_a [\mathbf{h}_s; \mathbf{h}_t]) & \text{(mlp)} \end{cases} \tag{2.1}$$

The *dot* method is also used by Luong et al.. Both Luong et al. and Bahdanau et al. also use a slightly different version of the mlp method in which the *tanh* function is used instead of the *ReLU* activation. For the *concat* method, $\boldsymbol{v}_a$ is a $[D_e + D_d \times 1]$ vector, where D_e and D_d are the dimensionalities of the encoder and decoder RNN cells respectively. For the mlp method, $\boldsymbol{W}_a$ and $\boldsymbol{v}_a$ are $[D_e + D_d \times D_a]$ and $[D_a \times 1]$ matrices respectively, where D_a is a parameter to choose. In all of our experiments we used $D_e = D_d = D_a$.

In the global attention mechanism, for any decoder step, all encoder states are attended to with varying degree. All information is theoretically available to the decoder, and this operation is fully differentiable. A probability vector $\mathbf{a}_t$ is created of which the length is the same as the number of encoder states. We call this the *attention vector* and it represents the degree to which each encoder state is attended to. It is usually calculated with the Softmax function.[4]

[4]When deemed necessary we will use superscript to disambiguate between encoder, decoder (and transcoder). In other cases it is intensionally left implicit.

$$\mathbf{a}_t(s) = \text{align}(\mathbf{h}_t^{dec}, \mathbf{h}_s^{enc}) = \frac{\exp\{\text{score}(\mathbf{h}_t^{dec}, \mathbf{h}_s^{enc})\}}{\sum_{i=1}^{N} \exp\{\text{score}(\mathbf{h}_t^{dec}, \mathbf{h}_i^{enc})\}} \tag{2.2}$$

The normalized alignment scores $\mathbf{a}_t(s)$ are used as the weights in a weighted average over the encoder states that are attended to. From a pair of encoder state and decoder state, an alignment score is thus calculated, which represents the weight with which the decoder attends to this encoder state. The weighted average over the encoder states is often referred to as the *context vector*.

$$\mathbf{c}_t = \sum_{s=1}^{N} \mathbf{a}_t(s) \cdot \mathbf{h}_s^{enc} \tag{2.3}$$

Luong et al. and Bahdanau et al. incorporate this context vector into the decoder in different ways. At time step t of the decoder, Luong et al. calculate the context vector based on the current decoder state $\mathbf{h}_t^{dec}$. Subsequently, the context vector is concatenated with the output of the current decoder $\mathbf{y}_t^{dec}$ to form the new output of the RNN cell $\bar{\mathbf{y}}_t^{dec} = [\mathbf{y}_t^{dec}; \mathbf{c}_t]$. This output can then be used to model the output distribution with additional feed-forward layers and a Softmax activation. The order of calculation could thus be summarized as $\mathbf{h}_t^{dec} \rightarrow \mathbf{a}_t \rightarrow \mathbf{c}_t \rightarrow \bar{\mathbf{y}}_t^{dec}$. Since the context vector is calculated at each decoder step independently, and not used in the recurrency of the decoder, we call this approach *post-rnn attention*. They view this approach as simpler than the one which was utilized by Bahdanau et al.. Their incorporation of the attention mechanism in the decoder can be summarized as $\mathbf{h}_{t-1}^{dec} \rightarrow \mathbf{a}_t \rightarrow \mathbf{c}_t \rightarrow \bar{\mathbf{x}}_t^{dec}$. At time step t, not the current decoder state $\mathbf{h}_t^{dec}$,but the previous decoder state $\mathbf{h}_{t-1}^{dec}$ is used to calculate the attention vector and context vector. This is then concatenated to the input of the current decoder step to get the new input $\bar{\mathbf{x}}_t^{dec} = [\mathbf{x}_t^{dec}; \mathbf{c}_t]$. The context vector can thus be incorporated in the recurrency of the decoder, allowing the attention mechanism and decoder to condition on previous attention choices.[5] We will henceforth refer to this scheme as *pre-rnn attention*. We have experimented with both post-rnn and pre-rnn attention in combination with all the three alignment functions. However, pre-rnn attention in combination with the mlp alignment function is used as the default method because of its observed superior performance.

2.2.3 RNN cells

Both the encoder and decoder in a sequence-to-sequence network are recurrent neural networks. However, we can distinguish different variants of RNN cells. In this thesis we will generally express these state transition models as

$$\mathbf{y}_t, \mathbf{h}_t = \mathcal{S}(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{2.4}$$

They thus take as input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$. They output both the next hidden state $\mathbf{h}_t$ and an output $\mathbf{y}_t$. Note that $\mathbf{y}_t$ is the output of the RNN cell itself. This can

[5] It must be noted that Luong et al. also appreciate the possible benefits of incorporating the attention mechanism in the recurrency of the decoder. They therefore also propose the *input-feeding approach* in which they concatenate $\bar{\mathbf{y}}_t$ to the input of the next decoder step.

additionally be fed through one or multiple linear layers to model the actual output distribution.

Vanilla RNN The simplest RNN cell that we distinguish is the vanilla RNN. In this cell, the output equals the next hidden state.

$$\mathbf{y}_t = \mathbf{h}_t = \tanh(\boldsymbol{W}_{hh}\mathbf{h}_{t-1} + \boldsymbol{W}_{xh}\mathbf{x}_t) \tag{2.5}$$

This is similar to concatenating the previous hidden state and the input, and transforming them with a linear layer. This is then activated with the *tanh* function. Note that we omit bias units for simplicity.

Although its simplicity makes it great for introducing the concept of RNN's, it is not practical for learning long-term dependencies, and can be unstable during training. When backpropagating through time, the RNN is unrolled and is thus equivalent to a deep feed-forward network with the same matrix multiplication and activation function performed at every layer. Thus, the derivative of the loss with respect to the earlier states involves a multitude of multiplications of $\boldsymbol{W}_{hh}$ and the derivative of *tanh*. When $\|\boldsymbol{W}_{hh}\| < 1$ or $\tanh' < 1$, this can result in vanishing gradients, which disables learning (Pascanu et al., 2013). Alternatively, when $\|\boldsymbol{W}_{hh}\| > 1$, this might result in unstable exploding gradients. In our experiments we thus do not use this RNN cell.

LSTM The Long-Short Term Memory (LSTM) cell addresses the problem of vanishing gradients by having the identity function as activation function of the recurrent layer (Hochreiter and Schmidhuber, 1997). However, since the norm of the recurrent weights may still be larger than 1, it can still show exploding gradients.

The LSTM cell can be described as

$$i_t = \sigma(\boldsymbol{U}_{hi}\mathbf{h}_{t-1} + W_{xi}\mathbf{x}_t) \tag{2.6}$$

$$f_t = \sigma(\boldsymbol{U}_{hf}\mathbf{h}_{t-1} + W_{xf}\mathbf{x}_t) \tag{2.7}$$

$$o_t = \sigma(\boldsymbol{U}_{ho}\mathbf{h}_{t-1} + W_{xo}\mathbf{x}_t) \tag{2.8}$$

$$\tilde{\mathbf{h}}_t = \tanh(\boldsymbol{U}_{hh}\mathbf{h}_{t-1} + W_{xh}\mathbf{x}_t) \tag{2.9}$$

$$\mathbf{h}_t = \sigma(f_t \cdot \mathbf{h}_{t-1} + i_t \cdot \tilde{\mathbf{h}}_t) \tag{2.10}$$

$$\mathbf{y}_t = o_t \cdot \tanh(\mathbf{h}_t) \tag{2.11}$$

GRU The Gated Recurrent Unit (GRU) is another well-known recurrent cell that solves the vanishing gradient problem (Cho et al., 2014a). Since it has fewer parameters, it is faster to train, but shows comparable performance to the LSTM (Chung et al., 2014). The GRU can be summarized as

$$z_t = \sigma(\boldsymbol{U}_{hz}\mathbf{h}_{t-1} + W_{xiz}\mathbf{x}_t) \tag{2.12}$$

$$r_t = \sigma(\boldsymbol{U}_{hr}\mathbf{h}_{t-1} + W_{xir}\mathbf{x}_t) \tag{2.13}$$

$$\tilde{\mathbf{h}}_t = \tanh(\boldsymbol{U}_{hh}(r_t \cdot \mathbf{h}_{t-1}) + W_{xh}\mathbf{x}_t) \tag{2.14}$$

$$\mathbf{y}_t = \mathbf{h}_t = (1 - z_t) \cdot \mathbf{h}_{t-1} + z_t \cdot \tilde{\mathbf{h}}_t \tag{2.15}$$

In this thesis we experiment with both LSTM and GRU cells, and perform gradient clipping to mitigate exploding gradients.

Chapter 3

Related Work

The research documented in this thesis was mainly inspired by some papers that exhibit the lack of compositional understanding in current sequence-to-sequence networks. Lake and Baroni (2018) show this problem in the SCAN domain. They show with specialized distributions of the data in training and test sets that regular sequence-to-sequence models are unable to generalize in multiple ways. Loula et al. (2018) reuse this SCAN domain to define even more tasks that analyze this problem in more detail. The premise of both papers can be summarized shortly as: Although current sequence-to-sequence models can generalize almost perfectly when the train and test data are drawn randomly from the same distribution, they are unable to understand and utilize the compositional nature of the task in order to generalize to out-of-distribution data. Even more recently, Bastings et al. (2018) argue that the original SCAN domain itself lacks enough target-side dependencies which might render it too easy and unrealistic. They propose a relatively simple solution to mitigate this problem; They swap the source and target sequences of the domain and call it NACS.

The second domain that inspired the work of this thesis is that of the lookup table compositions (Liška et al., 2018). Arguable, this toy task tests for systematic compositionality in even more isolation as the tasks consists of a rote memorization task performed in a systematic compositional manner. Liška et al. initialized a large amount of models randomly and trained them on this task. Their main finding was that only a very small number of the trained models were able to generalize to out-of-distribution test data, again confirming the hypothesis that sequence-to-sequence models generally do not utilize the compositional nature of the task for generalization purposes. Contrary to the findings of Lake and Baroni and Liška et al., earlier work argues that standard RNNs do already display strong systematicity without any special learning techniques (Brakel and Frank, 2009; Bowman et al., 2015).

This thesis builds upon work on sequence-to-sequence networks (Cho et al., 2014b; Sutskever et al., 2014), and extensions to these models in the form of attention mechanisms (Bahdanau et al., 2014; Luong et al., 2015). The main contribution of this work, which is the Seq2Attn architecture and its analysis, is motivated by our earlier work on Attentive Guidance (Hupkes et al., 2018a), which aims to sparsify the attention vectors and put more focus on it in the network in order to arrive at models with more compositional understanding. Because of the work done on this project and the relatedness, we included a chapter on this topic in this

thesis (Chapter 5). Mi et al. (2016) have implemented something very similar to Attentive Guidance and showed improved performance on a machine translation task. On the task of visual question answering, Gan et al. (2017) and Qiao et al. (2018) have shown a similar approach with attention alignment in image data. We see the difference between their and our work on Attentive Guidance as twofold. Firstly, we distill the contribution of correct Attentive Guidance by using *Oracle Guidance* and secondly, we analyze the contribution of Attentive Guidance specifically in the context of achieving models with compositional understanding. In a whole different order, Vaswani et al. (2017) and Dehghani et al. (2018) also developed models for sequence-to-sequence tasks that put more focus on the attention mechanism. However, they do away completely with sequential processing of the input and output symbols, and instead develop an architecture that consists of successive application of intra-attention.

To sparsify attention vectors for the Seq2Attn model, we use the Gumbel-Softmax Straight-Through estimator (Jang et al., 2016) as activation function. This is used to achieve one-hot vectors without having to resort to learning techniques as reinforcement learning, as this activation function is differentiable. We use one-hot vectors to show and distill the contribution of sparse attention vectors in the Seq2Attn model. For more practical cases, such an activation function could be too restrictive. Luckily, multiple drop-in approaches have been proposed to make attention vectors more sparse, without restricting them to bee truly one-hot. These were originally developed with the intent of improving performance or increasing interpretability of the model. Most notable is the Sparsemax operator, an activation function similar to Softmax, but able to output sparse probabilities (Martins and Astudillo, 2016). Niculae and Blondel (2017) have introduced a framework for sparse and structured attention vectors, that, among others, includes a slightly generalized version of Sparsemax. We view the use and development of such activation function as parallel work.

The idea of using attention as a regularization technique is mainly inspired by Hudson and Manning (2018). They introduce the Memory, Attention and Composition (MAC) cell, consisting of three components. Within one cell, these components are restricted to only communicate with each other using attention mechanisms. This model was designed with the task of visual question reasoning (Johnson et al., 2017) in mind and is therefore designed for a multi-modal input where the model uses a query and a knowledge base to reason. The Seq2Attn model, on the contrary, is designed for unimodal sequence-to-sequence tasks. We accomplish this with a network that shows resemblance to the Pointer Network (Vinyals et al., 2015). Our model can conceptually be thought of as having two components. The first component generates sparse attention and context vectors, which is similar to the Pointer Network. On top of that we add the second component, a decoder that receives solely these context vectors.

Traditional attention mechanisms use the entire encoder states to calculate the attention vectors and context vectors. However, recent work has experimented with dividing the encoder states in two or three parts that fulfill different needs. Mino et al. (2017) and Daniluk et al. (2017) have applied this separation of keys and values. Vaswani et al. (2017) aimed to achieve something similar. However, they did not separate the encoder state vectors in multiple parts. Instead, they fed the entire vector through specialized feed-forward networks to calculate the queries, keys and values independently. In our work, we experiment with using the input sequence embeddings as attention keys and values, in addition to using the encoder states.

Besides efforts to induce more compositional understanding in sequence-to-sequence models, both Attentive Guidance and the Seq2Attn model discussed in this thesis are also aims towards interpretable and explainable AI. For production, analysis and deployment of practical AI that

is safe, reliable and accountable, it is imperative for models to be interpretable or be able to explain their decisions to human operators. We can distinguish two approaches to self-explaining AI, which are nicely summarized by Holzinger et al. (2017). An ante-hoc system is a system that is interpretable by design. This includes linear regression and decision trees. However, in deep learning these approaches are uncommon. Post-hoc approaches are used to explain the decision-making of a certain example after the fact. This includes visualizing the receptive fields of convolutional layers (Simonyan et al., 2013; Zintgraf et al., 2017) or finding input images that maximize the activation of certain units in such a network (Erhan et al., 2009; Yosinski et al., 2015). An approach that sits more in between post-hoc and ante-hoc explanations is the work done by Hendricks et al. (2016), who train, next to an image classifier, a separate deep learning model that outputs discriminative image captions. In this thesis we focus specifically on the recurrent neural network. There have also been some attempts to unfold their inner workings. Most are focused on visualizing and analyzing the activations of the hidden states and memory cells and the weights of the recurrent layers (Strobelt et al., 2018; Li et al., 2016; Karpathy et al., 2016; Tang et al., 2017). Hupkes et al. (2018b) additionally trained diagnostic classifiers to determine whether some kind of information is in some way present in a certain hidden state. Lastly, Bahdanau et al. (2014) already appreciated the interpretable feature of attention mechanisms.[1] In practice, the attention vectors of an attention mechanism may be distributed and spurious, and might not be used extensively by the model. We improve on the interpretability of the encoder-decoder model by putting more stress on the attention mechanism and making the attention vectors more sparse.

Finally, our work shows resemblance to the field of program synthesis and program induction. Synthesized programs can be highly interpretable, are discrete and can potentially generalize to sequences of infinite length. One can thus argue that they can capture systematic compositionality by design. A good overview of the current research status on program synthesis is provided by Kant (2018). Program synthesis is often done using reinforcement learning. This makes it hard to train, often requiring curriculum learning. Program induction approaches often use differentiable memory (Joulin and Mikolov, 2015; Graves et al., 2014) or are heavily supervised to learn an execution trace (Reed and De Freitas, 2015). A Neural GPU approach allows for learning algorithmic patterns that can generalize to longer test sequences (Kaiser and Sutskever, 2015). Other approaches to increasing systematicity in neural networks are learning finite state automate in second-order RNN's (Giles et al., 1992), and hierarchical reinforcement learning, where a more explicit hierarchy of tasks and skills is learned (urgen Schmidhuber, 1990; Sutton et al., 1999; Barto and Mahadevan, 2003; Taylor and Stone, 2009).

[1] Their attention results are nicely visualized by Olah and Carter (2016)

Chapter 4

Testing for Compositionality

Our proposed methods (Chapters 5 and 6) are tested on a set of toy tasks that are designed specifically for the assessment of the amount of compositional understanding in learning agents. These include the *lookup tables* task (Section 4.1), the *symbol rewriting* task (Section 4.2) and the *SCAN* domain (Section 4.3). These task contain training and test sets such that a good performance on the test set is assumed to correlate with a good compositional understanding of the domain. Thus following the authors of these proposed tasks, we use test accuracies as a way to quantify the amount of compositional understanding in a model.

4.1 Lookup Tables

The binary lookup tables task, introduced by Liška et al. (2018), is a simple task that tests the sequential application of lookup tables. Both inputs and outputs of the lookup tables themselves are in the same space. They consist of bitstrings. In our experiment these are 3-bit strings, therefore resulting in $2^3 = 8$ possible inputs to the lookup table functions. Contrary to Liška et al., we present these bitstrings as symbols instead of at character-level. An input sequence $x_1, \ldots, x_n$ consists of one such a bitstring (x_1), followed by one or multiple function names from $\{t_1, t_2, \ldots, t_8\}$. These functions names refer to lookup tables, which are bijective mappings from bitstrings to bitstrings of the same length. Since the lookup tables are bijective mappings with similar input and output space, a multiple of such functions can be applied in arbitrary sequence. An example of this is shown in Fig. 4.1.

Since the functions to be applied are simple lookup tables that require rote memorization, this task tests mostly on the systematic application of these functions. To illustrate, let's say that $t_1(001) = 010$ and $t_2(010) = 111$. Then a training example could be 001 t_1 $t_2 \rightarrow$ 001 010 111. First t_1 has to be applied on the input bitstring, the intermediate output (010) has to be stored (and outputted), and t_2 has to be applied on the intermediate output. Note that we use *Polish* notation to allow an incremental application of the functions, instead of forcing the network to memorize each table and reverse the order of application. Similar to Liška et al., we include the intermediate function outputs (010) in the target output sequence. Contrary to their work, we also include the original input bitstring (001) in the output sequence. This we call the *copy-step*.

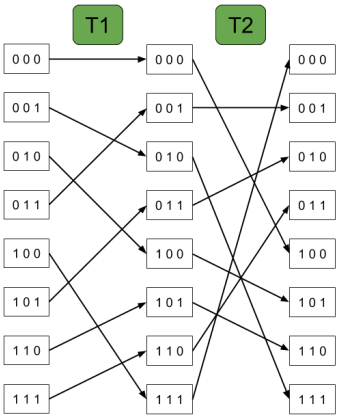


Figure 4.1: Example of all input-output pairs of the composition $t_1 \ t_2$.

We do this such that the input and output sequence are of equal length, and that there exists a direct semantic alignment between each input and output symbol. It will be clear in Chapter 5 why this is useful.

In our experiments we randomly create 8 lookup tables. Since there are 8 possible inputs for each table, there are 64 *atomic table* applications. These all occur in the training set, such that each function can be fully learned. The training set additionally contains some length-two compositions, meaning that two lookup tables are applied in succession on an input bitstring. These are provided such that the learning agent can learn to apply sequential function application. Some of the length-two compositions are reserved for testing purposes. Liška et al. used only one testing condition (corresponding to our held-out inputs condition, explained below). Since our methods showed impressive performance on this condition, we additionally created increasingly harder conditions, which also allow for a more fine-grained analysis of the strengths and weakness of certain models.

We reserve one test set that contains all length-two compositions containing only t_7 and t_8 (*new compositions*), and one test set that contains compositions of which only one function is in $\{t_7, t_8\}$ (*held-out tables*). Of the remaining length-two compositions, that include only functions in $\{t_1, \ldots, t_6\}$, 8 randomly selected compositions are held out from the training set completely, which form the *held-out compositions* test set. From the remaining training set, we remove 2 of the 8 inputs for each composition independently to form the *held-out inputs* set. None of the training and test sets are overlapping. A validation set is formed by reserving a small set of the *held-out inputs*. Figure 4.2 shows a comprehensive visualization of the generation of data sets.

4.2 Symbol Rewriting

A second test we consider is the symbol rewriting task introduced by Weber et al. (2018). The goal for the learning agent here is to produce three output symbols for each input symbol in the same order as they are presented in the input. Each input symbol represents a vocabulary from which three output symbols should be sampled without replacement. Let's take, as a simplified example, the input $A \ B$. The vocabularies associated with these input symbols are

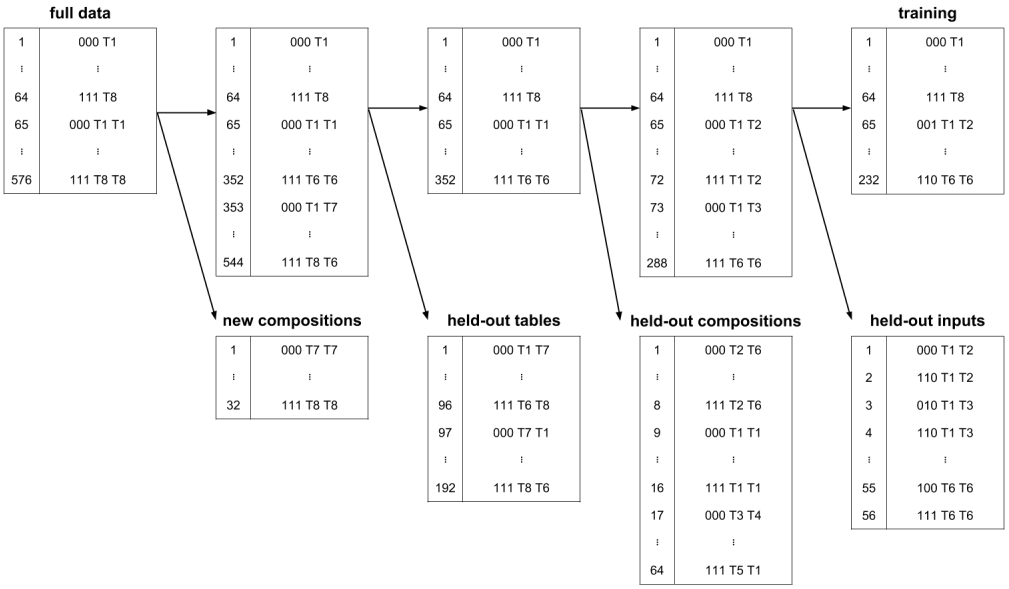


Figure 4.2: Generation of all sets in the lookup tables task. The full data set consists of 576 examples; 64 atomic function applications and 512 length-two compositions. The bottom four tables show how some length-two compositions are reserved for testing. The final train set contains all unary compositions and the remaining length-two compositions. Of the 56 examples in *held-out inputs*, 16 are reserved for validation.

$\{a_1, a_2, a_3, a_4\}$ and $\{b_1, b_2, b_3, b_4\}$ respectively. One of the multiple correct outputs would thus be $a_2\ a_3\ a_4\ b_3\ b_1\ b_2$. However, to add more stochasticity to the outputs, Weber et al. allow two possible values for each output, such that each output $\hat{y}_i$ can take on either $\hat{y}_{i1}$ or $\hat{y}_{i2}$. Thus a correct output would be $a_{21}\ a_{32}\ a_{41}\ b_{32}\ b_{12}\ b_{21}$.

For one input sequence, many output sequences can be correct. This is by design, such that an agent cannot resort to pure memorization. During training, one of the possible outputs is presented at a time and backpropagated to update the parameters. For evaluation, we use an accuracy metric that counts an output to be correct if it lies in the space of all correct outputs. For consistency with the other tasks, we will simply refer to this metric as the (sequence) accuracy.

The original training set consists of 100.000 randomly sampled input-output pairs, with input lengths within [5–10]. There are no repetitions of input symbols within one single input sequence. There are four test sets, each with 2.000 examples. The *standard* set is sampled from the same distribution as the training set. The *short* set contains shorter inputs within the range [1–4], *long* within range [11–15], and *repeat* within range [5–10] but with repetition of input symbols allowed.

Weber et al. used a validation set containing a mixture of all test sets for choosing the hyper-parameters and early stopping. Since we want to show the generalizability of the model to data it has not seen during training, we also created a different validation set. Of the original training set of 100.000 examples, we have reserved 10% for validation, bringing the number of training examples down to 90.000, which we will call *standard validation*. The original validation set, we

$C \rightarrow S$ and S	$V \rightarrow D$[1] opposite D[2]	$D \rightarrow$ turn left
$C \rightarrow S$ after S	$V \rightarrow D$[1] around D[2]	$D \rightarrow$ turn right
$C \rightarrow S$	$V \rightarrow D$	$U \rightarrow$ walk
$S \rightarrow V$ twice	$V \rightarrow U$	$U \rightarrow$ look
$S \rightarrow V$ thrice	$D \rightarrow U$ left	$U \rightarrow$ run
$S \rightarrow V$	$D \rightarrow U$ right	$U \rightarrow$ jump

Table 4.1: Context-free grammar with which commands of domain C are created in the SCAN domain. Courtesy of Lake and Baroni (2018).

will refer to as *mix validation*.

This task is set up to mimic the alignment and translation properties of natural language in a much more controlled environment, and to test the ability to generalize to test sets that are sampled from different distributions than the training set. Because of the introduced stochasticity in the outputs, an optimal learning agent should not memorize specific examples, but should learn to do local, stochastic translation of the input symbols in the order at which they are presented, while following the appropriate syntactical rules.

4.3 SCAN

As a third task that tests for compositionality we used the SCAN domain. It was introduced by Lake and Baroni (2018) and can be seen as a (and abbreviates for) **S**implified version of the **C**omm**AI** **N**avigation task that is learnable in a supervised sequence-to-sequence setting. The CommAI environment was introduced earlier by Mikolov et al. (2016). Input sequences are commands composed of a small set of predefined atomic commands, modifiers and conjunctions. An example input is *jump after walk left twice*, where the learning agent has to (mentally) perform these actions in a 2-dimensional grid and output the sequence of actions it takes (*LTURN WALK LTURN WALK JUMP*).

There are four command primitives in the original domain. These include *jump*, *walk*, *run* and *look*, which are translated in the actions *JUMP*, *WALK*, *RUN* and *LOOK* respectively. Additionally there are some modifiers and conjunctions. The language is defined such that there can be no ambiguity about the scope of modifiers and conjunctions. The grammar with which an expression C can be constructed is listed in Table 4.1. The interpretation of such commands are detailed in Table 4.2

The authors mention three possible experiments.[1] However, in later work, Loula et al. (2018) define another three experiments in this domain, as they hypothesize that the earlier experiments might test for something different than compositional understanding. We will shortly summarize the experiments, which we will henceforth call *SCAN experiments 1-6*.

- *SCAN experiment 1:* The total number of possible commands in the SCAN domain (20.910) was split randomly in a 80-20 distribution for training and testing. With this, Lake and

[1] An updated version of this paper now contains a fourth experiment, similar to their third experiment, but with a more realistic translation data set. We do not cover this.

[[walk]] → WALK	[[turn opposite left]] → LTURN LTURN
[[look]] → LOOK	[[turn opposite right]] → RTURN RTURN
[[run]] → RUN	[[u opposite left]] → [[turn opposite left]] [[u]]
[[jump]] → JUMP	[[u opposite right]] → [[turn opposite right]] [[u]]
[[turn left]] → LTURN	[[turn around left]] → LTURN LTURN LTURN LTURN
[[turn right]] → RTURN	[[turn around right]] → RTURN RTURN RTURN RTURN
[[u left]] → LTURN [[u]]	[[u around left]] → LTURN [[u]] LTURN [[u]] LTURN [[u]] LTURN [[u]]
[[u right]] → RTURN [[u]]	[[u around right]] → RTURN [[u]] RTURN [[u]] RTURN [[u]] RTURN [[u]]
[[x twice]] → [[x]] [[x]]	[[x_1 and x_2]] → [[x_1]] [[x_2]]
[[x thrice]] → [[x]] [[x]] [[x]]	[[x_1 after x_2]] → [[x_2]] [[x_1]]

Table 4.2: Interpretation of commands in the SCAN domain into actions that should be performed by the agent. Variable u may be replaced by any symbol in domain U of Table 4.1 (*walk*, *look*, *run*, *jump*). Variable x may be replaced by expressions in all domains but C. Courtesy of Lake and Baroni (2018).

Baroni show that vanilla Seq2Seq models can achieve near-perfect performance when the test and train set are sampled from the same distribution.

- *SCAN experiment 2:* To test the generalization performance to longer sequences, the training set is constructed by taking all examples of which the output has a length of up to 22 actions, and using longer sequences (23-48) for testing. With a mean sequence accuracy of at most 20.8% they consider the models to fail on this task.

- *SCAN experiment 3:* In this experiment the command *jump* is only seen atomically, but not in compositions with modifiers and conjunctions. The test set contains all examples in which *jump* is used in compositions. This tests the ability to understand that *jump* has to be treated similarly to the other primitive commands, and must thus be used similarly in compositions. To accommodate for the class imbalance in the training set, the *jump* example is repeated such that it covers roughly 10% of the training set. The best observed performance on this task was only 1.2% sequence accuracy.

- *SCAN experiment 4:* Four different train-test distributions are created. Each is associated with a certain template. The used templates are *jump around right*, ***primitive*** *right*, ***primitive*** *opposite right* and ***primitive*** *around right*, where ***primitive*** is a placeholder for all the four primitive commands. Each of the train-test distributions is created by adding all commands containing the templates to their respective test set, and using the rest for training sets. The task is designed such that it tests for the ability to recombine words in unseen contexts (similar to *SCAN experiment 3*), but giving the model the ability to learn correct word embeddings for each of the tested words.

- *SCAN experiment 5:* In *SCAN experiment 4*, a very low test accuracy (2.46%) was reported when no examples of the template ***primitive*** *around right* occurred in the training set (*0 fillers*), while models would score extremely high (98.43%) when *3 fillers* of this template were presented during training and were tested on only 1 (*jump around right*). Loula et al. thus also created the *1 filler* and *2 fillers* experiments. The *0 fillers* experiment thus contains no examples of the template ***primitive*** *around right* in the training set, and now tests for only examples containing *jump around right*. The *1 filler* experiment keeps the

same test set, but now also adds commands containing *look around right* to the training set. The *2 fillers* and *3 fillers* experiments increasingly also add commands containing *walk around right* and *run around right*.

- *SCAN experiment 6:* The test conditions were even further refined from *SCAN experiment 5* since a huge performance gap was observed between the *0 fillers* and *1 filler* experiment. The training set of the latter experiment contains 1.100 examples more than the *0 fillers* experiment. For a more fine-grained analysis, Loula et al. create additional training sets by taking the one from the *0 fillers* experiment and randomly adding respectively 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 and 1024 examples containing the template *look around right*.

It must be noted that all of the 6 experiments lack a validation set. For experiment 1, we created such a set ourselves by reserving approximately 10% of the training set for this.

Chapter 5

Attentive Guidance

In this chapter we will discuss the motivation for and results of *Attentive Guidance* (AG), a learning technique that aims to induce a compositional bias in regular sequence-to-sequence models. The results and findings of this technique is what later on motivated us to come up with a new sequence-to-sequence architecture (*Seq2Attn*) which is described in Chapter 6.

Firstly we will motivate why we need such a new learning technique (Section 5.1), followed by the details of how this technique works (Section 5.2). We will share the results on several tasks (Section 5.3) and conclude and discuss our findings (Section 5.4).

5.1 Motivation

We mentioned earlier in this thesis that ANNs are in theory capable of representing any function or algorithm (Section 1.1). However, in Section 2.1, we showed by an example that from seeing only the training data and with *learning by example*, it is theoretically impossible for the model to infer how this task should be solved. To encourage a model to converge on a more compositional solution, we thus might have to induce biases or priors to the model on a different level. With Attentive Guidance we try to induce a bias for the type of solutions that a sequence-to-sequence model will converge on. We add additional information on how the problem should be solved by guiding its attention mechanism such that a model will converge on solutions with more compositional understanding using standard learning techniques.

To illustrate the intended effect of Attentive Guidance, let us draw an analogy. Consider a teacher teaching a child to read. Most certified teachers would not shove this child into a dark closet with the complete works of William Shakespeare in their hands, only periodically testing whether they have improved. Instead, a teacher might sit next to the child and help read it letter-for-letter, word-for-word, using its finger to point at the individual letters and teaching them separately. This allows the child to learn the appropriate semantic meaning and phones of the relatively small vocabulary which is the Latin alphabet. Once the child has acquired this knowledge, the teacher could show how these individual characters are combined into words, and how words are combined into sentences. This teaches the rules of how to combine multiple

individual letters and words into more complex constructions and how the semantic meaning and sound of such a construction depends on the combination of the smaller parts.

This method of teaching, by pointing out smaller parts explicitly and how they combine into larger parts can be seen in many examples of learning. Before (dis)assembling a car engine, a mechanic should first be familiar with the individual components and how together they form a working engine. A driving instructor's most important job might be to teach its students what to focus on. And without proper guidance, an academic student might get lost in the abundance of information. In other words, this method of teaching, by pointing out individual parts and how they are combined, teach a compositional understanding which should allow for generalization to out-of-distribution data. For many tasks, including language understanding, such a compositional solution is what we want a model to find. We thus take inspiration from this method of teaching and try to guide an ANN in a similar way.

Similar to how a human teacher could teach a child, we also try to teach an ANN by pointing out individual alignments. Specifically, we draw our attention to RNNs in an encoder-decoder structure, as these are very suited for language-like tasks and because they can easily be augmented with an attention mechanism. On these attention mechanisms, we apply a learning technique we call *Attentive Guidance* (AG). By aligning output symbols to input symbols during training, we try to make models find the relevant correlations between these two. This should make the training process more resemblant of the described manner in which humans may teach, and it should provide the right biases to the model to converge on a compositional solution. Instead of only telling a model *what* to learn by providing input-output examples, we also tell the model *how* it should learn by providing an alignment between input and output symbols. Although this requires to augment the training data with these alignments, we see impressive results on a variety of toy tasks, which motivate to continue research in this direction.

5.2 Implementation

If one is familiar with sequence-to-sequence models (Cho et al., 2014b; Sutskever et al., 2014) and extensions of these models with attention mechanisms (Bahdanau et al., 2014; Luong et al., 2015), Attentive Guidance is a simple extension to such models. It does not require changes to the architecture and is also not suited solely for the architecture we will use. In short, Attentive Guidance utilizes the existing attention mechanism of a sequence-to-sequence model to either encourage or force specific, sparse alignment between output symbols and input symbols. It does so by minimizing an additional loss term that encourages the model to generate attention vectors like the provided ground-truth vectors (*Learned Guidance*). Additionally, we also do experiments in which the ground-truth attention vectors are directly used in the model (*Oracle Guidance*). To encourage the use of the attention mechanism, we also experiment with a variant of the pre-rnn method (Section 2.2.2) called *full-context-pre-rnn*.

To teach a model what it should attend to, we should provide this information to the model. Table 5.1 shows examples of how such information could be provided in the data set. We have an input and output sequence consisting of multiple symbols, which we number with indices left-to-right, starting with 0. The third column shows the attention patterns that are provided to the model. The number of attention targets always equals the number of output targets. For each output target, it tells what input symbol the model should align or attend to. These

Input	Output	Attention targets
000 t_2 t_1	000 100 111	0 1 2
A B	a_{21} a_{32} a_{41} b_{32} b_{12} b_{21}	0 0 0 1 1 1
te video	i see you	1 1 0

Table 5.1: Three artificial examples of how attention targets are represented in the data sets. The first example comes from the Lookup tables task where we always have a diagonal attention pattern, the second from the symbol rewriting task, and the third example is an example of a language translation task.

attention targets can be interpreted as index representations of one-hot vectors with the same dimensionality as the input sequence.[1] Looking at the third example, this thus means that to produce *i* and *see*, a model should attend (fully) on the second input symbol *video*. To produce *you*, it must use the first input symbol *te*.

Our current research is limited to using one-hot vectors. One can imagine that some tasks require more flexible alignments, using multi-hot vectors and possibly different weights for each alignment. Such extensions are considered trivial. One would simply have to choose a more explicit representation of the attention vector in the data set. How we produce these ground-truth attention data for all considered data sets is explained in Section 5.3.

In the next two sections we will more technically explain the implementation of Learned Guidance and Oracle Guidance. For both methods, the attention patterns should be provided in the training set for all input-output examples.[2] The main focus of our research on this topic is the Learned Guidance. With this method, attention targets only have to be provided at training time. From this data the model is expected to learn to reproduce these patterns and generalize to test sets without attention targets. However, we observed this generalization to be difficult for certain test sets. To disentangle this difficult generalization from the benefits that correct guidance might provide, we also do experiments with Oracle Guidance. With this, the attention patterns are directly used in the model and thus not learned, which requires this data to also be present at inference time. We acknowledge, however, that it might not be realistic to assume that this data is always available.

Acquisition of high-quality, annotated data is generally a big problem in deep learning. Most architectures are known to be data-hungry. They require lots of data to learn a function without significant overfitting (Halevy et al., 2009). To obtain attention pattern data as shown in Table 5.1 might thus be even harder. For some data sets, the optimal attention pattern that encourages compositional understanding may be inferred or computed algorithmically from the input-output examples. For some tasks, including language-like tasks, this is not as simple. The generation of data for such tasks could be done by human annotators. This severely limits the amount of data you can produce as the process of data-generation is time consuming, expensive and error-prone. Other approaches also exist that are trained to replicate human annotations (Ittycheriah and Roukos, 2005). Additionally, other alignment models exist that are trained unsupervised (Brown et al., 1993; Och and Ney, 2000), which produce suboptimal alignments. But

[1] We will use the same notation to refer to the index of the attention target as well to the actual one-hot vector representation. This should be inferred from the context.

[2] Experiments have actually shown that using Learned Guidance on only a small subset of the training set, and training on all other examples regularly, already greatly improves performance.

more importantly: this data is not available at inference time. This makes that Oracle Guidance is not a realistic method to be deployed. We only study this to show the upper bound of what could be achieved with Attentive Guidance. Learned Guidance could be a realistic method in these situations as only (a small part of) the training set would have to be annotated.

5.2.1 Learned Guidance

In Section 2.2 we explained how a vanilla sequence-to-sequence network operates. This also includes the technicalities of an additional attention mechanism. Traditionally, this attention mechanism produces a context vector for each decoder step. This is a weighted average of the RNN states of the encoder. The weights $\mathbf{a}_t(s)$ form a proper probability distribution for decoder step t, and are calculated as in Eqs. 2.1 and 2.2. We will call this the *attention vector*, as it is a vector that indicates to what degree the encoder states should be attended to. As we want the model to produce here the right alignment, ideally we want the attention vector to be the same as the provided attention target for the corresponding output symbol. We therefore add a weighted cross-entropy loss term to the total loss to minimize this difference

$$\mathcal{L} = \mathcal{L}_{output} + \frac{\lambda_{attn}}{M} \sum_{i=i}^{M} \sum_{j=i}^{N} - \log \mathbf{a}_i(j) \cdot \delta_{i,j} \tag{5.1}$$

where $\mathcal{L}_{output}$ is the original loss, λ_{attn} is the weight of the additional loss term, δ is the Kronecker delta function that is only 1 for aligned symbols, and N and M are the lengths of the input and output sequences respectively. Figure 5.1 shows an illustration of this model.

When weighted properly, a model would thus learn to model the correct output while also modeling the correct attention alignments. The attention vector is encouraged to mimic the predefined behavior of the attention patterns that are present in the training set. In this way, we hypothesize that a model learns to utilize the right correlation between input and output symbols, learns the individual meaning of each input symbol, which should guide the model in its search of the loss function to converge on a solution that exhibits a compositional understanding. In all our experiments we used $\lambda_{attn} = 1$.

5.2.2 Oracle Guidance

We hypothesize that a model trained with Learned Guidance shows more compositional understanding and would thus be better at generalizing to unseen data, sampled from different distributions than the training set, but which could be explained by the compositional nature of the task. However, this is assuming that the generation of attention vectors itself generalizes perfectly to unseen data. We have experienced this not always to be the case. To better examine the contributions of correct guidance in the search for compositional models, we also do experiments with Oracle Guidance. A model with Oracle Guidance does not calculate the attention vector on its own as in Eq. 2.1. Instead we simply replace this by the one-hot vector that is provided in the data set. This helps to distill the contribution of correct guidance in finding a compositional model, from the difficulties that the model might have with generalizing the attention vector

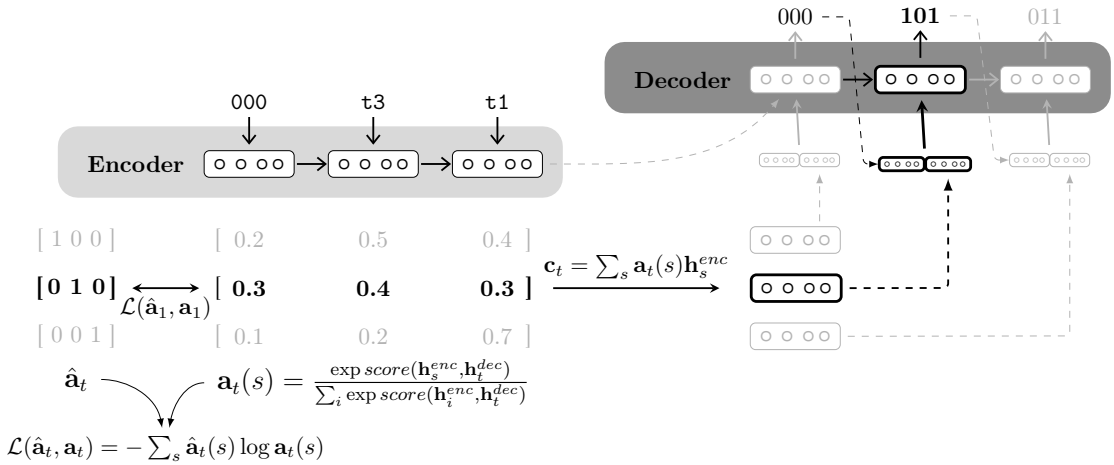


Figure 5.1: An illustration of the Learned Guidance model, created by Dieuwke Hupkes (Hupkes et al., 2018a). At $t = 2$ the decoder calculates weights [0.3 0.4 0.3]. These are the weights it uses for the context vector; A weighted average over the encoder states. The provided ground-truth attention vector is [0 1 0]. The cross-entropy loss between these probability vectors is backpropagated, such that the model learns to attend to t_3.

generation process to out-of-distribution data. However, this requires the attention vectors to be presented to the model at inference time, and is thus not a realistic setting.

We could formalize this method as replacing the alignment function (Eq. 2.1) with

$$\mathbf{a}_t(s) = \delta_{t,s} \tag{5.2}$$

where again, δ is the Kronecker delta function that is only 1 for aligned symbols in the annotated data set.

5.2.3 Full Context Focus

Both with Learned Guidance and Oracle Guidance we hope to make a model find a more compositional model by showing it informative, compositional attention patterns. We introduce a minor adjustment to the pre-rnn attention method with the intention to make sure that the model can not easily ignore this information. For some tasks we found this adjustment helpful, while for others it was not. This is therefore tuned with a hyper-parameter search.

As explained earlier (Section 2.2.2), the context vector $\mathbf{c}_t$ is calculated on basis of the previous decoder state $\mathbf{h}_{t-1}^{dec}$ in the pre-rnn setting. The context vector is concatenated to the input of the decoder as $\bar{\mathbf{x}}_t^{dec} = [\mathbf{x}_t^{dec}; \mathbf{c}_t]$ such that its information may be incorporated by the current and all subsequent decoder states. With *full-context-pre-rnn*, we first feed $\bar{\mathbf{x}}_t^{dec}$ through a learnable linear layer to reduce its dimensionality to the same as $\mathbf{c}_t$. Then, we apply an element-wise multiplication to increase the relative importance of the context vector.

Hyper-parameter	Lookup Tables	Symbol Rewriting	SCAN
Batch size	1	128	128
Optimizer	Adam, default parameters		
RNN cell	{LSTM, GRU}		LSTM
Embedding size	{16, 32, 64, 128, 256, 512}	{32, 64}	200
RNN size	{16, 32, 64, 128, 256, 512}	{64, 128, 256}	200
Dropout	0		0.5
Teacher forcing	0	1	0.5
Attention method	{post-rnn, pre-rnn, full-context-pre-rnn}		pre-rnn
Alignment method	{dot, concat, mlp}		mlp

Table 5.2: Fixed and tuned hyper-parameters for baseline and Learned and Oracle Guidance models on all considered tasks.

$$\bar{\mathbf{x}}_t^{dec} = \mathbf{c}_t \odot \boldsymbol{W}_{fc}^{\intercal}[\mathbf{x}_t^{dec}; \mathbf{c}_t] \tag{5.3}$$

5.3 Experiments

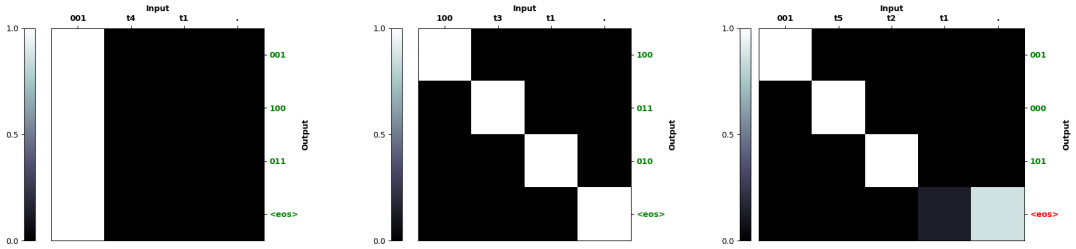
We compare the performance of the Attentive Guidance models on three different tasks, which to some extent are designed to test for compositional understanding. These tasks and data sets are describer earlier in Chapter 4. We compare three different models; A baseline model which is a commonly used sequence-to-sequence model augmented with attention, and similar models with Learned Guidance and Oracle Guidance. For each task, we perform a hyper-parameter grid search on a small amount of parameters for the baseline and Learned Guidance models. For the Oracle Guidance models we use the same hyper-parameters as for the Learned Guidance models. All relevant hyper-parameters are summarized in Table 5.2. The optimal hyper-parameters are mentioned for each task separately.

5.3.1 Lookup Tables

Attention Targets

The ground-truth attention targets for this task are straightforward. We believe that for this task there exists only one solution that can truly be called compositional. The first output symbol should always be the same as the first input symbol. We call this the copy-step. The model should therefore attend to the first input to produce this output. Next, it should apply the first table lookup, thus the model has to attend to the first table. All other tables are irrelevant at this point. This continues until the end of the sequence; For each output, the model should attend to the next input. Finally to produce the EOS symbol, the model should also look at the EOS symbol in the input sequence, which we denote with a full stop. If both the input and output sequence are of length N, the attention targets should be $0, 1, \ldots, N$.[3] We also call this a diagonal pattern as the stacked one-hot vectors of the attention targets form an identity matrix.

[3]This results in $N+1$ attention vectors, but we must also account for the attention target of the EOS symbol.



(a) (b) (c)

Figure 5.2: (a) Uninformative attention pattern found by a baseline model on the validation set of lookup tables. (b) Correctly learned guidance on the validation set of the lookup tables. (c) Wrongly Learned Guidance on length-three compositions of the lookup tables makes that the model predicts the EOS symbol too early.

Optimal Models

We found the baseline Seq2Seq with an embedding size of 128 and GRU size of 512 to be best performing. The Learned Guidance and Oracle Guidance models performed best with embedding and GRU sizes of 16 and 512 respectively. For Attentive Guidance models, we see a clear trend that larger models converge faster and that they perform better on both the training data, as well as the testing data. Seq2Seq models, however, are more prone to overfitting with larger models. For baseline models the pre-rnn method was optimal, while full-context-pre-rnn delivered best results for both Learned Guidance and Oracle Guidance. The mlp alignment method was optimal in all cases.

We report average sequence accuracies over 10 runs with random weight initializations.

Results

Figure 5.3 shows the results of all three models on four different test sets. This figure shows clearly that the baseline model is unable to properly generalize to data drawn from different distributions. The Learned Guidance model performs significantly better than the baseline model on data that is of similar length as the data in the training set. It is able to almost perfectly solve the held-out inputs case, which Liška et al. (2018) originally studied, but also additional data sets which are arguably more complex. Performance on held-out tables and new compositions is slightly lower. These are the data sets that contain tables t_7 and t_8 which are only seen atomically in the training set and thus not in compositions. The drop in performance on these sets might thus partly be explained by little capabilities of generalizing to longer length sequences.

Longer Lengths

The Learned Guidance models have surprisingly good zero-shot performance on instances of the same length as in the training set. Both the baseline and Learned Guidance models are, however, virtually unable to generalize to longer lengths. The Oracle Guidance model is the only model

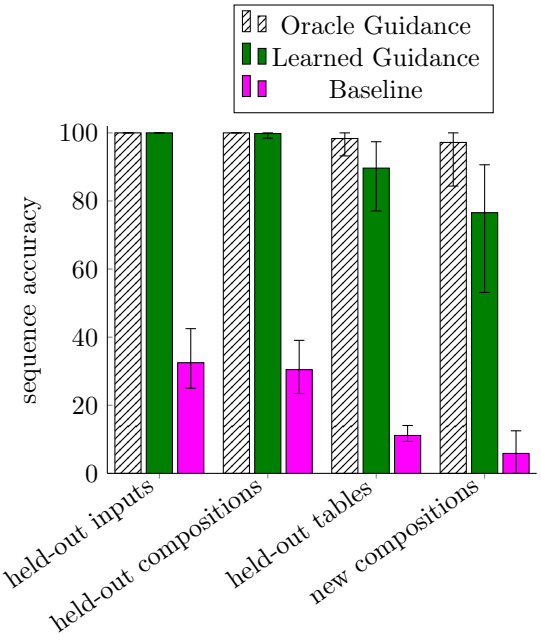


Figure 5.3: Average accuracies on the lookup tables task. Error bars indicate the best and worst performing models. Oracle Guidance results are included but shaded to indicate that it is not a viable method for real tasks.

capable of generalizing to longer lengths as can be seen in Fig. 5.4a. As soon as we increase the length of the sequence with only one composition, the performance drops dramatically. This inability to generalize to longer sequences is a notorious problem in deep learning (Cho et al., 2014b; Lake and Baroni, 2018). We see that the Oracle Guidance models are able to generalize to sequences of significantly longer lengths. Furthermore, if we take a model that was trained using Learned Guidance, but at inference time replace the calculated attention vector with the Oracle Guidance attention vector, we see similar generalization capabilities. This indicates that both the Learned and Oracle Guidance models have converged on a solution in which they are able to solve the task compositionally, given that the models receive the correct Attentive Guidance. The reason that the Learned Guidance models cannot generalize to longer sequences is because of its inability to generate the correct attention patterns. In an additional experiment we tried to assess whether this is a problem of badly designed data sets. The training set consists of composition of up to only two tables, which might be too limited for a learning agent to infer that it should generalize to longer lengths. We tested this by creating training sets of compositions up to length five. This did not change the observed behavior.

Overfitting

Historically, most deep learning models that are large enough to solve a task tend to overfit on the train data at some point. To some extent, this does not seem to be the case for both Learned Guidance and Oracle Guidance for this task. Both models are trained for a significantly longer period, and the loss on the validation set is shown in Fig. 5.4b. We argue that Attentive

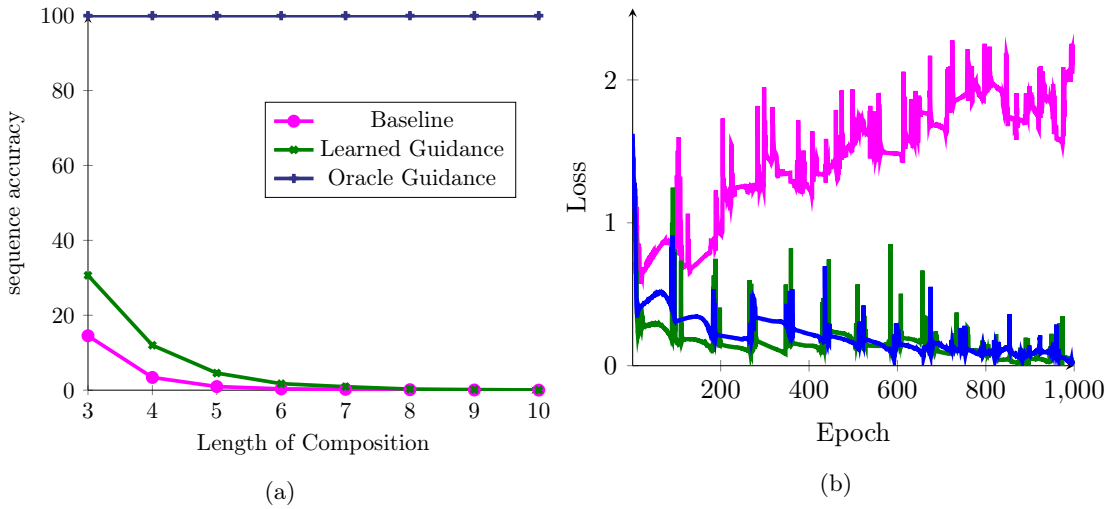


Figure 5.4: (a) Average sequence accuracy on longer held-out input compositions in the lookup tables task. (b) Loss development on the validation set of the lookup tables task, showing no indication of overfitting for the Learned and Oracle Guidance models. Spikes might be explained by the use of mini-batches or by the second-moment estimate of the Adam optimizer (Kingma and Ba, 2014) becoming too low with respect to the first-order estimate.

Guidance guides the models into converging on a compositional solution that over time not only fits the training set, but all data that can be explained by the same rules.

5.3.2 Symbol Rewriting

Attention Targets

As for the lookup tables task, the generation of attention targets for the symbol rewriting task is straightforward. We have a diagonal-like pattern again. The first three output symbols depend solely on the first input symbol. Thus, the first three steps all attend to the first input symbol. The second triplet of outputs should attend to the second input symbol, and so on. Finally, to produce the output EOS symbol, the model should attend to the input EOS symbol, which we denote with a full stop.

Optimal Models

We see again that the Learned and Oracle Guidance models benefit in greater effect from larger model sizes. We use embedding and LSTM sizes of 32 and 256 respectively. The baseline performs best with sizes of 64 and 64. However, since Weber et al. (2018) report a high variance in model performance on this task, we also include a second baseline model with the same model size as the Learned Guidance model, which allows for fairer comparison. Both baselines and Attentive Guidance models use pre-rnn attention and mlp alignment. We will therefore henceforth use this

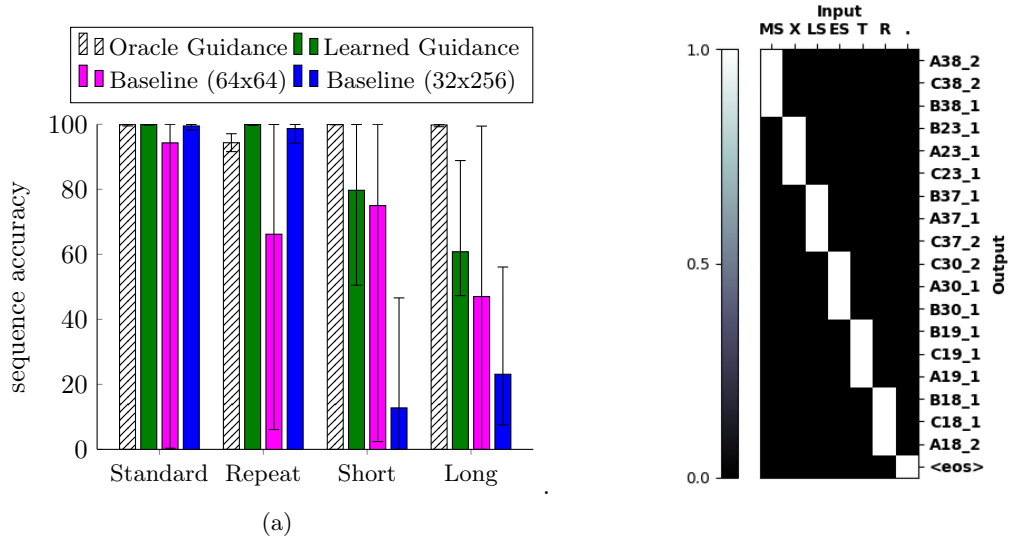


Figure 5.5: (a) Average sequence accuracies on the four test sets of the symbol rewriting task. The error bars indicate the best and worst performing models in the set of 50. (b) Correctly learned attention pattern on the symbol rewriting task.

attention mechanism as default. Because of the expected high variance among models, we train each model 50 times with the same configuration and look at the average performance, similar to Weber et al.. The best model is determined by looking at the average accuracy on the *mix validation* set. As explained in Section 4.2, we use a slightly different accuracy metric adapted to this specific domain.

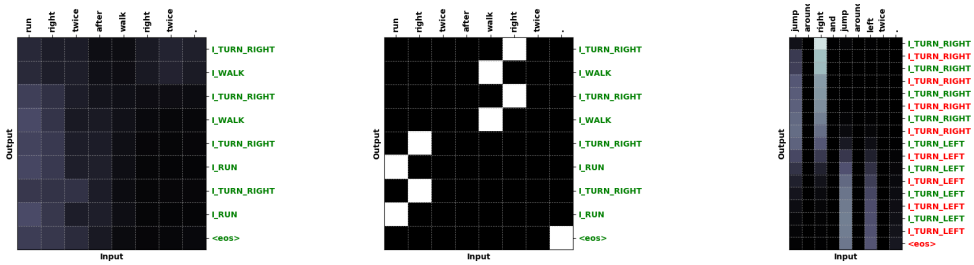
Results

The results on all four test sets are depicted in Fig. 5.5a. These results are in line with what we concluded from the lookup tables task. The Learned Guidance models are able to generalize almost perfectly to new instances of the same length as the training set. For shorter and longer data, it is more robust and stable than the baseline. On average, it slightly outperforms the baseline models, but again is unable to perfectly fit data of different lengths.

5.3.3 SCAN

Attention Targets

We also provide a brief analysis of Attentive Guidance on the SCAN domain. For this task it is less clear what attention pattern could reflect a compositional solution. In our tests we have chosen for the following attention pattern, but several others could be thought of.



(a) (b) (c)

Figure 5.6: These plots show typical attention patterns for baseline and Learned Guidance models on the test sets of SCAN experiments 1 and 2. (a) Uninformative attention pattern found by a baseline model on SCAN experiment 1. (b) Correctly Learned Guidance on SCAN experiment 1. (c) Wrongly Learned Guidance on SCAN experiment 2.

The output vocabulary in this task consists of only the following symbols: *LTURN*, *RTURN*, *WALK*, *LOOK*, *RUN*, *JUMP* and the *EOS* symbol. For each of these symbols, we encourage the Attentive Guidance models to attend to encoder states associated with the primitive commands that produced them. That is, for the decoder step that produces *LTURN*, we make it attend to the relevant input symbol *left*. All other output symbols (*RTURN*, *WALK*, *LOOK*, *RUN*, *JUMP* and the *EOS* symbol) we align to the following input symbols respectively: *right*, *walk*, *look*, *run*, *jump* and ".", which is the EOS symbol used for the input sequences.

Optimal Models

For the baseline, Learned Guidance and Oracle Guidance models we use the same hyper-parameters as used by Lake and Baroni (2018) for their *overall-best* model. This is an encoder-decoder model with 2-layer LSTMs and a 0.5 dropout rate. All embedding and LSTM layers have 200 hidden units. Their original model used no attention mechanism. Since Attentive Guidance is implemented on the attention mechanism, we extend the baseline model with pre-rnn attention with the mlp alignment method. All models were ran 10 times on SCAN experiments 1-3. The original experiments did not include validation sets. We did therefore not perform early stopping based on validation loss, as we did in all other experiments. Instead, we ran the models for a predetermined number of epochs (30) and selected the parameters sets that would achieve the lowest loss on the respective training sets.

Results

We see that Oracle Guidance models show significant improvements over the baseline models on SCAN experiment 2 (Fig. 5.7). This indicates that with a more informative attention pattern the task is much easier to solve. However, producing this attention pattern is certainly not as easy, as indicated by the discrepancy between Learned Guidance and Oracle Guidance models. These results again show that when an encoder-decoder model is provided with informative attentive guidance, solving the task becomes much easier. However, modeling this attention pattern could have the same complexity as solving the original task.

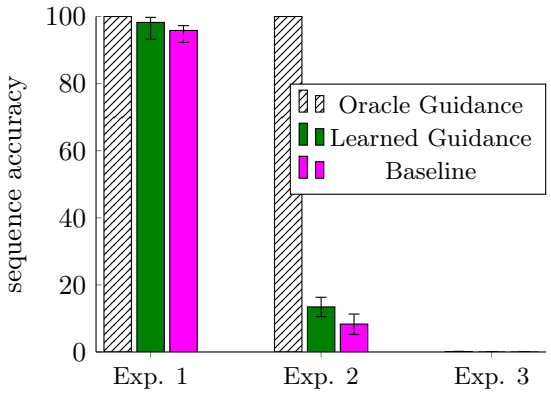


Figure 5.7: Mean sequence accuracies of the baseline, Learned Guidance and Oracle Guidance models on SCAN experiments 1-3. Error bars indicate the best and worse performing models.

All models achieve near-perfect scores on experiment 1. On experiment 2, the baseline model achieves lower scores than reported by Lake and Baroni (8.31%). The Learned Guidance models achieve a slightly higher mean sequence accuracy (13.43%), but still do not solve the task completely. However, when the correct guidance is always provided in the Oracle Guidance models, a perfect accuracy (100%) is achieved. On experiment 3, the baseline and Learned Guidance models fail completely. With Oracle Guidance, the mean sequence accuracy reaches only 0.06%. As suggested by Loula et al. (2018), this task could be badly designed. It tests not for compositional understanding as the models are not in any way given evidence that the *jump* command is similar to the other primitive command *look*, *run* and *walk*.

Results in the SCAN domain are inconclusive. Although a significant rise in performance is observed when a Seq2Seq model receives Oracle Guidance, learning to reproduce this guidance is a hard task to solve. Furthermore, we are unsure about what attention annotation would be optimal for inducing a compositional solution. We have chosen for a direct alignment to atomic commands in the input sequences, but other attention patterns could be thought of.

5.4 Conclusion

We have implemented a simple and effective way to guide a sequence-to-sequence model in its optimal attention pattern. For most discussed tasks, this results in greatly improved performance on out-of-distribution data. These tasks are designed to test for compositional understanding and execution. We therefore conclude that for these specific tasks, a model with optimal attention patterns can solve the task in the correct manner.

We see three objections that can be made against the effectiveness of the proposed methods.
Firstly, when we compare the Learned Guidance models to the baseline models, we do not see greatly improved performance on data that is of different length than in the training set. We could say that, given the correct guidance, the model is compositional, but it is unable to generate this compositional guidance itself for out-of-distribution data.
Secondly, we question the effectiveness on more complex tasks like neural machine translation

(NMT). The current implementation only allows for one-hot attention vectors which might be unrealistic for the general case. It might be the case that the model needs to incorporate a larger context to be able to produce correct output.
Thirdly, we see that the annotation of data with attention alignments is impractical in most cases or even unfeasible in some cases. This is another reason that makes it hard to apply this method to more complex tasks like NMT.

This last point is what we will try to tackle in the next chapter. The results that we gathered from Attentive Guidance lead us to believe that a model which utilizes the attention mechanism in a more direct and emphasized way is likely to be more compositional. A clearer alignment between input and output tokens might help the model in uncovering the underlying semantic meaning of each symbol individually. In addition to this, we argue that a more sparse or discrete attention pattern discourages the model to use any spurious attention pattern, and also helps humans to interpret how the model came to a certain solution. By plotting the sparse attention pattern, it will be easier for humans to see what input information the model used to come up with the predicted output. In the next chapter we will design a new architecture with which we aim to produce similar results as the Learned Guidance model, without the need of extra data annotation.

Chapter 6

Sequence to Attention

With the information we have gained by doing experiments with Attentive Guidance, we were encouraged to design a new sequence-to-sequence architecture that exploits these intuitions. We will first motivate this (Section 6.1). Then we will describe the architecture in full detail (Section 6.2) and share our results on some of the earlier defined test conditions (Section 6.3). Lastly we will conclude our findings in Section 6.4.

6.1 Motivation

We have gained an understanding in the previous chapter about how, without architectural changes, an RNN can be given an inductive bias about how a problem should be solved. By interpreting the internal attention mechanism as some sort of execution trace of a program, or as an alignment between the input and output symbols, we can enforce a particular alignment on the model. We show a model how it should align each output symbol to input symbols, such that it can learn a more distinct connection. This encourages the model to learn the individual meaning of each symbol, and how to combine them. Thus, showing a model this pattern might lead it towards converging on a more generalizable solution. One that exhibits systematic compositionality.

As indicated before, such a technique does require annotated data in the form of attention patterns. Two methods are developed. With Oracle Guidance, the model is provided with these attention patterns at both training and inference time. This is only to show the capabilities of a model with correct attention, as these are generally not available at all times. With Learned Guidance the model only sees the attention patterns at training time and tries to replicate them at inference time. This releases the burden of having correct attention available at inference time, but still requires it to be available at training time.

Ground-truth attention alignments are often made available by costly and error-prone human annotators. Because of the involved costs, the amount of data that can be produced is limited, while it is known that ANNs generally benefit from more data (Halevy et al., 2009). Ideally we would like to get away with the requirement of having to manually annotate the data.

Besides the generation of annotated data often being costly, it might even be impossible. As an example, when one wants to make conclusions or predictions on basis of vast amounts of historical financial data, humans often do not know which data they should focus on, let alone inform a deep learning system on such information. On the contrary, in those cases an explainable or interpretable model could prove useful to help humans to get insights in the flood of information. Or as another example, let us imagine that an entire ancient language was newly discovered, which no one knows how to translate to any known languages. However, some translations are available similar to the Rosetta stone. It would be beneficial if a model could, on its own, learn to translate sentences from the ancient language to known languages and at the same time show humans how it has done this in the form of the used attention pattern.

In this chapter we try to develop a new architecture to accomplish something similar, which we will call the *sequence-to-attention* (Seq2Attn) model. The goal is to have a similarly systematically compositional model as developed in the previous chapter, without the need to provide it with information on *how* it should solve the task. We thus eliminate the need to show the model correct attention alignments. On the contrary, this highly explainable model can show humans how it has come up with the solutions is has. The model should learn correct alignments on its own, which can be analyzed by human operators.

In a regular sequence-to-sequence model with attention, the attention mechanism can be seen as a secondary way of providing information to the decoder, the primary way being the encoded vector produced by the encoder. This attentional data might be ignored by the decoder, or could be uninformative or very distributed. In the previous chapter, we have seen that adding Attentive Guidance on such a model increases the importance of the attention mechanism, which in turn seems to increase compositional understanding in the model. The hypothesis is thus made that an informative attention pattern that is incorporated in the model with high importance, increases the compositional understanding of this model. The intuition is taken to the extreme by making the model's solution rely completely on the attention mechanism. The decoder of a regular Seq2Seq model is split into two RNNs which communicate with only information produced by an attention mechanism. Additional techniques limit the amount of information that can be passed. The second decoder must complete the task with very little information, that it can only get through an attention mechanism. As such, the model is forced to be systematic in the information it passes with this attention mechanism. This greatly reduces the chance of spurious attention patterns as observed in regular sequence-to-sequence models, and increases the explainability and interpretability of the system.

The first decoder - the one that uses its attention mechanism to generate context vectors for the second decoder - may be called a *transcoder*. Its role can be understood as having to interpret and understand the information it gets from the encoder, and to translate this into a series of informative context vectors for the second decoder. This second decoder we call simply the decoder. It receives very sparse input information and generally only executes small local functions or translations.

This design is inspired by the MAC cell (Hudson and Manning, 2018). Here the authors propose a new recurrent cell specifically designed for visual question answering in a compositional manner. The model is tested on the CLEVR task (Johnson et al., 2017), where the input consists of a query (sequence) and a knowledge base (image), and the model should output in the form of classification. Their newly designed cell consists of three units, namely the control, read and write units. Inter-unit communication is done almost entirely in the form of attention on either the query or knowledge base. The intuition and motivation for both approaches are thus similar.

As with their model, our model tries to achieve compositionality in models, and both models use *regularization through attention* to achieve this. However, there are some major difference between the architectures. Where their architecture is a single recurrent network specifically designed for a multi-model input consisting of a query and a second knowledge base, our architecture is a variant of the general sequence-to-sequence model and could thus be used for several tasks that can be represented as sequence-to-sequence tasks. Furthermore, the MAC architecture requires the number of reasoning steps to be set manually. The Seq2Attn architecture is designed for sequence-to-sequence data and the number of reasoning steps is thus determined by the length of the data.

6.2 Method

In this section we describe the Seq2Attn architecture more technically. This model consists in a multiple of extensions to the decoder and attention mechanism of a regular encoder-decoder network as described in Section 2.2. The combination of these four methods results in our studied model. Note that in the following four sections we explain the contributions incrementally. That is, we take the regular Seq2Seq model and increasingly introduce the four components that are built on top of each other to form the Seq2Attn model. However, each component could be used individually and in Section 6.3.1, the individual contribution of each component is studied.

The first two modifications are changes to the decoder and are used to encourage the use of the attention mechanism in the decoder. The first and most rigorous change we make to the regular encoder-decoder model is the introduction of a third RNN that makes it an encoder-transcoder-decoder model (Section 6.2.1). Next we describe another method to encourage the use of the context vector by an element-wise multiplication with the hidden state (Section 6.2.2).

This is followed by two changes that minimize the amount of obsolete, excess information in the context vector. This include a separation of key and value vectors (Section 6.2.3) and the use of a sparsity-enforcing activation function in the attention mechanism (Section 6.2.4).

6.2.1 Transcoder

The first component that makes the Seq2Attn model, the *transcoder*, might be seen as the main contribution. It is proposed as a way to completely remove the decoder's direct dependency on the final hidden state of the encoder. This is thus a way to enforce the use of the context vector as the primary way of information retrieval over the use of the hidden state of the encoder. In fact, the decoder that will model the output sequence does not have any direct access to the hidden states of the encoder in the forward pass.

When one would completely remove the direct information flow from the encoder to the decoder by initializing $\mathbf{h}_0^{dec}$ with a constant (zero) vector, the decoder would only get information of the input sequence through the attention mechanism. We would have thus accomplished our goal, an increased relative use of the attention mechanism. However, in the calculation of the attention alignment (Eq. 2.1), $\mathbf{h}_1^{dec}$ would thus always be the same as it is independent of the encoder

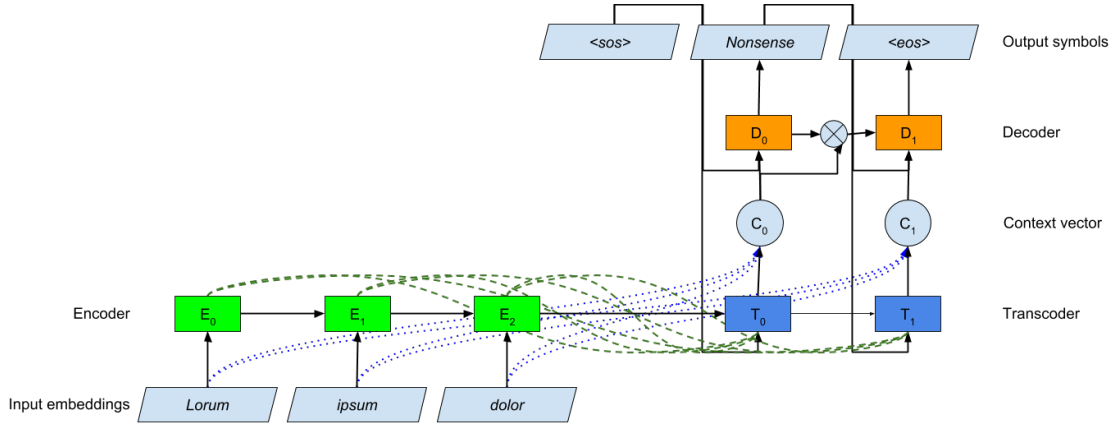


Figure 6.1: Basic schematic of the Seq2Attn model. The encoder (E) encodes the input sequence. The transcoder (T) is initialized with the encoded vector. It calculates attention weights using the encoder states, and the context vector (C) as a weighted average over the input embeddings. The decoder (D) receives the context vectors as input and models the output sequence. The context vector is additionally multiplied with the hidden state of the decoder.

and therefore independent of the input sequence.[1] We see this as undesirable behavior as the attention alignment is entirely determined by the encoder alone. This would be especially hard for a non-bi-directional encoder, as the modulation of alignment scores must be done without considering the entire input sequence and without looking ahead.

To solve this we make use of two decoder-like RNNs. The first, which we call *transcoder*, is initialized with the final hidden state of the encoder. It is augmented with an attention mechanism. However, the resulting context vectors of this mechanism are not fed to the next state of this transcoder. Instead it is the (partial) input to the second RNN. For this second RNN we simply keep the name *decoder*. This decoder is initialized with a constant, learnable vector, which is thus independent of the input sequence. The only input to this decoder is the context vector provided by the transcoder and the symbol it has predicted itself in the previous step.

We can describe the recurrent transcoder with the state transition model

$$\mathbf{y}_t^{trans}, \mathbf{h}_t^{trans} = \mathcal{S}^{trans}(\mathbf{x}_t^{trans}, \mathbf{h}_{t-1}^{trans}) \tag{6.1}$$

where the initial hidden state $\mathbf{h}_0^{trans}$ is initialized with the last hidden state of the encoder $\mathbf{h}_N^{enc}$. $\mathcal{S}^{trans}$ may be a vanilla RNN, GRU or LSTM. For the vanilla RNN and GRU, its output $\mathbf{y}_t^{trans}$ equals it hidden state $\mathbf{h}_t^{trans}$. Only for the LSTM cell this is different. The inputs $\mathbf{x}_t^{trans}$ are embeddings of the previously predicted output symbol y_{t-1} of the decoder.

$$\mathbf{x}_t^{trans} = \mathcal{E}^{trans}(y_{t-1}) \tag{6.2}$$

where y_0 is a special start-of-sequence (SOS) symbol.

[1] Since $\mathbf{h}_0^{dec}$ is a constant vector and the first input symbol is always the SOS symbol, $\mathbf{h}_1^{dec}$ is also a constant.

The transcoder uses an (almost) regular attention mechanism to provide context vectors to the decoder. It thus first computes alignment scores for each encoder state (Eq. 2.1) and normalizes these with the Softmax activation function. These probabilities are used as weights for the weighted average of the encoder states that results in the context vector (Eq. 2.3). However, we have to update Eq. 2.2 since the attention alignments are now based on the transcoder states instead of the decoder states.

$$\mathbf{a}_t(s) = \text{align}(\mathbf{h}_t^{trans}, \mathbf{h}_s^{enc}) = \frac{\exp\{\text{score}(\mathbf{h}_t^{trans}, \mathbf{h}_s^{enc})\}}{\sum_{i=1}^{N} \exp\{\text{score}(\mathbf{h}_t^{trans}, \mathbf{h}_i^{enc})\}} \tag{6.3}$$

The decoder is another state transition model with $\mathbf{h}_0^{dec}$ being a constant, learnable initial hidden state. It is thus restricted in comparison to a regular decoder in that it does not receive direct information from the encoder in the form of its last hidden state. It instead receives this information sparingly from the transcoder by means of the context vectors. The input to the decoder is a concatenation of the context vector it receives from the transcoder, and the output is has predicted itself in the previous time step.

$$\mathbf{x}_t^{dec} = [\mathbf{c}_t; \mathbf{x}_t^{trans}] \tag{6.4}$$

where it shares the same input $\mathbf{x}_t^{trans}$ with the transcoder (Eq. 6.2). We now describe the decoder as

$$\mathbf{y}_t^{dec}, \mathbf{h}_t^{dec} = \mathcal{S}^{dec}(\mathbf{x}_t^{dec}, \mathbf{h}_{t-1}^{dec}) \tag{6.5}$$

Where we take Softmax($\boldsymbol{W}_o \mathbf{y}_t^{dec}$) to model the distribution over the output symbols, of which we take the symbol with highest probability as output y_t.

6.2.2 Full Context Focus

For some tasks we found that, even with the decoder receiving only context vectors as input, the output modeling of the decoder would not depend directly enough on the context vector it would receive at a particular time step. Instead it would use context vectors of previous decoder steps and the way in which they would be processed by the recurrent connection of the decoder. We thus search for a second way to either encourage or enforce the direct use and relative importance of the attention mechanism. Our arguably ad-hoc method to do this is by an element-wise multiplication of the context vector with the hidden state of the decoder, which we call *full-context-decoder*.[2] This shows resemblance to full-context-pre-rnn attention (Section 5.2.3), but could be considered more rigorous as it directly affects the hidden state of the decoder which will be propagated through its recurrency.

We update Eq. 6.5 to be

[2]We do not have a clear logical argument for this transformation, or any proof of cognitive plausibility. This multiplication is simply performed to increase the importance of the context vector in the decoder.

$$\mathbf{y}_t^{dec}, \mathbf{h}_t^{dec} = \mathcal{S}^{dec}(\mathbf{x}_t^{dec}, \tilde{\mathbf{h}}_{t-1}^{dec}) \tag{6.6}$$

with

$$\tilde{\mathbf{h}}_t^{dec} = \mathbf{h}_t^{dec} \odot \mathbf{c}_t \tag{6.7}$$

which of course requires the decoder and context vector to be of the same dimensionality, or otherwise to be transformed to the same size.

It is easy to show that this reintroduces the vanishing gradients problem for LSTMs and GRUs, as $\frac{\delta \tilde{\mathbf{h}}_t^{dec}}{\delta \mathbf{h}_{t-1}^{dec}}$ is not the identity function. In our experiments, we did not experience the possible detrimental results of this, and we thus consider this problem future work. We imagine that one could also perform an element-wise multiplication with the *candidate hidden state* within the LSTM or GRU cell.

6.2.3 Attention is Key (and Value)

The standard attention mechanism uses a hidden state of the decoder/transcoder and a hidden state of the encoder to calculate the weight of the respective state of the encoder. This weight can be calculated with multiple alignment models like dot, concat and mlp (Eq. 2.1). From the weights, a probability distribution is calculated using a Softmax function (Eq. 6.3). The context vector is then a weighted average of the hidden states of the encoder, where the calculated probabilities are the weights (Eq. 2.3).

From this method of information retrieval, we can draw an analogy to a data structure implemented in most modern programming languages: the dictionary. There, the relevant data are stored as *values*, each associated with a different *key*. This key on its own does not necessarily contain any relevant information, but is merely used as a unique pointer to the corresponding value. In order to retrieve a value from this data structure, one must *query* the dictionary. In traditional terms, when a query is exactly similar to a key in the dictionary, the corresponding value will be retrieved.

Now let's compare this to the attention mechanism. In Eq. 2.3, the context vector is made up of a weighted average over the hidden states of the encoder. We could see these hidden states as value vectors as they contain the information that is retrieved. In Eq. 2.1, the hidden states of the decoder and encoder can be seen as the query vectors and key vectors respectively. The hidden state of the decoder is used to retrieve a specific value vector, and the hidden state of the encoder is used to point to that value. Two main differences between this method of information retrieval and a traditional dictionary are that (i) the query and key do not have to be exactly the same in order to retrieve the value that corresponds to the key and (ii) the key and value are the same vector. Instead of requiring exactly similar queries and keys, the alignment model may be interpreted as determining the closeness of the query, or alternatively, as the relevancy of the value. We do not necessarily see this as an issue. However, there is no need for the key and value to be the same vector. In fact, this may be detrimental to the efficacy of the model that uses this attention mechanism. One could say that this will put too much burden on the encoder to store too much and too varying information in its hidden states.

Recently, more clear separations of queries, keys and values have been proposed. Vaswani et al. (2017) and Dehghani et al. (2018) still use the same vectors, but transform each with a separate linear layer. Mino et al. (2017) still use the decoder states as query vectors, but separate the hidden states of the encoder into key and value vectors. Thus, of a hidden state with dimensionality n, the first $\frac{n}{2}$ units will be used as the key vector, and the remaining as the value vector. Daniluk et al. (2017) do something similar for a single-RNN network, but differentiate an additional *predict* vector and thus separate the hidden state in three parts.

We propose to also differentiate between the key and value vectors, but instead of dividing the hidden state of the encoder in two, we use the input embedding of the respective encoder state as the value vector. This makes that the context vector is a weighted average over the embeddings of the input sequence. This severely limits the amount of information that can be passed in the context vector. The information of merely unprocessed input symbols can be contained in the context vector. With this we aim for more interpretability, and for a more systematic use of the context vectors. When we would use the hidden states of the encoder as value vector, this could contain any information from the past, or with a bidirectional encoder, also from the future. We hypothesize that this would be detrimental to the compositional understanding of the model as it has more capacity to overfit, and that it would discommode to discriminate between input symbols.

Equation 2.3 is thus changed to

$$\mathbf{c}_t = \sum_{s=1}^{N} \mathbf{a}_t(s) \cdot \mathbf{x}_s^{enc} \tag{6.8}$$

6.2.4 Attention Sampling

Besides limiting the decoder in the the information it gets by using embeddings as attention values, we use a second method to limit the possible information that can be passed between transcoder and decoder with the aim to enforce a compositional, informative solution. We change the activation function which calculates the probability distribution over input states that can be attended to. The aim of this change is to have a more sparse attention vector, such that, at any time, the decoder only uses the specific information it needs and no more. As a second benefit this could help in a more interpretable model. Several methods have been develop to introspect and understand the layers of a CNN (Simonyan et al., 2013; Zintgraf et al., 2017; Erhan et al., 2009; Yosinski et al., 2015), but the inner workings of fully-connected networks and RNNs remain hard to interpret. The attention mechanism is a common method to inspect a sequence-to-sequence model, and sparse attention could help in this endeavor.

These intuition were shared by Martins and Astudillo (2016) as they proposed the Sparsemax activation function. An almost fully differentiable activation function that could replace the Softmax activation function to induce more sparsity. Niculae and Blondel (2017) have introduced a framework for sparse activation functions with Softmax and Sparsemax as special cases. Although these methods might be suited for our model, we first focus on a different activation. Our current research is focused on developing a new class of models, where we place more emphasis on the understanding of the model than the raw performance.

To this effort, we drew our attention to the Straight-Through Gumbel-Softmax function. The Gumbel-Softmax function was independently proposed by Jang et al. (2016) and Maddison et al. (2016), while the biased Straight-Through Estimator was proposed earlier Hinton (2012). In short, we experiment with this activation function to be able to use pure one-hot vectors as activations in the forwards pass, while still being fully differentiable.

We will start by explaining the Straight-Through Estimator. Originally, this is a biased gradient estimator for the binary threshold function proposed in one of Hinton's video lectures (Hinton, 2012). In the forward pass, the threshold function is used as activation function, while in the backward pass it is replaced by the identity function. This can be extended to a gradient estimator for the arg max function as well in the following way.

When we have a vector of activated hidden units $\mathbf{v}$, we can use the arg max operator to create the one-hot vector $\mathbf{v}_{max} = \text{one\_hot}(\arg\max(\mathbf{v}))$, which is obviously not differentiable with respect to $\mathbf{v}$. Next, we also create a copy of $\mathbf{v}$, which we call $\mathbf{v}_c$, which we also make non-differentiable. Now,

$$\mathbf{v}_{st} = \mathbf{v} - \mathbf{v}_c + \mathbf{v}_{max} \tag{6.9}$$

holds the same value as the one-hot vector $\mathbf{v}_{max}$, while still being differentiable with respect to $\mathbf{v}$. We can thus use a one-hot vector in the forward pass. As $\frac{\delta \mathbf{v}_{st}}{\delta \mathbf{v}}$ is the identity function, this gradient estimator is biased though.

The Gumbel-Softmax activation function was originally proposed as a way to draw samples from a categorical distribution with continuous relaxation, such that the parameterization of the categorical distribution is still trainable with backpropagation. This was originally used in a setting of Variational Auto-Encoders (Kingma and Welling, 2014). For an attention mechanism we generally don't need the stochasticity that this function adds. However, we interpret the use of this at training time as a way to regularize and to increase exploration of the network. Without it, a model could easily get stuck in a local optimum of the loss function.

Recalling Section 6.2.1, instead of computing the Softmax activations (Eq. 6.3), we first compute the log-probabilities using Log-Softmax

$$\log \pi_t(s) = \log \frac{\exp\{\text{score}(\mathbf{h}_t^{trans}, \mathbf{h}_s^{enc})\}}{\sum_{i=1}^{N} \exp\{\text{score}(\mathbf{h}_t^{trans}, \mathbf{h}_i^{enc})\}} \tag{6.10}$$

and then apply the Gumbel-Softmax function

$$\mathbf{a}_t(s) = \frac{\exp \frac{\log \pi_t(s) + g_s}{\tau}}{\sum_{i=1}^{N} \exp \frac{\log \pi_t(i) + g_i}{\tau}} \tag{6.11}$$

where g_i are samples drawn from Gumbel(0, 1), and $\tau > 0$ is the temperature. Compared to a Softmax function, the Gumbel-Softmax function can thus be interpreted as having two extra components. Firstly a temperature τ is introduced. As $\tau \to \infty$, the function approaches the uniform distribution. A low τ, approaching positive zero, will make the function more spiky and

approach the arg max function, with more noisy gradients. In our experiments we usually use quite high temperature as to allow exploration. Secondly, a reparameterization trick is performed, namely the Gumbel-Max trick (Gumbel, 1954; Maddison et al., 2016) to draw stochastic samples from the categorical distribution. This is continuously relaxed by using the Softmax function instead of the arg max function. The context vector is still calculated as in Eq. 6.8.

The temperature can be interpreted as a measure of uncertainty. A high temperature allows more random samples, while a low temperature approaches the arg max function over $\mathbf{a}_t$. Jang et al. thus advice to start training with a high temperature, but slowly anneal this over time. Instead, we have used three different settings to control the temperature. In the first setting, which we call the *constant* setting, the temperature is set to a constant. In the second setting, we let the model learn the temperature as one single latent variable. This we call the *latent* setting. In the third setting, the temperature is *conditioned* on the hidden state of the decoder for which we compute the attention vector. This allows for a more fine-grained certainty estimate. The model might be more confident about certain attention vectors than others. In this case, we let the model learn the inverse temperature as a function of the transcoder's state, similar to Havrylov and Titov (2017).

$$\frac{1}{\tau(\mathbf{h}_t^{trans})} = \log(1 + \exp(\boldsymbol{v}_\tau^\top \mathbf{h}_t^{trans})) + \tau_0 \tag{6.12}$$

where τ_0 is the maximum allowed temperature.

To assess the importance of using the Gumbel-Softmax function and the Straight-Through Estimator, we experiment with all different settings. At training time, we use either the Softmax function, or extend it with the Straight-Through Estimator, which we call Softmax ST. We also use the Gumbel-Softmax function with and without the ST estimator. For Gumbel ST and Softmax ST, we also experiment with replacing the activation function at inference time with one_hot(arg max($\mathbf{a}_t$)). Which would have the same effect as $\tau \to 0$, thus removing the stochasticity which we generally do not need in the sampling of attention vectors.

6.3 Experiments

For each of the three domains specified in Chapter 4, we will provide a quantitative analysis by looking at how well the models perform on test sets designed to test for compositionality. A qualitative analysis is also provided by looking at the attention pattern that the models produce. Additionally, we show a preliminary study on neural machine translation.

6.3.1 Lookup Tables

Optimal Models

Our hyper-parameter search found a Seq2Attn model with GRU cells and input embeddings of dimensionality 256 to be optimal, indicating that a large enough embedding space is necessary. The model had a dropout rate of 0.5, used Gumbel-Softmax ST as attention activation at training

Hyper-parameter	Lookup Tables	Symbol Rewriting	SCAN	NMT
Batch size	1	128	128	50
Optimizer	Adam, default parameters			
RNN cell	{LSTM, GRU}			GRU
Embedding size	{32, 64, 128, 256, 512, 1024}			500
RNN size	{32, 64, 128, 256, 512, 1024}			500
Dropout	{0, 0.2, 0.5}			0.2
Teacher forcing	0.5	0.5	1	0.5
Attention activation (train)	{Softmax, Softmax ST, Gumbel-Softmax ST}			Sparsemax
Attention activation (infer)	{Softmax, Softmax ST, Gumbel-Softmax ST, arg max}			Sparsemax
Attention key	$\mathbf{h}^{enc}$			
Attention value	$\mathbf{x}^{enc}$			
Temperature setting	{constant, conditioned, latent}			-
Initial/max. temperature	{0.5, 1, 5}			-
Full-context-decoder	yes	no	yes	no

Table 6.1: Fixed and tuned hyper-parameters for Seq2Attn on all considered tasks. The activation function of the attention vector at inference time is always the same as at training time. However, we also experiment with using arg max when using the Straight-Through estimator (ST) at training time.

time and arg max at inference time. For the activation function it would use a constant temperature of 5. This temperature is considerably higher than found in most literature. However, one must consider that a correct attention pattern is highly important for a correct solution and the network thus needs a high temperature for exploration.

For the baseline model we used the same hyper-parameters as found in Chapter 5.

All models were run 10 times with random weight initializations and we report average sequence accuracies.

Results

Figure 6.2 compares the sequence accuracies on all test sets. We see that the Seq2Attn model significantly outperforms the baseline on all test sets, indicating a more compositional solution. This includes the held-out tables and new compositions. These are tests sets that include the tables t_7 and t_8, which are only seen as atomic table application in the training set. When testing on composition sequences of three or longer, we see that all models are virtually unable to generalize to longer lengths. However, the Seq2Attn models perform especially poorly on this kind of generalization. They are even more prone to predicting the EOS symbol too early than the baseline models. On a test set containing all length-three compositions, the Seq2Attn models achieve only 0.55% mean sequence accuracy.

Attention Patterns

Figure 6.3 shows typical examples of the attention patterns found by both the baseline Seq2Seq and Seq2Attn models. The found attention patterns of the Seq2Attn models correspond with intuition about how the task should be solved compositionally and is also in line with the supervised Attentive Guidance.

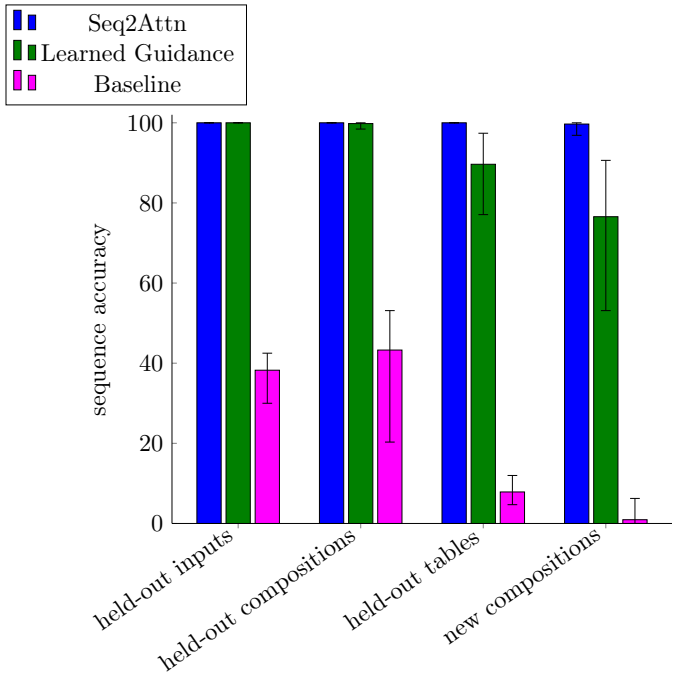
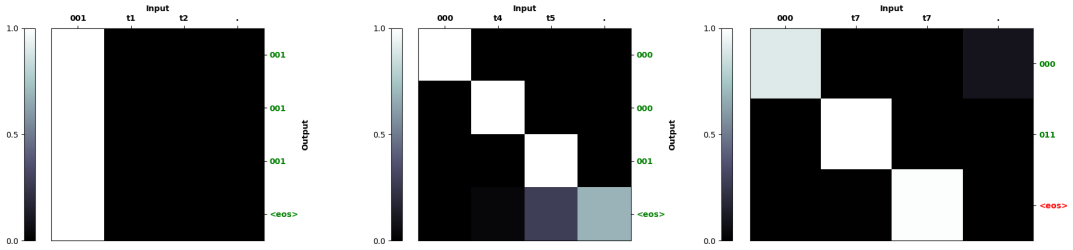


Figure 6.2: Average sequence accuracies of the baseline and Seq2Attn models on all lookup tables sets. For comparison the results of Learned Guidance are also added. Error bars indicate the minimum and maximum performing models.



(a) (b) (c)

Figure 6.3: (a) A typical attention pattern of the baseline model on the lookup tables task (*held-out inputs*) (b) A typical attention pattern of the Seq2Attn model on the same test set, using Gumbel-Softmax (without ST) as attention activation. (c) A typical attention pattern of the Seq2Attn model on *new compositions*, where it predicts the EOS symbol too early, after processing t_7. Again using Gumbel-Softmax as attention activation.

	held-out inputs	held-out compositions	held-out tables	new compositions
Baseline	30.83 ± 6.24	38.54 ± 0.74	8.51 ± 4.28	0.00 ± 0.00
Baseline+G	34.17 ± 8.25	38.54 ± 12.39	8.16 ± 3.57	0.00 ± 0.00
Baseline+E	82.50 ± 12.42	85.42 ± 12.39	31.08 ± 7.85	16.67 ± 13.09
Baseline+F	85.83 ± 16.50	91.67 ± 11.79	30.03 ± 16.12	0.00 ± 0.00
Baseline+T	43.33 ± 12.30	47.40 ± 15.33	3.99 ± 2.70	0.00 ± 0.00
Baseline+GE	82.50 ± 12.42	83.85 ± 7.48	30.21 ± 3.32	11.46 ± 1.47
Baseline+GF	69.17 ± 21.25	76.04 ± 13.28	4.69 ± 1.47	0.00 ± 0.00
Baseline+GT	32.50 ± 8.90	45.31 ± 10.13	1.56 ± 1.53	0.00 ± 0.00
Baseline+EF	85.00 ± 9.35	82.29 ± 18.46	24.13 ± 2.99	4.17 ± 1.47
Baseline+ET	**100.00± 0.00**	**100.00± 0.00**	41.49 ± 3.30	0.00 ± 0.00
Baseline+FT	68.33 ± 21.44	71.88 ± 23.00	19.44 ± 19.06	7.29 ± 10.31
Baseline+GEF	74.17 ± 36.53	72.40 ± 37.94	37.33 ± 22.10	11.46 ± 1.031
Baseline+GET	97.50 ± 3.54	98.44 ± 1.28	24.31 ± 17.87	0.00 ± 0.00
Baseline+GFT	90.83 ± 3.12	91.15 ± 3.21	28.30 ± 7.23	2.08 ± 2.95
Baseline+EFT	66.67 ± 47.14	66.67 ± 47.14	66.67 ± 47.14	66.67 ± 47.14
Seq2Attn	**100.00± 0.00**	**100.00± 0.00**	**100.00± 0.00**	**100.00± 0.00**

Table 6.2: Mean accuracies and standard deviation on the lookup tables task. The baseline model is a vanilla Seq2Seq model with attention module. We test the contribution of each of the four contributions in isolation and in combination. **G**=Gumbel-Softmax ST, **E**=embeddings as attention values, **F**=full-context-decoder, **T**=transcoder

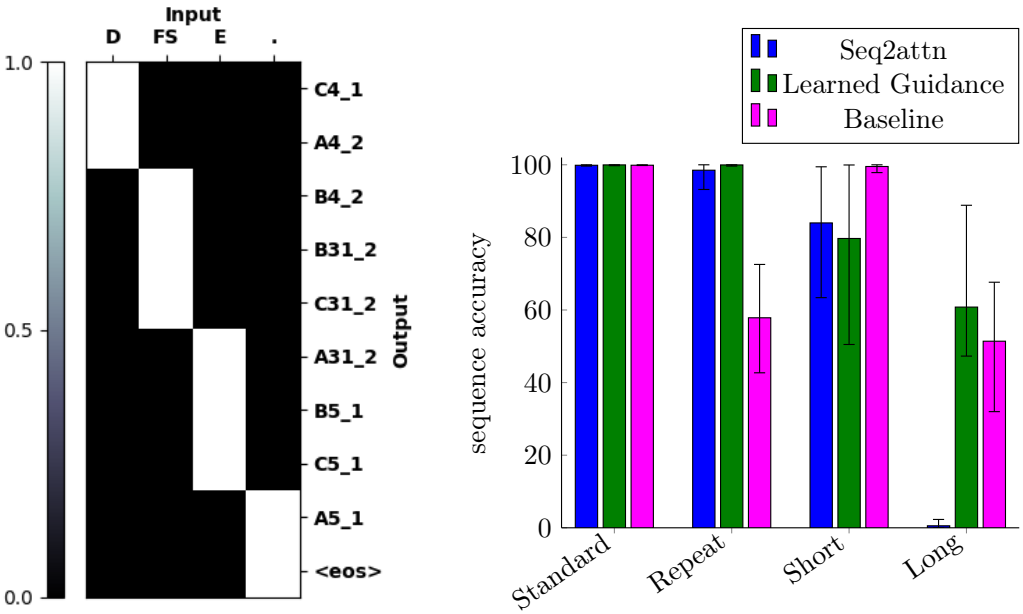
Ablation Study

We also did an ablation study to assess the contribution of each of the four components of the Seq2Attn model. These can all be applied individually or in combination on top of a vanilla Seq2Seq model with relative ease. The Seq2Attn model with optimal parameters was taken as the base model. From this model we increasingly removed components one-by-one. The hyper-parameter search was not repeated for all 16 configurations. All models thus used the same hyper-parameters as the original Seq2Attn model, which explains the baseline model to have different results here. Table 6.2 summarizes the results. There are quite some interesting observations to make. Gumbel-Softmax ST seems to have little positive or even negative effect on performance in most combinations, but essential as the final components of the Seq2Attn model. By observing the results for *Baseline+E* and *Baseline+F*, we also see that we can get already significant improvements over the baseline both by enforcing the use of the attention mechanisms as well as by forcing the attention mechanism to be more compositional.

6.3.2 Symbol Rewriting

Optimal Models

For the baseline model we again use the same hyper-parameters as found in Chapter 5. For the Seq2Attn model we found the same hyper-parameters to be optimal as the ones found for the



(a) (b)

Figure 6.4: (a) A typical attention pattern of the Seq2Attn model on *short* of the symbol rewriting task, (b) Average accuracies of both models on all test sets of the symbol rewriting task. Error bars indicate the minimum and maximum performing models.

lookup tables, except for the embedding and GRU sizes which were increased to be 512. Similar to full-context-pre-rnn being detrimental for the symbol rewriting task, the full-context-decoder is as well. We therefore do not use it for this domain. We ran each model 10 times.

Results

Figure 6.4 shows a typical attention pattern and the results on all four test sets. We see again that the Seq2Attn model outperforms the baseline on sequences of similar length as the training sequences, or is on par. The attention pattern is again mostly in line with what one would expect for a compositional solution, but often not a perfect diagonal-like pattern. It must be noted though that for each triplet of outputs, only the first has to attend to the correct input symbol, which is often the case. Although the model can still somewhat handle shorter sequences, it fails completely on longer sequences. This is in line with the results on the lookup tables.

6.3.3 SCAN

We report also results on the SCAN domain, where we focus on experiments 5 and 6.

Optimal Models

We first established the optimal hyper-parameters for the SCAN domain. For this, we used SCAN experiment 1 where the training, validation and test set all consist of uniformly sampled examples. We have chosen these sets since we do not want to bias our model selection process by considering the test set that will be tested on. We see reserving a random subset of the training set for validation as the fairest way for model selection.

A GRU Seq2Attn with embedding and hidden sizes of 512, 0.5 dropout and Gumbel ST with a constant temperature of 5 was found to be optimal.

As a baseline model we used the *overall-best* model of Lake and Baroni (2018). This was an encoder-decoder model with 2-layer LSTMs of 200 units per layer and a dropout rate of 0.5. This model did not use any attention mechanism. In order to qualitatively compare our Seq2Attn model with this baseline Seq2Seq model by comparing attention weights, we augmented the baseline with pre-rnn attention and the mlp alignment model.

Each model was ran 10 times with randomly initialized weights. We report average accuracies over the 10 models.

Results

With a near-perfect test accuracy on SCAN experiment 1 (98.47%), we confirmed that the Seq2Attn model is able to model this more complex domain. Results on the lookup tables and symbol rewriting tasks showed that the Seq2Attn model does not improve on generalization to longer sequences. We see a similar pattern in this domain as the Seq2Attn models achieve similar performance as the baseline models on SCAN experiment 2. SCAN experiment 3 is omitted, because we agree with Loula et al. that this might test for different qualities, such as one-shot learning. SCAN experiment 4 is also omitted, because experiments 5 and 6 are more fine-grained and more informative. We thus focus primarily on SCAN experiments 5 and 6 (Fig. 6.5). The results for our baseline model are very similar to those found by Loula et al..

Firstly we see that in the 0 filler experiment, where the models see zero examples of ***Primitive*** *around right* at training time, the Seq2Attn model already greatly outperforms the baseline model on the test set. The test set contains only examples containing *jump around right*. From seeing other examples like *walk right* and *jump around left* the model is thus better able to distill the meaning of the individual symbols and to recombine them in unseen contexts. The baseline models fail at this task with an average accuracy of 0.26%, while the Seq2Attn model reaches 36.23%. Although a great improvement, this is of course still far from a perfect accuracy. However, because of the interpretability of the model, we can get a more clear idea about what types of error the model makes and thus in which directions one might look when trying to solve this type of error. More on this in the next section.

For experiment 6 we see that the baseline gradually climbs from 2.86% with 1 example of *look around right* in the training set to 93.44% with 1024 examples. This gradual increase could indicate that the model is not solving this task in a systematic compositional way. Instead, with more examples, it builds up evidence for pattern recognition. The Seq2Attn model on the other hand performs already better with 0 or 1 examples and reaches to some degree a plateau at

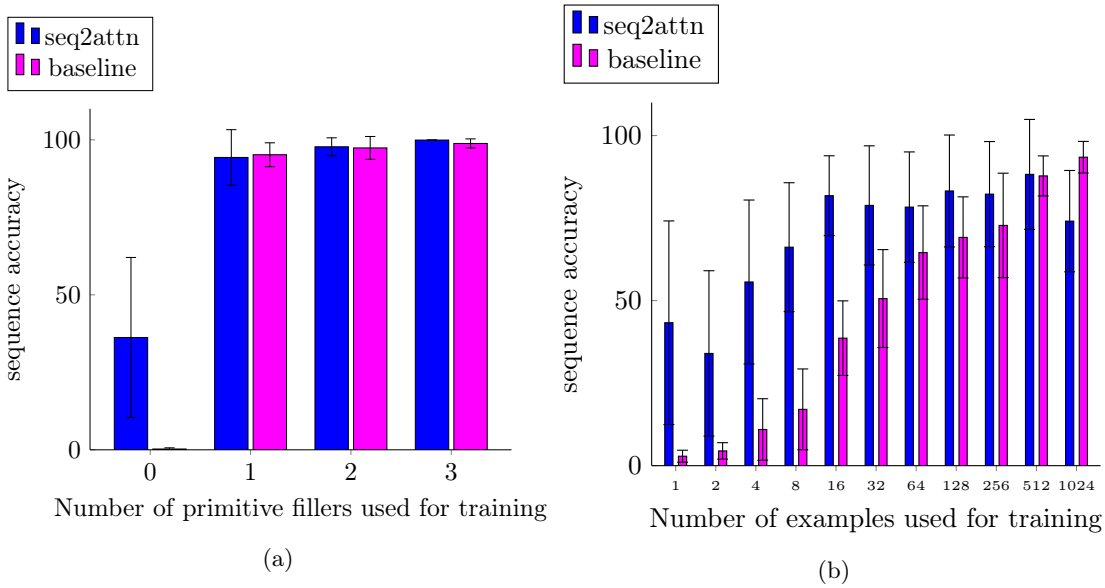
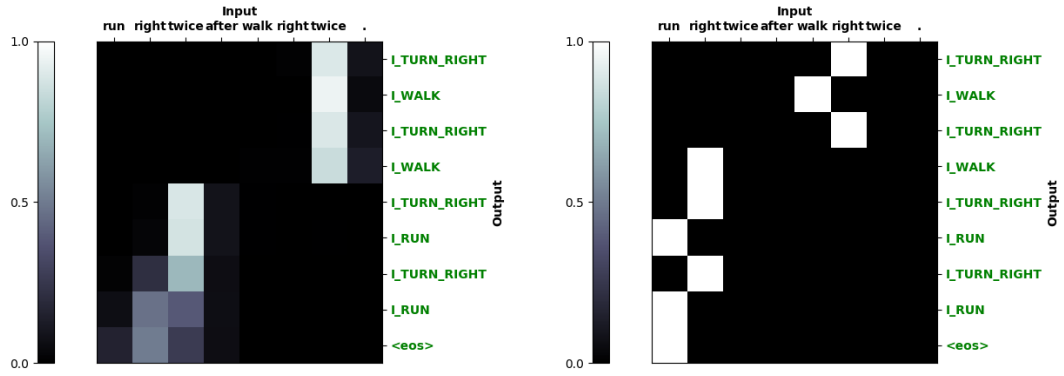


Figure 6.5: Average sequence accuracy of the baseline and Seq2Attn models on SCAN experiments 5 and 6 (experiment 2 and 3 of Loula et al. (2018)). Error bars show the bootstrapped 95% confidence interval.

already 16 examples with an accuracy of 81.78%. With 512 examples the model only slightly improves on this with 88.24%. This might be interpreted as evidence that the solution that is favored by the Seq2Attn model requires less training data, as is attributed to the principle of compositionality. Interestingly, performance drops with 1024 examples. This anomaly was not observed in repeated experiments.

Attention Patterns

We show typical attention patterns of the baseline and Seq2Attn models in Fig. 6.6. We see that in comparison to the lookup tables (Fig. 5.2a), the SCAN domain requires a more active use of the attention mechanism in the baseline models. For the lookup tables task, we saw that all relevant information was accumulated in the final hidden state of the encoder, and that the decoder would often only attend to this final state. For the SCAN domain we see a more distributed attention pattern in the baseline. However, the plot shows that in order to translate, for example, *walk right twice*, the model still only attends to *twice*. This is different for the Seq2Attn model (Fig. 6.6b). By reading this plot one could conclude that the transcoder interprets the modifiers and conjunctions like *twice* and *after*. The decoder is given a sequence of atomic input symbols that it has to directly translate to output symbols. E.g., when the decoder receives the context vector representing *run*, it directly outputs *I_ TURN_ RIGHT*. An often-seen exception to this is that, in conjunctions, the transcoder will already start providing information about the second subsentence while the decoder still has to output the first subsentence. Figure 6.6b shows this when the decoder outputs *I_ WALK* while the transcoder already moves its attention to the first subsentence *run right twice*. We thus conclude that at the fourth decoder step, the decoder can already predict what action it has to perform next. One would also suspect that the transcoder



(a) (b)

Figure 6.6: Attention patterns of the baseline (a) and Seq2Attn (b) on the same test examples of SCAN experiment 1.

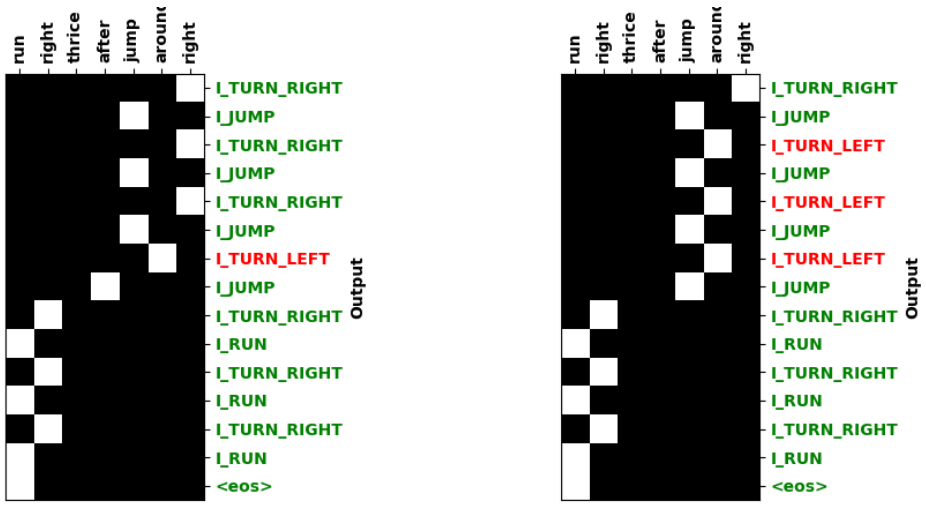
would attend to the full stop symbol to signal to the decoder that it has to emit the EOS symbol. Instead an arguably spurious strategy is chosen. Both behaviors could be interpreted as a form of overfitting

As said earlier, the highly interpretable model provides an insight in the model's solution and could pave the way towards improvements in model design or understanding. Figure 6.7 shows two attention patterns of the Seq2Attn model on the 0 filler experiment where it had an average sequence accuracy of 36.23%. Green and red labels indicate whether words are predicted correctly. These plots make clear that the errors that the models make are of a very particular kind. At training time the models have seen numerous examples of ***Primitive*** *around left*, but none of ***Primitive*** *around right*. *Around* thus has been only seen in combination with *left*. We hypothesize that the model has found that in this context both input words *around* and *left* can be used interchangeably. At testing time, the model uses *around* and *right* interchangeably as well in terms of attention patterns. However, whenever the decoder receives the context vector representing *around*, it will directly output I_TURN_LEFT because of their inferred similarity. We found the great majority of test errors on this task to be produced by the same mistake.

Since we hypothesize that the model might have inferred a similarity between *left* and *around*, one might think that this should be represented in their embeddings as well. We looked at cosine similarities between input embeddings and visualized the embedding space using t-SNE (Maaten and Hinton, 2008), but found no clear evidence for this.

6.3.4 Neural Machine Translation

The three above considered task are toy tasks that test for systematic compositionality in a controlled environment. Although they are fit for analyzing the specific solutions of models, they lack the complexities of real-world tasks. We will now make a big leap by considering neural machine translation (NMT). Note that this is an extremely preliminary study, and no hard conclusions must be drawn from this. This study is done merely to assess whether the Seq2Attn



(a) (b)

Figure 6.7: Attention patterns of two Seq2Attn models on the same test example of SCAN experiment 5, the 0 fillers experiment.

architecture is at all fit for modeling the complexities of natural language. It is debated however to which degree natural language follows the principle of compositionality (Pelletier, 1994; Szabó, 2004).

Data

We use the non-tokenized WMT'14 English-German data set. This includes a training set of 4.468.840 examples. We use *newstest2012* as a validation set, containing 3003 examples. Both sets are pre-processed by only selecting examples of which both the input and output sentence are no more than 50 symbols. Furthermore we lowercase all symbols. The models are trained on only the 50.000 most frequent input and output symbols respectively. All other symbols are replaced by *<unk>*.

Model Selection

Because of a partly non-parallelizable implementation and limited compute power, we did not perform a hyper-parameter search for this task. We have chosen for a Seq2Seq baseline model with a GRU layer of 500 hidden units, 500-dimensional embedding layers, a 0.2 dropout rate, and pre-rnn mlp attention. For the Seq2Attn model we chose similar parameters (Table 6.1). We found Gumbel-Softmax ST as the activation function for the attention vector to be too limiting for this task. Instead of resorting to the Softmax activation, we have chosen to use Sparsemax (Martins and Astudillo, 2016). This allows more input embeddings to be attended to, but results in more sparse attention vectors than with Softmax.

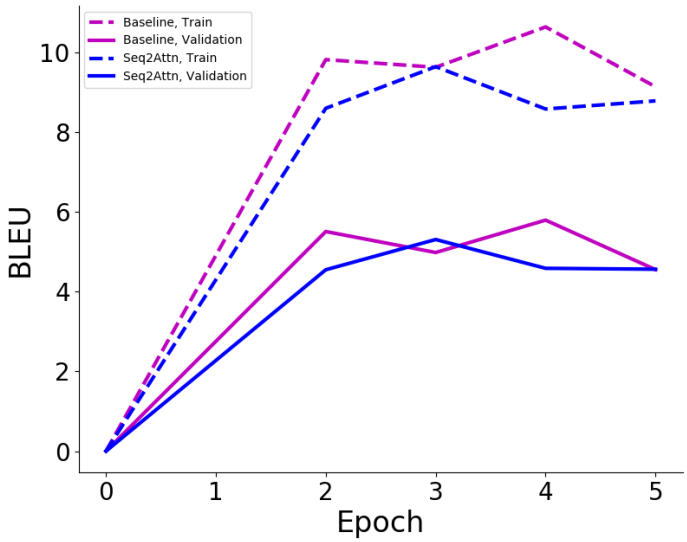


Figure 6.8: Progression of BLEU-4 score on training and validation set of WMT'14 English-German.

Results

Figure 6.8 shows the progression of the BLEU-4 score on the training and validation set during training. Although the data point for the first epoch is not present in this figure, we see that both models are mostly stable after the first epoch. Furthermore, both the training and test performance is very similar for both models, indicating that the Seq2Attn model might have similar capacities for modeling natural language as a regular Seq2Seq model. This requires, however, a more distributed attention vector than obtained with Gumbel-Softmax ST. Only 5 epochs are shown as the current implementation generally results in unstable learning after this time.

Although this experiment is not conclusive and more thorough analysis is needed, these results indicate that the Seq2Attn model does not dramatically fail when exposed to a more complex task. Even though the one-hot attention vectors produced by Gumbel-Softmax ST contain too little information, it is impressive that complex sentences can be modeled while the decoder is not initialized with the hidden state of the encoder and receives merely weighted averages over input embeddings as information from the input sequence. The similar validation scores between Seq2Seq and Seq2Attn indicate that this is not because it overfits the training set, but that the Seq2Attn model can generalize to a similar degree.

6.4 Conclusion

We have created a new architecture for sequence-to-sequence tasks where the decoder receives information solely through an attention mechanism on the input sequence. We have tested the effectiveness of this method on toy tasks that require a compositional understanding.

We found the new Seq2Attn architecture to show good performance on test data with similar input and output sequence lengths as the training data. Additionally it provides a very intuitive way to analyze its solutions and mistakes by looking at the attention pattern. The main shortcoming we found of the model is its inability to generalize to different lengths. On sequences longer than trained on, performance often drops dramatically.

Additionally we did a preliminary study on its capabilities on neural machine translation. The results give some evidence that the Seq2Attn architecture does not underperform on more complex tasks, compared to a baseline Seq2Seq model. However, a more distributed attention vector is required.

In future work we want to focus on other ways of mitigating the EOS problem. We also want to develop new methods of assessing the compositional understanding of a model by looking at its inner workings.

Chapter 7

Conclusion

We end this thesis with some final words on how we interpret the findings of Attentive Guidance and the Seq2Attn architecture. We use these to answer our original research questions and finally we give some recommendations for future work.

We augmented a sequence-to-sequence model with attention with Attentive Guidance (Hupkes et al., 2018a); A learning technique that aids the model in finding correct, compositional alignments between input and output symbols. Learned Guidance shows the practical advantages of using such a system on converging on models that allow for compositional generalization. Oracle Guidance takes away the possibly hard task of generalizing the attention vector modeling and can thus be used to analyze the added benefit of correct guidance in a more controlled, isolated environment. Compared to the baseline sequence-to-sequence model, we see that a model with Learned Guidance generalizes better to out-of-distribution data. This is shown both in the lookup tables task and the symbol rewriting task. It must be noted though that this does not strongly hold for data with input and output sequences of a different length than seen in the training set. We will address this later. Since both tasks are designed to test for compositional understanding, we conclude that we have to some degree introduced or enhanced this understanding in the models. This all is done without major adjustments to the architecture or learning techniques. This method could be applied to many similar encoder-decoder models and attention mechanisms. The only hard requirement is the need for a part of the training set to be annotated with correct attention vectors.

From the discrepancy in test accuracies between Oracle Guidance and Learned Guidance, we can conclude two things. Firstly, if we assume that models can generalize almost perfectly with correct (oracle) guidance and thus have compositional understanding, but that modeling this guidance is hard for the model to do, we could conclude that we have migrated the complexity of the tasks from the output modeling to the attention modeling. A guided model is assumed to generalize compositionally, given that it receives correct Attentive Guidance. It is thus the process of generating this guidance that has evolved into being the part of the task that requires compositional understanding. Secondly, the difference in test accuracies between Learned Guidance and Oracle Guidance is mainly observed on test examples of which the input or output sequence length is significantly different than the lengths seen during training. In these cases we often see that the EOS symbol is predicted too late or too early. In combination with the earlier

mentioned problem of complexity migration, we conclude from this that Learned Guidance does not bring significant improvements on the EOS problem (Cho et al., 2014a). However, since Oracle Guidance does show significant improvements on this, we conclude that in Learned Guidance models, this EOS problem has been migrated to an attention problem. For if the model would generate correct guidance - as in Oracle Guidance - it could generalize significantly better to longer sequences.

In summary, we thus see the following problems with Attentive Guidance: The complexity of the task has in most part been migrated to the attention modeling process, and this attention modeling process has, besides the arguable weak signal from the training data, no explicit bias to model the attention in a compositional manner. On top of this comes the problem that Attentive Guidance requires at least a part of the training data to be annotated with correct attention patterns. This is at least hard and labor-intensive for some tasks, or even impossible for other tasks as the correct guidance may be unknown. We thus argue that we should engineer a model with (i) a stronger, dedicated attention modeling component that (ii) is bounded in some way to be more compositional and (iii) does not require special annotation of the training data. We have tried to implement all these requirements in the Seq2Attn model; A model with a specialized attention modeling component called the transcoder that can communicate with the decoder solely through the attention mechanism. The output modeling thus relies extremely on the attention modeling, which by its restriction in information flow, is assumed to require a compositional solution by design.

With the added benefit of not requiring special data annotations, the Seq2Attn models show similar or improved performance compared with Learned Guidance models. On the lookup tables and symbol rewriting tasks we see significantly improved performance on test data that require compositional generalization. When analyzing the produced attention patterns, we see similar patterns as we have trained Attentive Guidance models with. However, similar to Learned Guidance models, Seq2Attn models do not improve on data with different output sequence lengths than seen at training time. They even seem to performance worse on this kind of data compared to baseline sequence-to-sequence models. Results show that this is caused by both incorrect attention modeling by the transcoder, as well as incorrect output modeling by the decoder.

With Seq2Attn models we see the following problems. Firstly, we have that the number of input symbols that can be attended to and thus the amount of information that can be provided to the decoder is directly linked to the number of produced output symbols. At each decoder step only one input symbol can be attended to. We acknowledge that this is too restrictive in some cases, such as neural machine translation. Contrarily, some tasks might require less context vectors than output symbols. The property that the number of context vectors is directly linked to the output sequence length holds for any encoder-decoder model with attention, but since the Seq2Attn model relies so heavily on the attention mechanism we hypothesize that the number of produced context vectors and the number of produced output symbols should be decoupled for the Seq2Attn model. To some degree, this too holds for Attentive Guidance models. Some suggestions to solve this are given in the Future Work section.

The second problem is greatly linked to the aforementioned problem. Comparing the results of the Seq2Attn model with Gumbel-Softmax ST on the lookup tables and symbol rewriting tasks with the performance on neural machine translation, we see that the model mainly prospers in environments that require sequential, atomic function applications. The restriction of the decoder only using context vectors produced by one-hot attention vectors as input makes that

the model has a too limited understanding of the context of the input sequence. The property of systematic compositionality that we introduced with this restriction limits at the same time the use of this model on tasks that require a more global view of the input. We hypothesize that this could be partly solved with techniques that we will cover in the Future Work section, but that this simultaneously might harm the compositional understanding of the model.

7.1 Research Questions Revisited

Our first research question was concerned with our ability to assess compositional understanding. We have used three different domains to assess the compositional generalization in sequence-to-sequence models. The lookup tables task tests for compositionality in most isolation as the application of the lookup tables requires rote memorization and the tables are small enough for typical ANNs to learn. This is thus a good task to test understanding of the compositional nature of the domain. This compositional nature is at the same time very limited though. The compose function consists in only sequentially applying functions on intermediate outputs. The symbol rewriting task is very similar in this respect. The SCAN domain is still a very controlled domain, but is already much more realistic as one input token can alter the semantic interpretation of another token. When we take as a fourth task the extremely realistic task of natural language translation[1], we think that this is a decent mixture of tasks with which compositional understanding could be assessed.

The second research question concerns our ability to improve on baseline models in the context of compositional understanding. For both Attentive Guidance and the Seq2Attn architecture we found the models to find more informative attention patterns that we associate with such an understanding. Furthermore we see significantly improved performance in the the lookup tables, symbol rewriting tasks and SCAN domains. It must be noted that on some test sets, this performance was limited mainly by faulty EOS symbol modeling. It is debatable however whether this falls in the discussion on compositional understanding.

7.2 Recommended Future Work

We see multiple research direction that might be interesting to pursue with in mind the overall goal of training deep learning models with a compositional understanding.

First opportunities lie in the design of test cases that can assess compositional understanding in a more controlled environment. Although the lookup tables task is a fine examples of a task that tests for compositional understanding in isolation, it only tests for a limited compositional understanding. Only a systematic application of simple lookup tables is required to solve the task. Further, it is not without reason that Loula et al. (2018) searched for better ways to assess compositional understanding within the SCAN domain than in the preceding work by Lake and Baroni (2018). Experiment 2 and 3 of Lake and Baroni do not only test for compositional understanding, but also test for the EOS problem and few-shot learning which, arguable, are not

[1]It is debated whether natural language actually follows the principle of semantic compositionality (Pelletier, 1994; Szabó, 2004).

completely solvable by an increase in compositional understanding. Better test cases could pave the way for easier development and analysis of compositional models.

Secondly, we see opportunities in improving on the Seq2Attn model. As discussed earlier, the number of processing steps and the number of input symbols that are attended to are directly linked to the number of produced output symbols. However, this might be very unrealistic for some tasks. We see multiple solutions. Firstly, the Gumbel-Softmax activation function (Jang et al., 2016) for the attention vector could be replaced by a sparsity-inducing activation function like the Sparsemax function (Martins and Astudillo, 2016). However, since we earlier hypothesized that compositional understanding might be linked to sparse attention vectors, such distributed attention vectors might results in a decrease in compositional understanding. We have currently only looked at the results of using Sparsemax for neural machine translation. Other approaches to increasing the information flow between transcoder and decoder might be using multi-head attention as by Vaswani et al. (2017), or using Gumbel-Sigmoid activation functions to allow multi-hot attention vectors. We especially mention adaptive computation time (Graves, 2016) as an alternative approach. Despite our efforts, we were unable to successfully implement a pondering mechanism for the Seq2Attn architecture. However, we argue that this could results in great improvements over the current architecture as the pondering mechanism could not only dynamically determine the number of processing steps per output symbol, but simultaneously determine the number of required input tokens per output symbol. We hypothesize that besides adaptive computation time, it could additionally provide adaptive attention time.

Lastly, we see opportunities of extending our work to other deep learning architectures and domains. The principle of introducing compositional understanding by means of *regularization through attention* could, for example, also be applied to image data as attention mechanisms for CNNs already exist (Xu et al., 2015; Hudson and Manning, 2018; Yin et al., 2016).

Bibliography

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate.

Barto, A. G. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77.

Bastings, J., Baroni, M., Weston, J., Cho, K., and Kiela, D. (2018). Jump to better conclusions: Scan both left and right.

Bowman, S. R., Manning, C. D., and Potts, C. (2015). Tree-structured composition in neural networks without tree-structured architectures. In *CEUR Workshop Proceedings*, volume 1583.

Brakel, P. and Frank, S. (2009). Strong systematicity in sentence processing by simple recurrent networks. In *31th Annual Conference of the Cognitive Science Society (COGSCI-2009)*, pages 1599–1604. Cognitive Science Society.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Chomsky, N. (2006). *Language and mind*. Cambridge University Press.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Daniluk, M., Rocktäschel, T., Welbl, J., and Riedel, S. (2017). Frustratingly short attention spans in neural language modeling.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. (2018). Universal transformers.

Erhan, D., Bengio, Y., Courville, A., and Vincent, P. (2009). Visualizing higher-layer features of a deep network.

Fodor, J. A. and Lepore, E. (2002). *The compositionality papers*. Oxford University Press.

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

Gan, C., Li, Y., Li, H., Sun, C., and Gong, B. (2017). Vqs: Linking segmentations to questions and answers for supervised attention in vqa and question-focused semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1811–1820.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252.

Giles, C. L., Miller, C. B., Chen, D., Chen, H.-H., Sun, G.-Z., and Lee, Y.-C. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405.

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*, volume 1.

Graves, A. (2013). Generating sequences with recurrent neural networks.

Graves, A. (2016). Adaptive computation time for recurrent neural networks.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines.

Gregor, K., Danihelka, I., Graves, A., Rezende, D., and Wierstra, D. (2015). Draw: A recurrent neural network for image generation. In *International Conference on Machine Learning*, pages 1462–1471.

Gumbel, E. J. (1954). *Statistical theory of extreme values and some practical applications: a series of lectures*. Number 33. US Govt. Print. Office.

Guo, C., Rana, M., Cisse, M., and van der Maaten, L. (2017). Countering adversarial images using input transformations.

Halevy, A., Norvig, P., and Pereira, F. (2009). The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12.

Havrylov, S. and Titov, I. (2017). Emergence of language with multi-agent games: learning to communicate with sequences of symbols. In *Advances in Neural Information Processing Systems*, pages 2146–2156.

Hendricks, L. A., Akata, Z., Rohrbach, M., Donahue, J., Schiele, B., and Darrell, T. (2016). Generating visual explanations. In *European Conference on Computer Vision*, pages 3–19. Springer.

Hinton, G. (2012). Neural networks for machine learning. Coursera, Video lectures, Lecture 15b.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012a). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors.

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

Holzinger, A., Biemann, C., Pattichis, C. S., and Kell, D. B. (2017). What do we need to build explainable ai systems for the medical domain?

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Hudson, D. A. and Manning, C. D. (2018). Compositional attention networks for machine reasoning.

Hupkes, D., Singh, A., Korrel, K., Kruszewski, G., and Bruni, E. (2018a). Learning compositionally through attentive guidance.

Hupkes, D., Veldhoen, S., and Zuidema, W. (2018b). Visualisation and'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure. *Journal of Artificial Intelligence Research*, 61:907–926.

Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., and Brox, T. (2017). Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2462–2470.

Ittycheriah, A. and Roukos, S. (2005). A maximum entropy word aligner for arabic-english machine translation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 89–96. Association for Computational Linguistics.

Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. (2017). Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 1988–1997. IEEE.

Joulin, A. and Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in neural information processing systems*, pages 190–198.

Kaiser, Ł. and Sutskever, I. (2015). Neural gpus learn algorithms.

Kant, N. (2018). Recent advances in neural program synthesis.

Karpathy, A., Johnson, J., and Fei-Fei, L. (2016). Visualizing and understanding recurrent networks.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *stat*, 1050:1.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957.

Lake, B. and Baroni, M. (2018). Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2879–2888.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Li, J., Chen, X., Hovy, E., and Jurafsky, D. (2016). Visualizing and understanding neural models in nlp. In *Proceedings of NAACL-HLT*, pages 681–691.

Liška, A., Kruszewski, G., and Baroni, M. (2018). Memorize or generalize? searching for a compositional rnn in a haystack.

Liu, P., Qiu, X., and Huang, X. (2016). Recurrent neural network for text classification with multi-task learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI-16*.

Loula, J., Baroni, M., and Lake, B. M. (2018). Rearranging the familiar: Testing compositional generalization in recurrent networks.

Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables.

Marcus, G. F. (1998). Rethinking eliminative connectionism. *Cognitive psychology*, 37(3):243–282.

Martins, A. and Astudillo, R. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning*, pages 1614–1623.

Mi, H., Wang, Z., and Ittycheriah, A. (2016). Supervised attentions for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2283–2288.

Mikolov, T., Joulin, A., and Baroni, M. (2016). A roadmap towards machine intelligence. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 29–61. Springer.

Mino, H., Utiyama, M., Sumita, E., and Tokunaga, T. (2017). Key-value attention mechanism for neural machine translation. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 290–295.

Minsky, M. (1988). *Society of mind*. Simon and Schuster.

Nguyen, H. G., Morrell, J., Mullens, K. D., Burmeister, A. B., Miles, S., Farrington, N., Thomas, K. M., and Gage, D. W. (2004). Segway robotic mobility platform. In *Mobile Robots XVII*, volume 5609, pages 207–221. International Society for Optics and Photonics.

Niculae, V. and Blondel, M. (2017). A regularized framework for sparse and structured neural attention. In *Advances in Neural Information Processing Systems*, pages 3338–3348.

Och, F. J. and Ney, H. (2000). Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 440–447. Association for Computational Linguistics.

Olah, C. and Carter, S. (2016). Attention and augmented recurrent neural networks. *Distill*.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

Pelletier, F. J. (1994). The principle of semantic compositionality. *Topoi*, 13(1):11–24.

Pelletier, F. J. (2001). Did frege believe frege's principle? *Journal of Logic, Language and information*, 10(1):87–114.

Qiao, T., Dong, J., and Xu, D. (2018). Exploring human-like attention supervision in visual question answering.

Radford, B. J., Apolonio, L. M., Trias, A. J., and Simpson, J. A. (2018). Network traffic anomaly detection using recurrent neural networks.

Reed, S. and De Freitas, N. (2015). Neural programmer-interpreters.

Siegelmann, H. T. and Sontag, E. D. (1992). On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449. ACM.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps.

Strobelt, H., Gehrmann, S., Pfister, H., and Rush, A. M. (2018). Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):667–676.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

Szabó, Z. G. (2004). Compositionality.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks.

Tang, Z., Shi, Y., Wang, D., Feng, Y., and Zhang, S. (2017). Memory visualization for gated recurrent neural networks in speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 2736–2740. IEEE.

Taylor, L. and Nitschke, G. (2017). Improving deep learning using generic data augmentation.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

urgen Schmidhuber, J. (1990). Towards compositional learning in dynamic networks technical report fki-129-90.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Weber, N., Shekhar, L., and Balasubramanian, N. (2018). The fine line between linguistic generalization and failure in seq2seq-attention models.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. (2018). Hotpotqa: A dataset for diverse, explainable multi-hop question answering.

Yin, W., Schütze, H., Xiang, B., and Zhou, B. (2016). Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association of Computational Linguistics*, 4(1):259–272.

Yosinski, J., Clune, J., Fuchs, T., and Lipson, H. (2015). Understanding neural networks through deep visualization. In *In ICML Workshop on Deep Learning*. Citeseer.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization.

Zhu, X. and Ramanan, D. (2012). Face detection, pose estimation, and landmark localization in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2879–2886. IEEE.

Zintgraf, L. M., Cohen, T. S., Adel, T., and Welling, M. (2017). Visualizing deep neural network decisions: Prediction difference analysis.