University of Amsterdam

MSc Computational Science

Master of Science Thesis

Pondering in Artificial Neural Networks

Inducing systematic compositionality in RNNs

Anand Kumar Singh
(11392045)

Thursday 16th August, 2018

Informatics Institute · University of Amsterdam

Pondering in Artificial Neural Networks

Inducing systematic compositionality in RNNs

Master of Science Thesis

For obtaining the degree of Master of Science in Computational Science at University of Amsterdam

Anand Kumar Singh

Thursday 16th August, 2018

Student number:	11392045	
Thesis committee:	Dr. ir. Elia Bruni,	ILLC UvA, Supervisor
	Ir. Dieuwke Hupkes,	ILLC UvA, Supervisor
	Dr. ir. Rick Quax,	UvA, Examiner
	Prof. dr. Drona Kandhai,	UvA, Second Assessor
	Dr. ir. Willem Zuidema,	UvA, Third Assessor

Informatics Institute · University of Amsterdam

UNIVERSITY OF AMSTERDAM

Abstract

In recent years, deep neural networks have become the dominant architectural paradigm in the field of machine learning and have even bested human beings on some complex tasks, such as the board game Go [Silver et al., 2016]. However, despite their amazing feats deep neural networks are brittle in the sense that they find it difficult to cope with data different from what they have been trained on. Human beings, on the other hand can one shot generalize to new data i.e. learn from a single sample. Their ability to do so is often attributed to their algebraic capacity [Marcus, 2003] to compose complex meaningful expression from known primitives in a systematic fashion.

In this thesis, I explore the concept of systematic compositionality and propose a novel mechanism called ‘Attentive Guidance’ (AG) to bias attention-based sequence to sequence (seq2seq) models towards compositional solutions. Using a dataset which exhibits functional nesting and hierarchical compositionality, I show that while a vanilla seq2seq model with attention resorts to finding spurious patterns in data, AG finds a compositional solution and is able to generalize to unseen cases. Subsequently I test AG on a rule-based dataset which requires a model to infer the probabilistic production rules from the data distribution. Finally, to test the pattern recognition and compositional skills of a learner equipped with AG, I introduce a new dataset grounded in sub regular language hierarchy. Over the course of this thesis, attention is motivated as a key requirement for a compositional learner. AG provides learning biases in a seq2seq model without any architectural overhead and paves the way for future research into integrating components from human cognition into the training process of deep neural networks.

Acknowledgements

First and foremost, I thank my supervisors Elia Bruni and Dieuwke Hupkes. The amount of heart and time they put into this project is overwhelming and I could not have completed it without their constant support, motivation and feedbacks. I am grateful to them for hearing out my ideas and encouraging me to pursue them in a structured way. I thank them for the engagements and meetings with the researchers at FAIR Paris, for the regular idea exchange meetings with other graduate students, which was an extremely helpful platform for discussing new ideas & getting feedback on my work and for the exposure to research in general that I received during the course of the thesis.

I would especially like to thank Kris Korrel who worked with me extensively on this project and is taking it further in even more interesting directions. I have rarely met someone with such an effortless approach to programming and I am grateful to him for helping me out whenever I got stuck with the coding part. I had some amazing technical discussions during the course of my thesis with Aashish Venkatesh and Yann Dubois, which have definitely broadened my outlook towards research and have given me amazing new ideas to ponder upon. I thank them for this and wish them all the best in their next endeavors.

Finally I would like to thank my girlfriend Pallavi for being my rock and for all her love and support, my family for their constant encouragement and the most amazing Aviral for always cheering me up.

Amsterdam, The Netherlands
Thursday 16th August, 2018

Anand Kumar Singh

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The concept of artificial neurons has existed since the 1940s [McCulloch and Pitts, 1943]. Although much of the groundwork was already done in the 80's and 90's, neural networks have only come to dominate the spectrum of artificial intelligence research since Krizhevsky et al. [2012] effectively deployed deep convolution neural networks (CNN; [LeCun et al., 1989]) to beat the previous state of the art on the ImageNet challenge [Deng et al., 2009]. Since then, deep learning has revolutionized the field of artificial intelligence and become the new state of the art in areas such as object recognition [He et al., 2015], speech recognition [Graves et al., 2013] and machine translation [Sutskever et al., 2014].

The biggest criticism of deep learning, however, is its overt dependence on voluminous training data, which has led many researchers to argue that deep neural networks are only good at pattern recognition within their training distribution [Marcus, 2018] and therefore conform to the basic tenet of statistical machine learning that train and test data should come from the same distribution [Zadrozny, 2004]. Deep neural networks, therefore, are still poor at generalizing to test data which, despite coming from the same 'rule-space', doesn't follow the exact same distribution as the training data. In contrast, human reasoning is governed by a rule-based systematicity [Fodor and Pylyshyn, 1988] which leads us to learn complex concepts or rich representations from finite set of primitives. Borrowing an example from Lake et al. [2016], human beings can learn to distinguish a segway from a bicycle from just one example, while, to do the same, a deep network might require hundreds of images of both classes.

Lake et al. [2016] argue that one of the ways of learning rich concepts in a data efficient manner is to build compositional representations. The segway in the previous example, for instance, can be represented as two wheels, connected by a base on which a handlebar is mounted while, for a bicycle, the representation could be two wheels, connected by a chain-drive all of which is mounted on a frame consisting of a rod, a seat and handlebars. A model which already has learned the 'concept' of a bicycle compositionally as described above, can re-use that knowledge to learn the representations of new parts, subparts and their relations in the case of a segway faster and in a more data efficient manner. The concept of compositionality is central to this thesis and therefore warrants an in-depth look, which is presented in the subsequent section.

1.1 Motivation

Humans exhibit algebraic compositionality in their thought and reasoning [Marcus, 2003]. I discuss the concept of compositionality and briefly review how current deep neural networks deal with it.

1.1.1 Systematic Compositionality

Compositionality is the principle of understanding a complex expression through the meaning and syntactic combination of its parts [Frege and Austin, 1968]. This definition of compositionality does not directly imply dependence on the larger context in which the expression appears or the intent of the speaker and therefore the compositional nature of natural language is an active area of debate among linguists and philosophers [Szabo, 2017]. Despite that, compositionality is arguably a crucial part of natural language owing to the fact that new meaningful complex expressions can be formed systematically by combining known words via valid syntactic operations.

One way to make progress in understanding and modeling compositionality is to focus on artificial languages, since they can be constructed to strictly follow the principles of systematic compositionality, leaving out the debated ingredients such as the influence of context and intentionality. In the next section, I discuss few artificial datasets which have been specifically created in accordance with the principle of compositionality.

1.1.2 Compositionality and Deep Learning

Deep neural networks have been shown to possess a modicum of compositional learning. LeCun et al. [2015] have argued that deep learning is adept at discovering hierarchical structures from data. For instance, in computer vision, deep neural networks learn primitive shapes (lines, circles etc.) in the shallower layer and, from that, sub-parts of an object in deeper layers [Zeiler and Fergus, 2014]. Although impressive, it can be argued that this is an instance of hierarchical feature learning, and it still requires thousands if not millions of samples (imagenet; [Deng et al., 2009]).

In the recent years, researchers have focused on creating new domains of learning which are inherently compositional and cannot be solved by mere pattern recognition. For instance, the CLEVR dataset introduced by Johnson et al. [2017] tests the visual reasoning abilities of a system such as attribute identification, counting objects, attending to multiple objects and logical operations.

Lake and Baroni [2017] introduced the SCAN dataset, which maps a string of commands to the corresponding string of actions in a completely compositional manner and is therefore a natural setting for seq2seq models (section 2.4). The commands consist of *primitives* such as “jump”, “walk” and “run”; *direction modifiers* such as “left”, “right” and “opposite”; *counting modifiers* such as “twice” and “thrice” and *conjunctions* “and” and “after”. A simple grammar generates a finite set of unambiguous commands which can be decoded if the meaning of the words are well understood. The experiments on this dataset by the authors indicates that seq2seq models fail to extract the *systematic* rules from the grammar which are required for generalization to test data that doesn't come from the same distribution as the train data but follows the same underlying rules. Seq2seq models generalize well (and in a data efficient manner) on novel samples from the same distribution on which they have been trained, but fail to generalize to samples which exhibit the same systematic rules as the training data but come from a different statistical distribution viz. longer or unseen compositions of commands. These results indicate that seq2seq models, as they are presently trained, lack the algebraic capacity to compose complex expressions from simple tokens by operating in a 'rule-space' and rather resort to pattern recognition mechanisms.

The inability of a vanilla seq2seq model to solve a compositional task was further shown by Liška et al. [2018]. The authors introduced a new dataset consisting of atomic tables which map 3-bit strings bijectively to 3-bit strings and these atomic tables can be applied sequentially to a given input to yield compositions that show functional hierarchy and nesting. This task is easily solved by a compositional system with enough memory to store all atomic tables. In their experiments the authors concluded that only a small fraction of all trained models found a compositional solution.

1.2 Objectives

The primary objectives of this thesis are as follows:

- Study the compositional abilities of current deep neural networks and come up with an approach motivated from human cognition to make deep learning more compositional.
- Introduce a new dataset for a compositional learner. The dataset should be grounded in formal language theory and should consist of tasks of progressively increasing difficulty.

1.3 Outline

In **Chapter 2**, I present the theoretical background. This chapter consists of a brief overview of the neural network architectues used throughout this thesis. It also presents the concepts of attention and pondering which are central to the work presented in chapter 3. This chapter also covers formal language theory which serves as the foundation for chapter 4.

In **Chapter 3**, I focus on the major contribution of this thesis - attentive guidance. I explain the implementation of this method and then introduce the datasets used to test it. Comprehensive results and their discussion for each of the datasets have been presented in this chapter.

In **Chapter 4**, I present a new dataset called Micro Tasks grounded in formal language theory, as a new setting for testing compositionality of attentive guidance. The experiments and results on this dataset for both baseline and attentive guidance follow the dataset description.

In **Chapter 5**, I summarize the thesis and list possible directions of future work.

Chapter 2

Technical Background

This chapter provides a brief technical overview of the models that we will frequently encounter in the rest of the thesis. A fundamental understanding of these systems is essential for appreciating the problems associated with them and the potential solution for overcoming those problems, which will be discussed in the subsequent chapters.

The key concepts of **attention** and **pondering**, whose conceptualization lies in the study of human reasoning are presented in this chapter as well. I elaborate on how these concepts have been the crucial first steps towards making deep neural networks think like human beings, thereby making their decision making process more interpretable. These concepts also serve as the backbone for the first major contribution of this thesis, which is presented in chapter 3.

This chapter concludes with an overview of formal language theory which is a prerequisite for understanding the theory behind creation of a new dataset - the second major contribution of this thesis which I present in chapter 4.

2.1 RNN - A non linear dynamical system

We frequently encounter data that is temporal in nature. A few examples would be, audio signals, videos signals, time series of a stock price and natural language. While traditional feed-forward neural networks such as a multi-layer perceptron (MLP) [Rosenblatt, 1962] are excellent at non linear curve fitting and classification tasks, it is unclear as to how they will approach the problem of predicting the value of a temporal signal T at time t given the states $T_0, T_1....T_{t-1}$ such that the states over time are not i.i.d. This is owing to the fact that a conventional feed-forward network is acyclic and thus doesn't have any feedback loops rendering it memoryless. Human beings arguably solve such problems by compressing and storing the previous states in a 'working' memory,[Miller, 1956] 'chunking' it [Neath and Surprenant, 2013] [Craik et al., 2000] and predicting the state T_t.

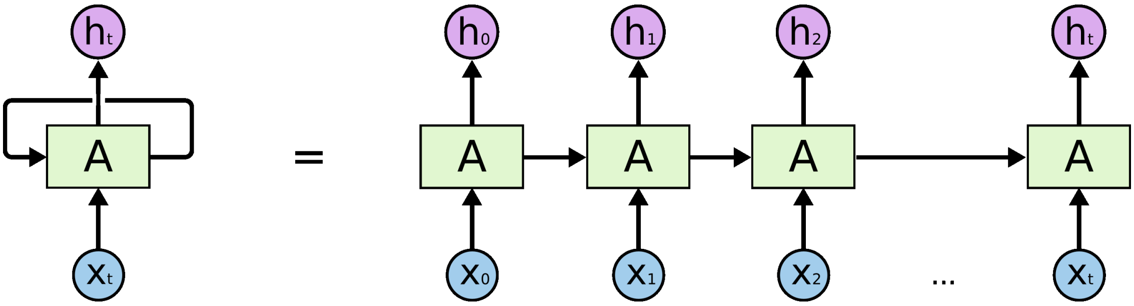


Figure 2.1: Schematic of a RNN Olah [2015]

A Recurrent neural network (RNN) [Hopfield, 1982][Elman, 1990] overcomes this restriction by having feedback loops which allow information to be carried from the current time step to the next. While the notion of implementing memory via feedback loops might seem daunting at first the architecture is refreshingly simple. In a process known as unrolling a RNN can be seen as a sequence of MLPs (at different time steps) stacked together. More specifically, RNN maintains a memory across time-steps by projecting the information at any given time step, t onto a hidden(latent) state through parameters θ which are shared across different time-steps. Figure 2.1 shows a rolled and unrolled RNN cell and the equations of an RNN are as follows:

$$c_t = tanh(Ux_t + \theta c_{t-1}), \tag{2.1}$$

$$y_t = softmax(Vc_t). \tag{2.2}$$

2.2 BPTT and Vanishing Gradients

The conventional method of training a feedforward neural network is a two step process. The first step is called the **forward pass** when values fed at input layer pass through the hidden layers, are acted upon by (linear or non-linear) activations and come out at the output layer. In the second step called the **backward pass** the error computed at the output layer (from the target output) flows backward through the network i.e. by applying the chain rule of differentiation, the error gradient at output layer, is computed with respect to all possible paths right upto input layer and then aggregated. This method of optimization in neural networks is called **backpropagation** [Rumelhart et al., 1986].

In the case of an RNN the optimization step i.e. the backward pass over the network weights is not just with regards to the parameters at the final time step but over all the time steps across which the weights (parameters) are shared. This is known as Back Propagation Through Time (BPTT) [Werbos, 1990] and it gives rise to the problem of vanishing (or exploding) gradients in vanilla RNNs. This concept is better elaborated upon through equation in the following section.

2.2.1 BPTT for Vanilla RNN

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\sum_t (y_t \log \hat{y}_t) \tag{2.3}$$

The weight W_{oh} is shared across all time steps. ∴ adding the derivatives across the sequence:

$$\frac{\partial \mathcal{L}}{\partial W_{oh}} = \sum_t \frac{\partial \mathcal{L}}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial W_{oh}} \tag{2.4}$$

For time-step t → t+1:

$$\frac{\partial \mathcal{L}(t+1)}{\partial W_{hh}} = \frac{\partial \mathcal{L}(t+1)}{\partial \hat{y}_{t+1}} \frac{\partial \hat{y}_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial W_{hh}} \tag{2.5}$$

W_{hh} is shared across time steps, we take the contribution from previous time steps as well, for calculating the gradient at time $t+1$. Summing over the sequence we get:

$$\frac{\partial \mathcal{L}(t+1)}{\partial W_{hh}} = \sum_{\tau=1}^{t+1} \frac{\partial \mathcal{L}(t+1)}{\partial \hat{y}_{t+1}} \frac{\partial \hat{y}_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_{\tau}} \frac{\partial \mathbf{h}_{\tau}}{\partial W_{hh}} \tag{2.6}$$

Summing over the whole sequence we get:

$$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \sum_t \sum_{\tau=1}^{t+1} \frac{\partial \mathcal{L}(t+1)}{\partial \hat{y}_{t+1}} \frac{\partial \hat{y}_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_{\tau}} \frac{\partial \mathbf{h}_{\tau}}{\partial W_{hh}} \tag{2.7}$$

From equation 2.7 it is clear than the gradient of a RNN can be expressed as a recursive product of $\frac{\partial h_t}{\partial h_{t-1}}$. **If this derivative is $\ll 1$ or $\gg 1$, the gradient would vanish or explode respectively** when the network is trained over longer time-steps. In the former case error back-propagated would be too low to change the weights (the training would freeze) while in the latter it would never converge.

Additionally revisiting equation 2.1 and rewriting it as follows:

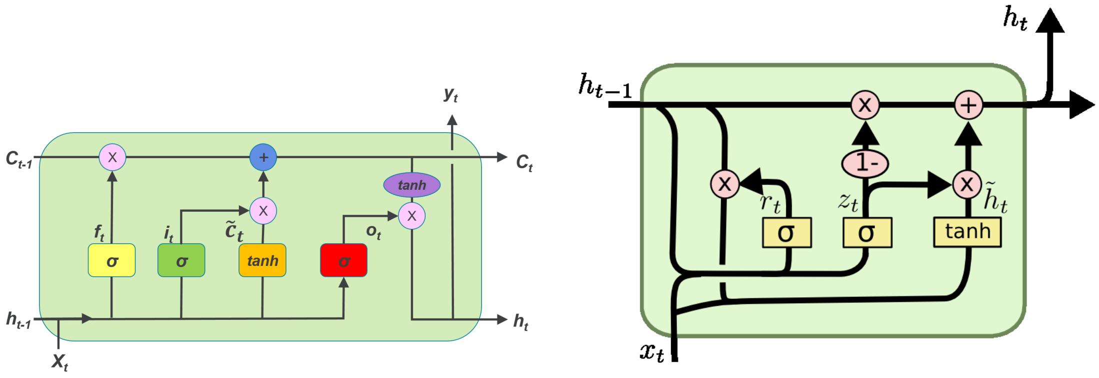
$$h^{(t)} = f(x^{(t)}, h^{(t-1)}), \tag{2.8}$$

it is not difficult to see that in the absence of an external input $x^{(t)}$ an RNN induces a a dynamical system. The RNN therefore, can be viewed as a dynamical system with the input as an external force (map) that drives it. A dynamical system can posses a set of points which are invariant under any map. These points are called the **attractor states** of a dynamical system. These set of points can contain a single point (fixed attractor), a finite set of points (periodic attractor) or an infinite set of points (strange attractor). The type of attractor in a RNN unit depends on the initialization of the weight matrix for the hidden state [Bengio et al., 1993]. Now under the application of map (input) if $\frac{\partial h_t}{\partial h_k}$ (for a large $t-k$ i.e. long term dependency), go to zero one can argue that the state h_t is in the basin of one of the attractor states. This implies that in order to avoid the vanishing gradient problem a RNN cell must stay close to the 'boundaries between basins of attraction' [Pascanu et al., 2012].

Owing to this problem of vanishing (or exploding) gradients a vanilla RNN can't keep track of long term dependencies which is arguably critical for tasks such as speech synthesis, music composition or neural machine translation. The architectural modifications which solved the vanishing gradient problem and are the current de-facto RNN cell(s) are presented in the next section.

2.3 Gated RNNs

The problem of vanishing (or exploding) gradients makes a vanilla RNN unsuitable for long term dependency modeling. However, if for instance, the RNN was to compute an identity function then the gradient computation wouldn't vanish or explode since the Jacobian is simply an identity matrix. Now while an identity initialization of recurrent weights by itself isn't interesting it brings us to the underlying principle behind gated architectures i.e. the mapping from memory state at one time step to the next is close to identity function.



(a) LSTM[Zheng et al., 2017]

(b) GRU [Olah, 2015]

Figure 2.2: Gated RNNs

2.3.1 LSTM

Long Short Term Memory (LSTM) introduced by Hochreiter and Schmidhuber [1997] is one of the two most widely used gated RNN architectures in use today. The fact that it has survived all the path-breaking innovations in the field of deep learning for over twenty years to still be the state of the art in sequence modeling speaks volumes about the architecture's ingenuity and strong fundamentals.

The fundamental principle behind the working of a LSTM is, to alter the memory vector only selectively between time steps such that the memory state is preserved over long distances. The architecture is explained as follows:

$$i = \sigma(x_t U^{(i)}, m_{t-1} W^{(i)}) \tag{2.9}$$

$$f = \sigma(x_t U^{(f)}, m_{t-1} W^{(f)}) \tag{2.10}$$

$$o = \sigma(x_t U^{(o)}, m_{t-1} W^{(o)}) \tag{2.11}$$

$$\widetilde{c_t} = \tanh(x_t U^{(g)}, m_{t-1} W^{(g)}) \tag{2.12}$$

$$c_t = c_{t-1} \odot f + \widetilde{c_t} \odot i \tag{2.13}$$

$$m_t = tanh(c_t) \odot o \tag{2.14}$$

- **input gate $i^{(t)}$:** The input computes a new cell state based on the current input and the previous hidden state and decides how much of this information to "let through" via. a sigmoid activation.
- **forget gate $f^{(t)}$:** The forget gate decides what to remember and what to forget for the new memory based on the current input and the previous hidden state. The sigmoid activation acts like a switch where 1 implies remember everything while 0 implies forget everything.
- **output gate $o^{(t)}$:** The output gate then determines determines (via a sigmoid activation) the amount of this internal memory to be exposed to the top layers (and subsequent timesteps) of the network.
- **The input modulation $g^{(t)}$** computed based on the present input and the previous hidden state (which is exposed to the output) yields candidate memory for the cell state via a tanh layer. The hadamard products of input gate and candidate memories is added to the hadamard product of forget gate and previous cell state to yield the new cell state.
- **hidden state $m^{(t)}$:** A hadamard product of the hyperbolic tangent of current cell state and output gate yields the current hidden state.

2.3.2 GRU

The Gated Recurrent Unit (GRU) introduced by Cho et al. [2014a] are a new type of gated RNN architecture whose details are as follows:

$$z_t = \sigma(x_t U^{(z)}, m_{t-1} W^{(z)}) \tag{2.15}$$

$$r_t = \sigma(x_t U^{(r)}, m_{t-1} W^{(r)}) \tag{2.16}$$

$$\widetilde{m_t} = \tanh(x_t U^{(g)}, r_t \odot m_{t-1} W^{(g)}) \tag{2.17}$$

$$m_t = (1 - z_t) m_{t-1} + z_t \widetilde{m_t} \tag{2.18}$$

- **update gate** $z_{(t)}$**:** The update gate is the filter which decides how much of the activations/memory to be update at any given time step.
- **reset gate** $r_{(t)}$**:** The reset gate is similar to the forget gate in a LSTM. When its value is close to zero it allows the cell to forget the previously computed state.
- **The input modulation** $g_{(t)}$ just as in the case of the LSTM serves the purpose of yielding candidate memories for the new cell state.
- **hidden state** $m_{(t)}$**:** The current hidden state is a weighted average of the previous hidden state and the candidate hidden state weighted by "(1 - update gate)" and the "(update gate)" respectively.

While there are a lot of similarities between a GRU and a LSTM the most striking difference is the lack of an output gate in a GRU. Unlike a LSTM a GRU doesn't control how much of its internal memory to expose to the rest of the units in the network. The GRU therefore has fewer parameters due to the lack of an output gate and is computationally less intensive in comparison to a LSTM.

2.4 Seq2Seq Models

Sequence-to-sequence (seq2seq) models introduced by Sutskever et al. [2014], Cho et al. [2014a] are a class of probabilistic generative models that let us learn the mapping from a variable length input to a variable length output. While initially conceived for machine translation, they have been applied successfully to the tasks of speech recognition, question answering and text summarization [Vinyals et al., 2015] [Anderson et al., 2018] [Lu et al., 2017].

Neural networks have been shown to be excellent at learning rich representation from data without the need for extensive feature engineering [Hinton and Salakhutdinov, 2006]. RNNs are especially adept at learning features and long term dependencies in sequential data (section 2.1). The simple yet effective idea behind a seq2seq model is learning a fixed size (latent) representation of a variable length input and then generating a variable length output by conditioning it on this latent representation and the previous portion of the output sequence.

$$h^{(t)} = f(x^{(t)}, h^{(t-1)}, v) \tag{2.19}$$

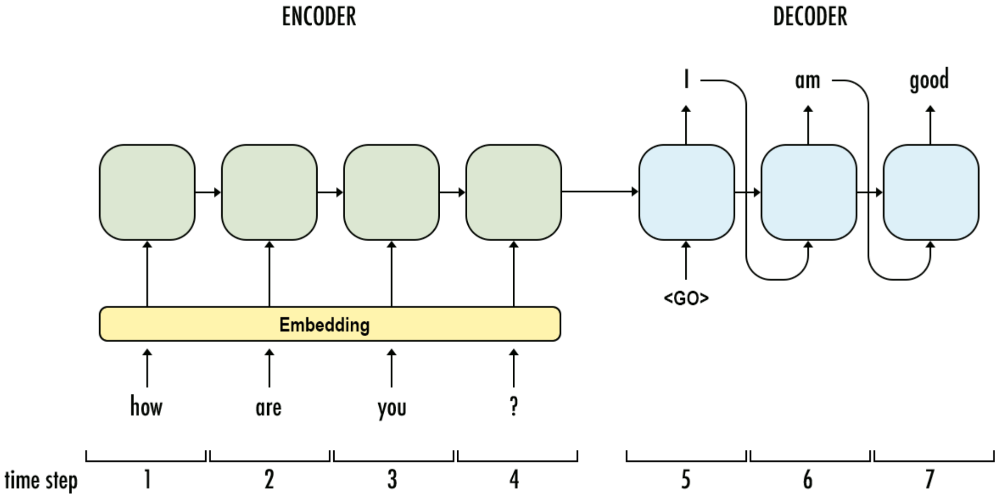


Figure 2.3: Schematic of a Seq2Seq [Chablani, 2017]

$$P(x^{(t+1)}|x^{(t)}, x^{(t-1)}, ..., x^{(1)}) = g(x^{(t)}, h^{(t)}, v) \tag{2.20}$$

It can be seen from equation 2.20 that the decoder is auto-regressive with the long term temporal dependencies captured in its hidden state. The term **v** represents the summary of the entire input state compressed into a fixed length vector (last hidden state of encoder) viz. the **latent space**. This encoder- decoder network is then jointly trained via. cross entropy loss between the target and the predicted sequence.

2.4.1 Se2Seq with Attention

It was shown by Cho et al. [2014b] that the performance of a basic encoder-decoder model as explained in section 2.4 is inversely related to the increase in length of the input sentence. Therefore in lines with concepts of the selective attention and attentional blink in human beings [Purves et al., 2013], Bahdanau et al. [2014] and later Luong et al. [2015] showed that soft selection from source states where most relevant information can be assumed to be concentrated during a particular translation step in neural machine translation (NMT) leads to improved performance. In the Bahdanau et al. [2014] framework of attention, the equations 2.19 and 2.20 are modified as follows:

$$h^{(t)} = f(x^{(t)}, h^{(t-1)}, c^{(t)}), \tag{2.21}$$

$$P(x^{(t+1)}|x^{(t)}, x^{(t-1)}, ..., x^{(1)}) = g(x^{(t)}, h^{(t)}, c^{(t)}). \tag{2.22}$$

Here unlike the traditional seq2seq model the probability of emission of the output at time step t isn't conditioned on a fixed summary representation **v** of the input sequence. Rather it is conditioned on a context vector $\mathbf{c^{(t)}}$ which is distinct at each decoding step. The context vector is calculated using a sequence of encoder outputs $(s^1, s^2, ..., s^N)$ to which the the input sequence is mapped such that an encoder output s^i contains a representation of the entire sequence with maximum information pertaining to the i_{th} word in the input sequence. The context vector is then calculated as follows:

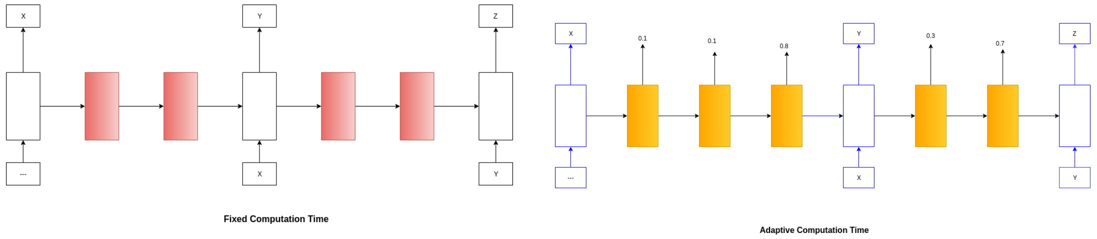
$$c^{(t)} = \sum_{j=1}^{N} \alpha_{tj} s^j, \tag{2.23}$$

where:

$$\alpha_{tj} = \text{softmax}(e(h^{(t-1)}, s^j)). \tag{2.24}$$

where e is a alignment/matching/similarity measure between the decoder hidden state $h^{(t-1)}$ i.e. just before the emission of output $x^{(t)}$. The alignment function can be a dot product of the two vectors or a feed-forward network that is jointly trained with the encoder-decoder model. The α_{tj} is an attention vector that weighs the encoder outputs at a given decoding step. Vaswani et al. [2017] view the entire process of attention and context generation as a series of functions applied to the tuple (query, key, value), with the first step being an **attentive read** step where a scalar matching score between the query and key $(h^{(t-1)}, s^j)$ is calculated followed by computation of attention weights α_{tj}. Weighted averaged of the 'values' using the attention weights is then done in the **aggregation** step.

2.5 Pondering



(a) Fixed Computation Time

(b) Adpative Computation Time

Figure 2.4: Schematic of Adaptive Computation Time

The task of positioning a problem and solving belong to different classes of time complexity with the latter requiring more time than the former. Graves [2016] argued that for a given RNN unit it is reasonable to allow for variable computation time for each input in a sequence since some parts of the input might be inherently more complex than the others and thereby require more computational steps. A good example of this would be *spaces between words and ends of sequences.*

Human beings overcome similar obstacles by allocating more time to a difficult problem as compared to a simpler problem. Therefore a naive solution would be to allow a RNN unit to have a large number of hidden state transitions (without penalty on amount of computations performed) before emitting an output on a given input. The network would therefore learn to allocate as much time as possible to minimize its error thereby making it extremely inefficient. Graves [2016] proposed the concept of **adaptive computation time** to have a trade-off between accuracy and computational efficacy in order to determine the minimum number of state transitions required to solve a problem [1].

Adaptive **C**omputation **T**ime (ACT) achieves the above outlined goals by making two simple modifications to a conventional RNN cell, which are presented as follows:

Sigmoidal Halting Unit If we revisit the equations of a vanilla RNN from section 2.1, they can be summarized as:

$$\begin{aligned} h_t &= f(Ux_t + Wc_{t-1}), \\ y_t &= g(Vc_t). \end{aligned} \tag{2.25}$$

ACT now allows for *variable state transitions* $(c_t^1, c_t^2, ..., c_t^{N(t)})$ and by extension an *intermediate output sequence* $(y_t^1, y_t^2, ..., y_t^{N(t)})$ at any given input step t as follows:

$$\begin{aligned} c_t^n &= \begin{cases} f(Ux_t^1 + Wc_{t-1}) \text{ if } n = 1 \\ f(Ux_t^n + Wc_t^{n-1}) \text{ if } n \neq 1 \end{cases}, \\ y_t^n &= g(Vc_t^n). \end{aligned} \tag{2.26}$$

A sigmoidal halting unit (with its associated weight matrix S) is now added to the network in order to yield a halting probability p_t^n at each state transition as follows:

$$\begin{aligned} h_t^n &= \sigma(Sc_t^n), \\ p_t^n &= \begin{cases} R(t) \text{ if } n = N(t) \\ h_t^n \text{ if } n \neq N(t) \end{cases}, \end{aligned} \tag{2.27}$$

where:

$$\begin{aligned} N(t) &= min\{m : \sum_{n=1}^{m} h_t^n \geq 1 - \epsilon\}, \\ R(t) &= 1 - \sum_{n=1}^{N(t)-1} h_t^n, \end{aligned} \tag{2.28}$$

and ϵ is a small constant.

Each (n^{th}) hidden state and output transition at input state t are now weighted by the corresponding halting probability p_t^n and summed over all the updates $N(t)$ to yield the final hidden state c_t and output y_t at a given input step. Figure 2.4 outlines the difference between a standard RNN cell and an ACT RNN cell by showing variable state transitions for input x and y respectively with the corresponding probability associated with each update step. It can be noted that $\sum_{n=1}^{N(t)} p_t^n = 1$ and $0 \leq p_t^n \leq 1 \; \forall n$, and therefore, it constitutes a valid probability distribution.

[1] Theoretically this is akin to halting on a given problem or finding the Kolmogorov Complexity of the data, both of which are unsolvable

Ponder Cost If we don't put any penalty on the number of state transitions then the network would become computationally inefficient and would '**ponder**' for long times even on simple inputs in order to minimize its error. Therefore in order to limit the variable state transitions ACT adds a ponder cost $\mathcal{P}(x)$ to the total loss of the network as follows: given an input of length T the ponder cost at each time step t is defined as:

$$\rho_t = N(t) + \ R(t) \tag{2.29}$$

$$\begin{aligned} \mathcal{P}(x) &= \sum_{t=1}^{T} \rho_t, \\ \widetilde{\mathcal{L}}(x,y) &= \mathcal{L}(x,y) + \tau \mathcal{P}(x), \end{aligned} \tag{2.30}$$

where τ is a penalty term hyperparameter (that needs to be tuned) for the ponder loss.

2.6 Formal Language Theory

The field of formal language theory (FLT) concerns itself with the syntactic structure of a formal language (=set of strings) without much emphasis on the semantics. More precisely a formal language L is a set of strings with the constituent units/words/morphemes taken from a finite vocabulary Σ. It is more apt to define the concept of a formal grammar before proceeding further. A formal grammar G is a quadruple $\langle \Sigma, NT, S, R \rangle$ where Σ is a finite vocabulary as previously defined, NT is a finite set of non-terminals, S the start symbol and R the finite set of valid production rules. A production rule can be expressed as $\alpha \to \beta$ and can be understood as a substitution of α with β and α, β coming from the following sets for a **valid** production rule:

$$\alpha \in (\Sigma \cup NT)^{*} NT (\Sigma \cup NT)^{*} \qquad \beta \in (\Sigma \cup NT)^{*}[2]. \tag{2.31}$$

From equation 2.31 it is easy to see that the left hand side of the production rule can never be null (ϵ) and must contain at-least one non-terminal. Now a formal language $L(G)$ can be defined as the set of all strings *generated* by grammar G such that the string consists of morphemes only from Σ, and has been generated by a finite set of rule (R) application after starting from S. The *decidability* of a grammar, is the verification (by a Turing machine or another similar computational construct e.g. a finite state automaton (FSA)) of whether a given string has been generated by that grammar or not (the *membership* problem). A grammar is decidable if the membership problem can be solved for all given strings.

2.6.1 Chomsky Hierarchy

Chomsky [1956] introduced a nested hierarchy for different formal grammars of the form $C_1 \subsetneq C_2 \subsetneq C_3 \subsetneq C_4$ as shown in figure 2.5. The different classes of grammar are

[2] The '*' denotes Kleene Closure and for a set of symbols say X, X^* denotes a set of all strings that can be generated using symbols from X, including the empty string ϵ

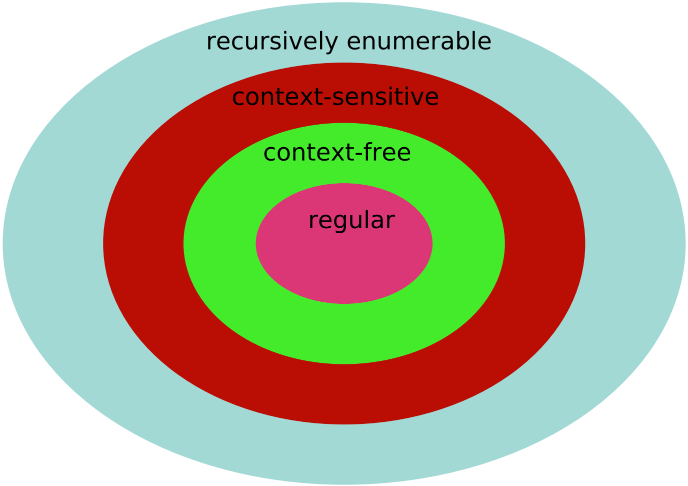


Figure 2.5: Chomsky Hierarchy

progressively strict subsets of the class just above them in the hierarchy. These classes are not just distinguished by their rules or the languages they generate but also on the computational construct needed to decide the language generated by this grammar. We now take a closer look at the classes in this hierarchy. Please note that for each of these classes the grammar definition is G = $\langle \Sigma, NT, S, R \rangle$.

Recursively Enumerable This grammar is characterized by no constraints on the production rules $\alpha \rightarrow \beta$. Therefore any valid grammar is recursively enumerable. The language generated by this grammar is called recursively enumerable language (REL) and is accepted by a Turning Machine.

Context-Sensitive This is a grammar in which the left hand side of the production rule i.e. α has the same definition as above (equation 2.31) but an additional constraint of the form $|\alpha| \leq |\beta|$ is now imposed on the production rules. This is turn leads to $\beta \in (\Sigma \cup NT)^+$, i.e. the right hand side of the production rule is now under Kleene Plus closure [3]. The non production of ϵ in context-sensitive grammars poses a problem to the hierarchy because the production of null symbol isn't restricted in its subclasses. While keeping the hierarchy as it is Chomsky [1963] resolved this paradox by defining noncontracting grammmar which is *weakly equivalent*(generates same set of string) to the context sensitive grammar. Noncontracting grammars allow the $S \rightarrow \epsilon$ production. Context sensitive grammars generate context sensitive languages which are accepted by

[3] $(\Sigma \cup NT)^+ = (\Sigma \cup NT)^* - \epsilon$

a linear bounded turing machine. While in principle this grammar is decidable, the problem is PSPACE hard and can be so complex, that it is practically intractable [Jäger and Rogers, 2012].

Context-Free This grammar is described by production rules of the form $A \to \alpha$ where $A \in NT$ and $\alpha \in (\Sigma \cup NT)^*$, such that $|A| = 1$. Context free grammar lead to context free languages (CFL) which are hierarchical in structure, although it is possible that same CFL can be described by different context free grammars, leading to different hierarchical syntactic structures of the language. A CFG is decidable in cubic time of length of string by push down FSA. A push down automaton employs a running stack of symbols to decide its next transition. The stack can also be manipulated as a side effect of the state transition.

Regular This grammar is characterized by production rules of the form $A \to \alpha$ or $A \to \alpha B$ where $\alpha \in \Sigma^*$ and $(A, B) \in NT$. The non terminal in production can therefore be viewed as the next state(s) of a finite state automaton (FSA) while the terminals are the emissions. Regular grammars are decidable in linear time of length of string by an FSA.

2.6.2 Subregular Hierarchy

The simplest class of languages encountered in section 2.6.1 were regular languages that can be described using a FSA. Jäger and Rogers [2012] however argue that the precursor to human language faculty would require lower cognitive capabilities and it stands to reason that even simpler structures can exist in the 'Regular' domain. They therefore introduced the concept of subregular languages. If a language can be described by a mechanism even simpler than the FSA then it is a subregular language. While far from the expressive capabilities of regular languages which in turn are the least expressive class in the Chomsky hierarchy, subregular languages provide an excellent benchmark to test basic concept learning and pattern recognition ability of any intelligent system.

Strictly local languages. We start with a string w and we are given a lookup table of k-adjacent characters known as *k-factors*, drawn from a particular language. The lookup table therefore serves the role of the language description. A language is k-local, if every k-factor seen by a *scanner* with a windows of size k sliding over the string w, can be found in the aforementioned lookup-table. A SL_k language description, is just the set of k-factors prefixed and suffixed by a start and end symbol, say $\#$. E.g. $SL_2 = \{\#A, AB, BA, B\#\}$

Locally k-testable languages. Instead of sliding a scanner over k-factors we consider all the k-factors to be atomic and build k-expression out of them using *propositional logic*. This language description is locally k-testable. As in the case of strictly local languages, scanner of window size K slides over the string and records for every k-factor in vocabulary its occurrence or nonoccurence in the string. The output of this scanner is then fed to a boolean network which verifies the k-expressions. E.g. 2-expression $(\neg\#B) \wedge A$, is a set of strings that doesn't start with B and consists of atleast one A.

Remarks on Chomsky Hierarchy: It is easy to see by looking at the production rules of all the grammars in Chomsky Hierarchy that, solving the languages generated by them requires an understanding of these rules. This allows us to create artificial languages such as SCAN [Lake and Baroni, 2017] which are context-free, in order to test compositionality in deep neural networks. That said, it is worth noticing that while the grammars in Chomsky Hierarchy are finite, the languages they generate can be infinite. For an infinite language one can argue that a model that can infer the grammar from the given strings is the one that will generalize well to unseen strings. However if the language itself only contains a finite number of strings then although the language itself is compositional, it can be solved also by pure memorization.

Chapter 3

Attentive Guidance

In this chapter I introduce the concept of **Attentive Guidance (AG)** which is a novel mechanism to equip seq2seq models with an additional bias to nudge them in the direction of finding a compositional solution from the search space of all possible solutions. This chapter begins with a brief overview of the prime motivation for my proposal, followed by the details of attentive guidance. I conclude my discussion on AG by showing how attentive guidance relates to the concept of pondering that we saw in section 2.5. This chapter then introduces two datasets which serve as testbeds for attentive guidance. I motivate the relevance of these datasets as pertinent test regimes for attentive guidance followed by the experimental setup for both a vanilla seq2seq baseline and an attentive guidance model. I conclude with the results obtained and a comparative analysis of vanilla seq2seq and attentive guidance on each of these domains.

3.1 Attentive Guidance

Human beings have a propensity for compositional solutions [Schulz et al., 2016] while deep neural networks lean towards pattern recognition and memorization [Marcus, 2018]. It is therefore reasonable to assume that for human level generalization learning in a systematic way is of essence. Attentive guidance aims to induce systematic learning in seq2seq networks by allowing them to focus on the primitive components of a complex expression and the way they are related to each other. The following sections elaborate on the motivation behind attentive guidance and its implementation.

3.1.1 Inspiration

Lake et al. [2015] introduced Hierarchical Bayesian Program Learning (HBPL) to learn complex characters (concepts) from few samples by representing them as probabilistic programs which are built compositionally via. Bayesian sampling from simpler primitives, subparts, parts and the relations between them, respectively. This approach led to human level generalization on the **omniglot** dataset [Lake et al., 2015] which is a dataset containing 1623 characters (concepts) with 20 samples each. Omniglot is therefore not sample intensive and hence ideally suited to test one-shot generalization capabilities of a model. This work served as the major motivation for learning nested functions such as *lookup tables* (section 3.2) of the form t1(t2(.)), by learning the compositions from simpler primitives i.e. atomic tables and then stacking them hierarchically. The procedure for learning the **trace** of the above-mentioned lookup table task is described subsequently.

3.1.2 Implementation

Attention (section 2.4.1) based seq2seq models produce a 'soft' alignment between the source (latent representation of the input) and the target. Furthermore seq2seq models require thousands of samples to learn this soft alignment. However in light of the aforementioned arguments presented in favor of concentrating on primitives to construct a complex 'composition' I propose the concept of **A**ttentive **G**uidance (AG). AG argues that the decoder having perfect access to the encoder state(s) containing maximum information pertaining to that decoding step, would lead to improved target sequence accuracy.

Revisiting the query-key-value triplet view of attention described in section 2.4.1, AG tries to improve the scalar matching score between the query and the keys during the attentive read step. Since the *keys* can be thought of as the memory addresses to the *values* which are needed at a given decoding step, AG tries to ensure a more efficient information retrieval. Similar to Lake et al. [2015] AG induces the trace of a program (although not probabilistic) needed to solve a complex composition by solving its subparts in a sequential and hierarchical fashion. This in turn forces the model to search for a compositional solution from the space of all possible solutions. AG eventually results in a 'hard' alignment between the source and target.

AG is implemented via an extra loss term added to the final model loss. As shown in figure 3.1, at each step of decoding, the cross entropy loss between calculated attention vector $\hat{a}_{t,i}$ and the ideal attention vector $a_{t,i}$, are added to the model loss. The final loss for an output sequence of length T and an input sequence of length N is therefore expressed as:

$$\widetilde{\mathcal{L}}(x,y) = \mathcal{L}(x,y) + \sum_{t=1}^{T}\sum_{i=1}^{N} -a_{i,t}\log \hat{a}_{i,t} \tag{3.1}$$

3.1.3 AG and Pondering

Pondering presented in section 2.5 facilitates variable hidden state transitions at any given input step in a recurrent unit. Attentive guidance can be seen as hardcoded or forced pondering in case of seq2seq models. This is best elaborated through an example as follows:

$$f(5) = 10 \qquad g(3) = 5 \qquad f(g(3)) = 10, \tag{3.2}$$

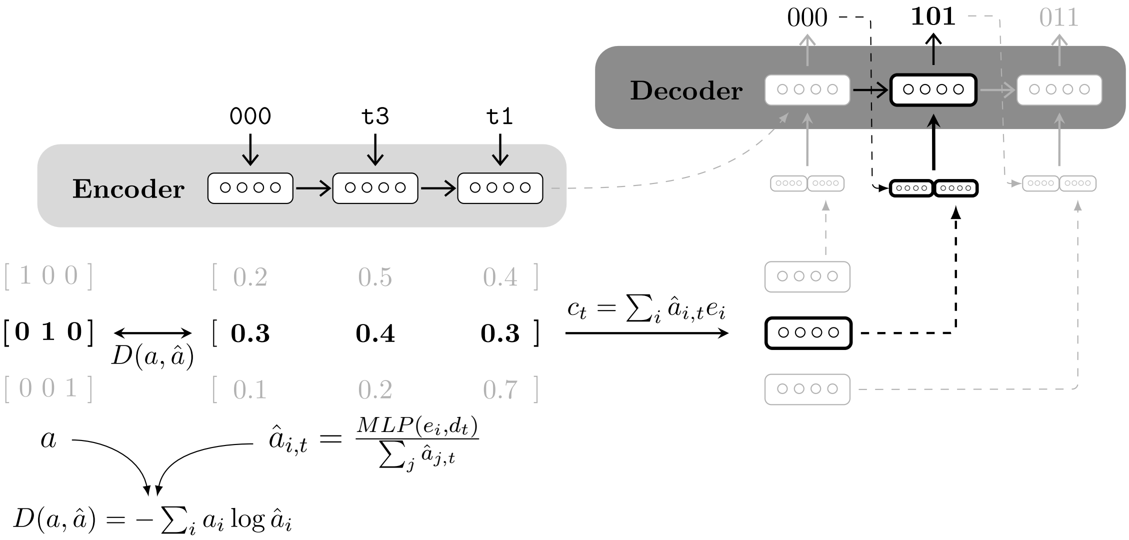

000
101
011
Decoder
000
t3
t1
Encoder
[1 0 0]
[0.2 0.5 0.4]
[0 1 0]
[0.3 0.4 0.3]
$D(a, \hat{a})$
[0 0 1]
[0.1 0.2 0.7]
$c_t = \sum_i \hat{a}_{i,t} e_i$
a
$\hat{a}_{i,t} = \frac{MLP(e_i, d_t)}{\sum_j \hat{a}_{j,t}}$
$D(a, \hat{a}) = -\sum_i a_i \log \hat{a}_i$

Figure 3.1: Attentive Guidance and calculation of AG loss [Hupkes et al., 2018]

rewriting the composed function as follows and expanding each step of composition:

$$f\ g\ 3 = 5\ 10. \tag{3.3}$$

It is easy to see that if the above example is presented to a model all it needs to do is emit the final output i.e. 10. However AG forces an additional (ponder) step to explicitly emit the intermediate output as well. Not only does this allow the decoder of the output to have an additional hidden state transition it also helps us see if the model is taking compositional steps in arriving at the final answer. In this thesis I use such 'mocked' pondering in case of lookup-tables (section 3.2) and micro tasks (chapter 4). As a future work we can think about coming up with a method to ensure that the number of ponder steps are learned by the decoder instead of being hardcoded.

3.2 Lookup Tables

The lookup tables task was introduced by Liška et al. [2018] within the CommAI domain [Baroni et al., 2017] as an initial benchmark to test the generalization capability of a compositional learner. The data consists of atomic tables which bijectively map three bit inputs to three bit outputs. The compositional task can be understood in the form of a nested function $f(g(x))$ with f and g representing distinct atomic tables. To clarify the task with an example, given $t1$ and $t2$ refer to the first two atomic tables respectively; also given that $t1(001) = 100$ and $t2(100) = 010$. Then a compositional task is presented to a learner as 001 $t1$ $t2$ = 010. Since the i/o strings are three bit, there can be a maximum of 8 input/output strings.

As is clear from the task description that since the table prompts t_i don't have any semantic meaning in itself, the meaning of each individual table prompt can be correlated only with the bijective mapping it provides. Secondly the dataset is in agreement with the systematic compositionality definition that we have outlined in section 1.1.1. Lastly one can argue that even a human learner might come up with an approach that is different than solving each function individually and sequentially in a given nested composition, but such an approach will not be able to scale with the depth of nesting.

Liška et al. [2018] in their experiments with lookup tables found that by having additional supervision on the weights of hidden state transitions, theoretically a finite-state automata (FSA) can be induced such that the recurrent layers encode the states of this automaton. This FSA can in principle solve the lookup table task upto a finite number of compositions. They further showed that this theoretical setup can achieve zero state generalization on unseen inputs on known compositions i.e. *heldout inputs* (section 3.2.1). However when trained purely on input/output mappings without this additional supervision, the authors noted that only a small percentage of networks could converge to a compositional solution.

3.2.1 Data Structure

I generated eight distinct atomic tables $t1......t8$ and work with compositions of length two, i.e. $t_i - t_j$. This leads to a possible 64 compositions. Since the requirement for the model is to not simply memorize the compositions but rather to land on a compositional solution, I propose to use the compositions only from tables $t1 - t6$ for the training set. However, since the model needs to know the mapping produced by tables $t7, t8$ in order to solve their compositions we expose the model to the atomic tables $t7, t8$ in the training set. The details of all the data splits and some dataset statistics are presented below. Examples from each split and the size of each split are presented in table 3.1

1. **train** - The training set consists of the 8 atomic tables on all 8 possible inputs. The total compositions of tables $t1 - t6 = 36$. Out of those 36 compositions we take out 8 compositions randomly. For the remaining 28 compositions we take out 2 inputs such that the training set remains balanced w.r.t. the compositions as well as the output strings. The algorithm for creation of this balanced train set is presented in appendix A.

2. **heldout inputs** - The 2 inputs taken out from the 28 compositions in training constitute this test set. However of the 56 data points, 16 are taken out to form a validation set. In creating this split I ensured that the splits i.e. *heldout inputs* and *validation* have a uniform distribution in terms of output strings at the expense of the uniformity in the compositions.

3. **heldout compositions** - This set is formed by the 8 compositions that were taken out of the initial 36 compositions. These 8 compositions are exposed to all 8 possible input strings.

4. **heldout tables** - This test is a hybrid of the tables which are seen in compositions during training i.e. $t1 - t6$ and those which are seen just atomically during training i.e. $t7 - t8$. There are total of 24 compositions in this split which are exposed to all 8 inputs.

5. **new compositions** - This split consists of compositions of $t7 - t8$ and therefore a total of 4 compositions on 8 inputs.

	Example	Size
train	t1 t2 011	232
heldout inputs	t1 t2 001	40
heldout compositions	t1 t3 110	64
heldout tables	t1 t8 111	192
new compositions	t7 t8 101	32

Table 3.1: Lookup Table Splits

In accordance with the data split described above I present the distribution of all compositions in the train and various test sets. It can be seen that the test sets 'heldout_tables' and 'new_compositions ' are the most difficult and require zero-shot generalisation owing to their significantly different distribution as compared to 'train'.

3.2.2 AG Trace

As explained in section 3.1.3 attentive guidance can help in mocking the pondering. For a lookup table composition of the form '$((000)t3)t1$' AG enforces pondering as follows:

$$t1(111) = 100 \qquad t3(000) = 111 \qquad ((000)t3)t1 = 100, \tag{3.4}$$

the composed function is expanded as follows:

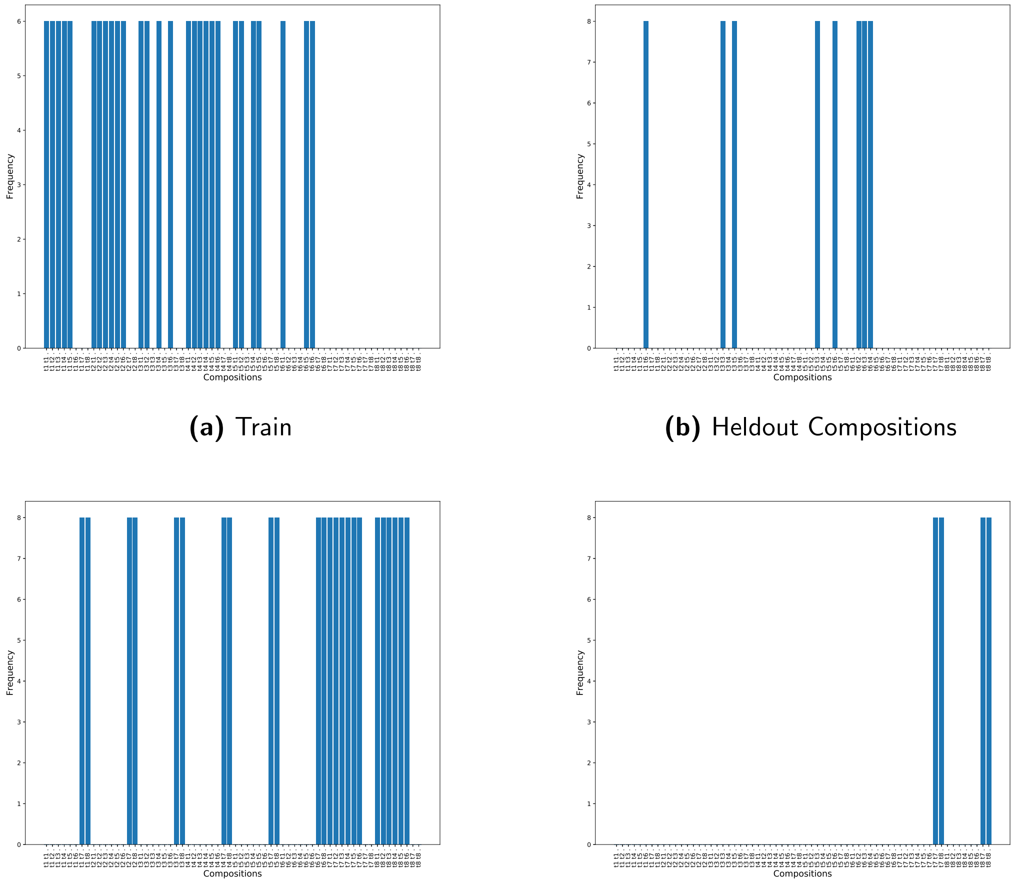
$$000\ t3\ t1 = 000\ 111\ 100. \tag{3.5}$$

AG therefore forces the decoder to ponder for two additional steps. The biggest difference from the Pondering in section 2.5 would be that at each pondering step we have an emission, instead of the ponder step being a silent one. The trace for the attentive guidance can be explained as follows:

- The first step is the copy step where the three bit input to the composition is copied as it is.
- After this the tables in the composition are applied sequentially to the three bit input preceding them.
- The diagonal trace is meant to capture this sequential and compositional solution of lookup tables. At each step of decoding the model was forced to focus on only that input prompt which results in the correct output for that step.

3.2.3 Accuracy

Since the lookup tables can be viewed as nested functions, accuracy of final output of the composition can be an adequate measure of model performance. However since we want to ensure that the model doesn't learn spurious patterns in data to land at an uncompositional solution, we want it to be accurate at each step of the composition. This hierarchical measure of accuracy is a viable test for the compositionality of the network. Therefore in all evaluations *sequence accuracy* is the performance metric of the model.



(a) Train

(b) Heldout Compositions

(c) Heldout Tables

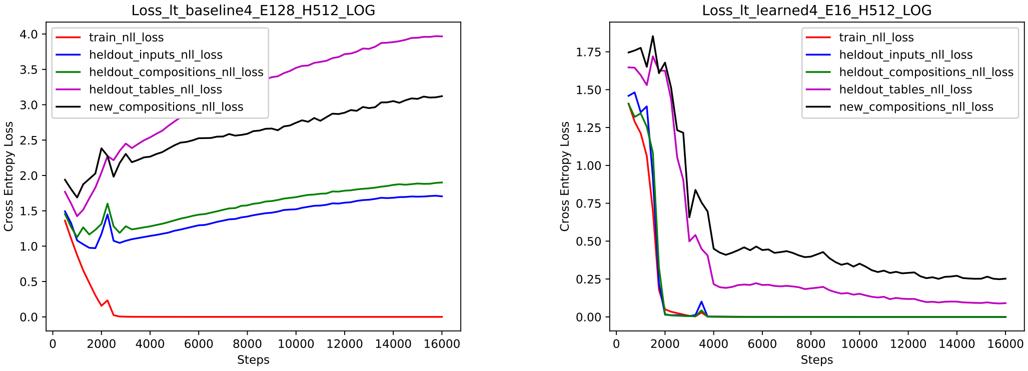
(d) New Compositions

Figure 3.2: Data distribution of train and test sets

Hyperparameters Based on the hyperparameter grid search conducted by Hupkes et al. [2018] I ran the experiments with the best hyperparameters for both the baseline - a vanilla seq2seq model (RNN cell=GRU, Embedding size=128, Hidden layer size=512, optimizer=Adam [Kingma and Ba, 2014], learning rate=0.001, attention=pre-rnn [Bahdanau et al., 2014], alignment measure=mlp(section 2.4.1)) and attentive guidance model (Embedding size=16, Hidden layer size=512, rest of the hyperparameters are same as the baseline).

3.2.4 Lookup Tables - Results

The impact of AG has been tested by making a comparative study of *learned/guided* models and *baseline* models. Both models are a standard seq2seq architecture as explained in section 2.3. The only difference between a guided model and baseline is the presence of the extra attention loss term (equation 3.1) along with the cross-entropy loss between predictions and targets. I sampled five different train and test sets and trained baseline and guided models on each one of them to account for stochasticity during different model runs. I present the results of both the models on the different test sets, discuss their zero shot generalization capabilities and end this section with an analysis of the impact of replacing the learned component of attentive guidance with the exact attention target vectors.



(a) Loss Curves for Baseline

(b) Loss Curves for Learned

Figure 3.3: Learning Curves for Lookup Tables

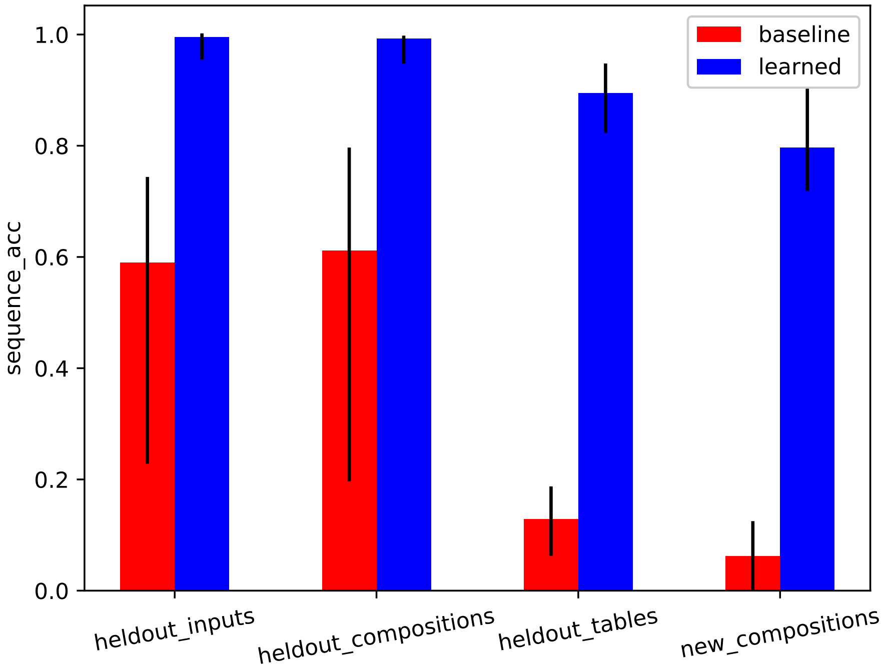
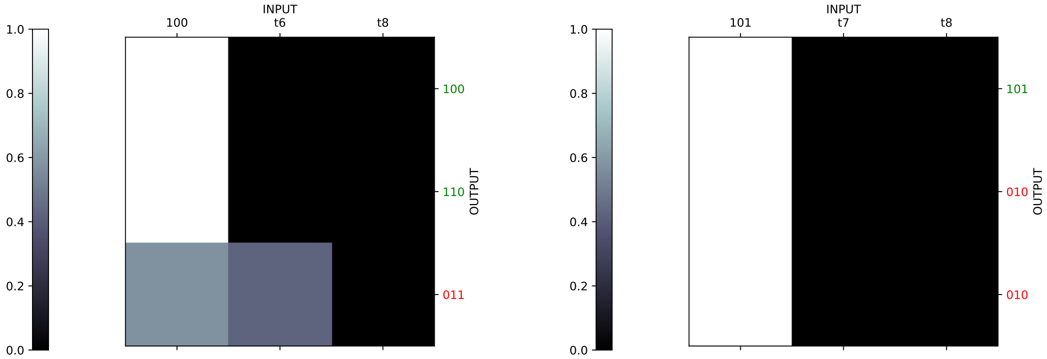


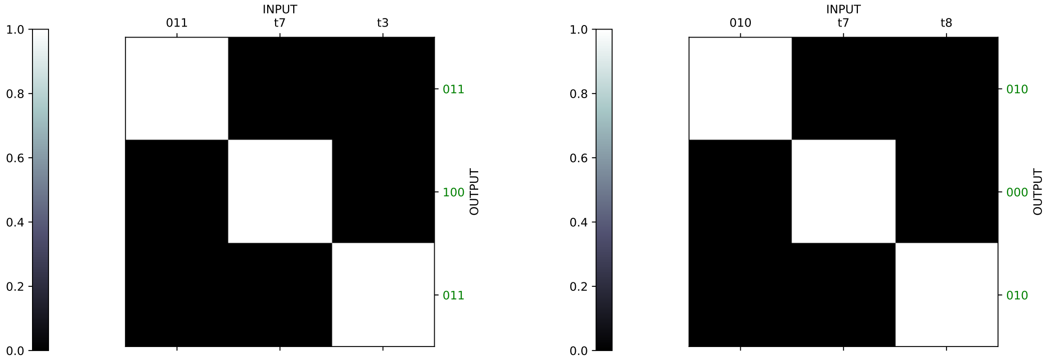
Figure 3.4: Average Sequence Accuracies. Error bars indicate maximum and minimum values achieved by the models.

Focusing on the loss development in the case of baseline (figure 3.3a) and comparing it to the loss development in the case of guided model (figure 3.3b) it can be easily concluded that attentive guidance offsets overfitting which is exhibited by the baseline on all the test sets. Further guided models converge much faster in comparison to the baseline models. Despite overfitting on the test sets, the baseline models perform reasonably well above chance level (which stands at 0.2% for this task) on the *heldout inputs* and *heldout compositions* test splits. However, in contrast to the approximately 60% accuracy achieved by the baseline of these test sets, the guided model achieves over 99% i.e they outperform the baseline by 65%. As we move towards more difficult datasets viz. *heldout tables* and *new compositions* which require zero shot generalization the performance of baseline model progressively deteriorates. Guided models however still manage to achieve 90% and 80% sequence accuracy respectively on these datasets.



(a) Heldout Tables **(b)** New Compositions

Figure 3.5: Attention plots for baseline model.

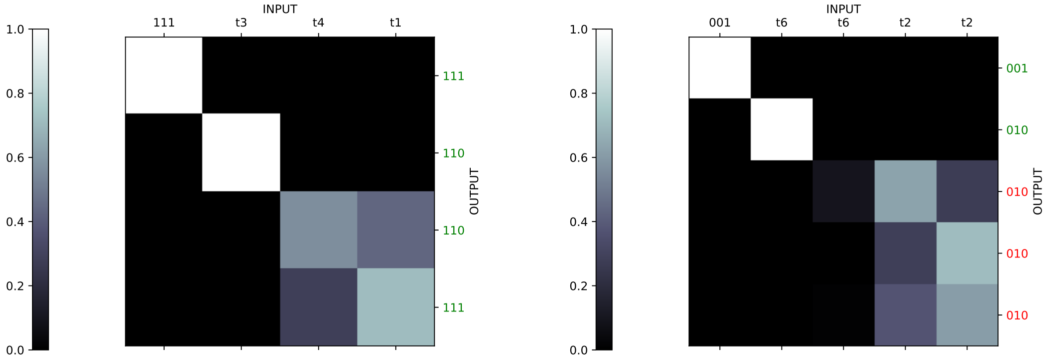


(a) Heldout Tables **(b)** New Compositions

Figure 3.6: Attention plot for guided model.

Analysis of Attention: One of the most salient features of attention based seq2seq models (section 2.4.1) is that it is easy to visualize the attention vectors generated by the decoder, thereby making the model interpretable to some degree. Figure 3.5 shows that the diffused attention in baseline prohibits it from landing on a compositional solution.

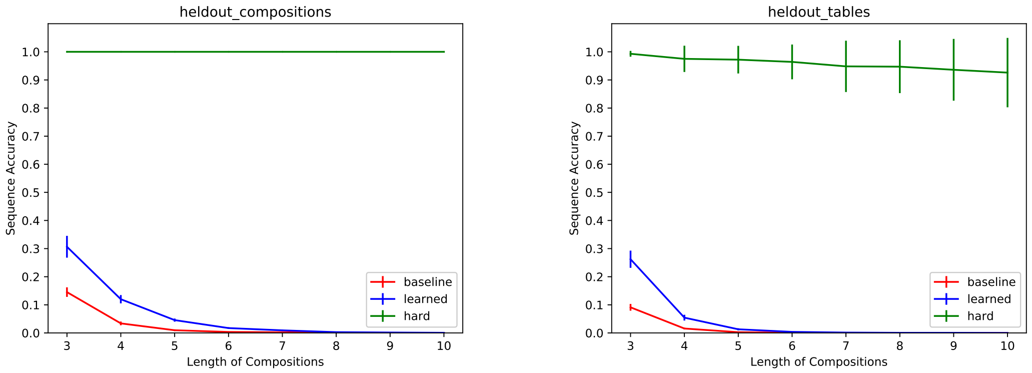
The green emissions are correct while the red emissions are incorrect at a given decoding step. Therefore in order to arrive at a compositional solution, the model would need to be compositional even at the intermediate steps. This is easily seen by comparing figures 3.5b and 3.6b.



(a) Attention plot for length=3 heldout composition

(b) Attention plot for length = 4 heldout composition

Figure 3.7: Attention plot for longer heldout composition processed by a guided model.



(a) Longer Heldout Compositions

(b) Longer Heldout Tables

Figure 3.8: Sequence Accuracy for Longer Compositions

Longer compositions and hard guidance: Longer compositions (of length 3 and above) provide the most challenging test case to the guided model and require absolute zero shot generalization from the guided model since it has never been exposed to a sample of length 3 or above. Looking at the attention plots of figure 3.7, we see the guided model now exhibiting the same diffused attention pattern which was the characteristic of baseline attention in figure 3.5. Unsurprisingly the performance of the guided models on longer compositions declines rapidly (it is still higher than baseline) as can be seen from figure 3.8. However this observation led to experiments with *hard guidance*. Hard guided model remove the need for learning the correct attention by providing the target attention vector at each decoding step. Therefore hard guidance can give some insight into the source of limitation of attentive guidance. As it turns out models trained with hard guidance gave perfect performance on longer heldout compositions, irrespective of the length of the composition (figure 3.8a). Even for a difficult case such as longer heldout tables, hard guidance exhibited high accuracy which had a small gradient with respect to length of composition.

Having tested AG on a dataset that explicitly enforces compositional solution for generalization, it can be concluded that attentive guidance forces a model to discover a compositional solution. This is owing to the fact that AG was able to generalize to compositions which are significantly different from what the model was trained on. But these compositions could be solved by solving the atomic steps in the composition sequentially. I now test AG on a dataset for which the rules are implicit and require a model to infer them from a training data which is considerably larger in size compared to the lookup table data.

3.3 Symbol Rewriting

Introduced by Weber et al. [2018] the symbol rewriting dataset is essentially a probabilistic context free grammar (PCFG). It consists of rewriting a set of input symbols to a set of output symbols based on this grammar. Before proceeding further with the task description, I'll elaborate on PCGFs briefly.

PCFGs are the stochastic version of CFGs (Context free grammars) that we encountered in section 2.6.1. The addition of this stochasticity was motivated by the non-uniformity of words in natural language. Assigning probabilities to production rules, leads to a grammar more in line with the Zipfian distribution of words in natural language [Jurafsky et al., 2014]. A PCFG consists of:

1. A CFG $\mathcal{G} = \langle \Sigma, N, S, R \rangle$ where the symbols have the same meaning as defined in section 2.6.
2. A probability parameter $p(a \to b) \mid \sum\limits_{a \to b | a \in N} p(a \to b) = 1$
3. Therefore, the probabilities associated with all the expansion rules of a given non-terminal should sum up to 1.

The parse tree shown in figure 3.9 illustrates the production rule for one input symbol. The grammar consists of 40 such symbols each following similar production rules. Weber et al. [2018] showed using this dataset that seq2seq models are powerful enough to learn some structure from this data and generalize on a test set which was drawn from the same distribution as the training set. They posit that given the simplicity of the grammar it should be possible to generalize to test sets (with some hyperparameter tuning) whose distribution is markedly different from the training distribution while still conforming to the underlying grammar. They however show that this indeed **is not** the case.

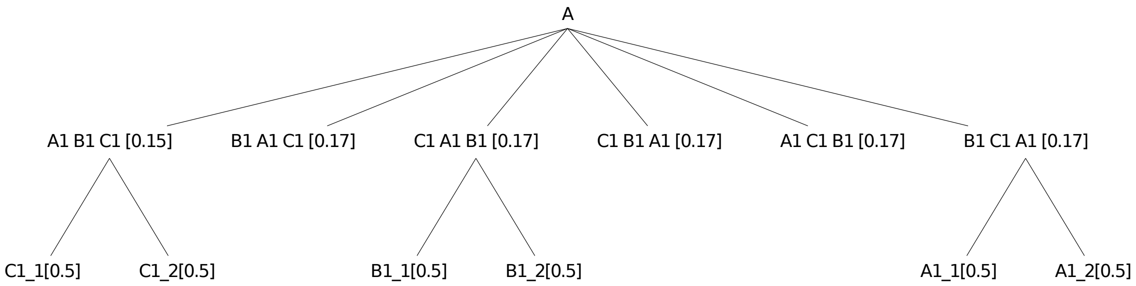


Figure 3.9: Parse Tree for one Input Symbol. The final symbols (leaf nodes) are shown just once with respect to their parent nodes for clarity.

3.3.1 Data Structure

The data-splits as furnished [Weber et al., 2018] consists of a training data and different test cases which are non-exhaustive and created by sampling randomly from all possible i/o pairs as described by the PCFG. The different test sets are created to ascertain if the seq2seq models actually understand the underlying grammar from the training data or simply memorize some spurious structure from the training distribution. For hyperparameter tuning, a validation set which is an amalgamation of random samples from all the different test sets, is used. The details of the different data splits are presented below. Examples from each split and the size of each split are presented in table 3.2

1. **train** consists of 100000 pairs of input output symbols with input string length ranging between ⟨ 5 and 10 ⟩. Output string length is therefore between ⟨ 15 and 30 ⟩. A crucial feature of this set is that no symbol is repeated in a given input string.
2. **standard test** consists of samples drawn from the same distribution as the training set.
3. **repeat test** includes input strings where repetition of symbols is allowed.
4. **short test** includes input strings which are shorter in length as compared to the input strings in the training data. The input string length ranges between ⟨ 1 and 4 ⟩.
5. **long test** consists of input sequences of lengths in the range ⟨ 11 and 15 ⟩.

	Example	Size
train	HS E I G DS	100000
standard test	LS KS G E C P T	2000
repeat	I I I I I I MS	2000
short	M I C	2000
long	Y W G Q V I FS GS C JS R B E M KS	2000

Table 3.2: Symbol Rewriting Splits

The datasets *repeat, short and long* come from distributions which are different from the training distribution on which the model has learned the data structure. It is expected of a compositional learner that given the sufficient size of the training data it would be able to infer a pattern which is close to the underlying PCFG and therefore generalize of the test sets which comes from different distributions but have the same underlying structure.

3.3.2 AG Trace

Every input symbol leads to three emissions in the output sequence and each output symbol is generated by exactly one input symbol. Therefore, for every emission in the output sequence the index of the input symbol responsible for the emission, is the trace. For every input there are three decoding steps and for all the three steps the decoder should attend to that particular input.

3.3.3 Accuracy

Given the probabilistic nature of the symbol rewriting dataset it is not difficult to reason that sequence accuracy wouldn't be the ideal performance metric for this task. For instance A → A1_1 B1_2 C1_1 and A → A1_2 B1_1 C1_2 are both valid productions and since the data is probabilistic, the possibility of both the targets in the training data is equally likely. The accuracy of a prediction is evaluated in the following three steps:

- Since every input symbol leads to emission of three output symbols, the output length should be $3 *$ (input length).
- The input vocabulary consists of 40 symbols (A - OS). The output is always a permutation of a three tuple of the form (Aj_i Bj_i Cj_i) with j = index of the corresponding input symbol in the vocabulary and $i \in [1, 2]$. The second check is to ensure that none of the Aj_i Bj_i Cj_i are repeated.
- The third check is that the j as described above = index of the corresponding input symbol in the vocabulary.

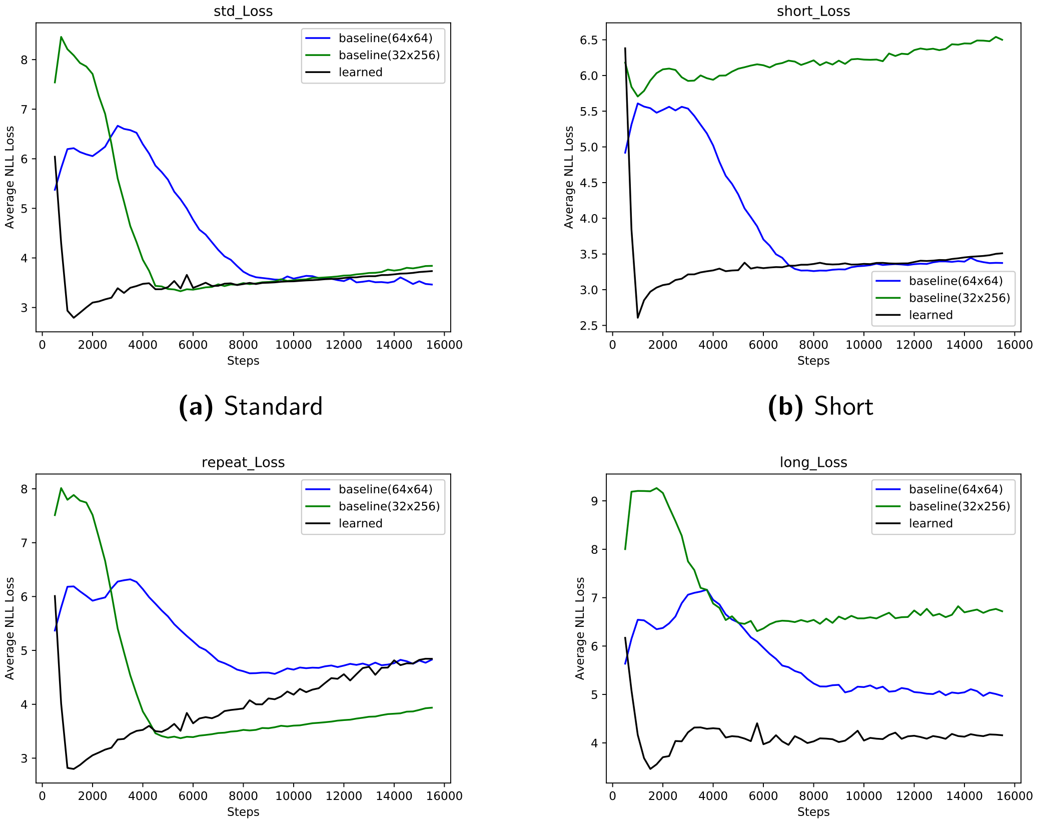
If a prediction passes all these three checks in that order, the accuracy of that prediction is 1 else 0.

Hyperparameters Based on the hyperparameter grid search conducted by Hupkes et al. [2018] I ran the experiments with the best hyperparameters for baseline models (RNN cell=GRU, Embedding size=64, Hidden layer size=64, optimizer=Adam [Kingma and Ba, 2014], learning rate=0.001, attention=pre-rnn [Bahdanau et al., 2014], alignment measure=mlp(section 2.4.1)) and the guided models (Embedding size=32, Hidden layer size=256, rest of the hyperparameters are same as in baseline). Additionally since the guided models are operating in a larger parameter space as compared to the baseline, I ran baseline models with the same hyperparameters as the guided models to ensure an unbiased comparison between them. Henceforth baselines (Embedding size=64, Hidden layer size=64) and (Embedding size=32, Hidden layer size=256) are referred to as *baseline1* and *baseline2* respectively.

3.3.4 Symbol Rewriting - Results

The experiments were set in accordance with the requirements outlined by Weber et al. [2018] 1) that a model should learn the generalizable structure from the data i.e should infer the PCFG rules and 2) should not learn spurious dependence between input and output lengths. The purpose of *repeat*, *short and long* datasets respectively is to test a model on these requirements. 50 runs for each of the two baselines and the guided model were carried out with different initialization and their results were then averaged. A comparative analysis of the baselines and guided model has been carried out. I conclude this section with an analysis of the effect of hard guidance.

Loss development on each of the 4 test sets, over the course of training as shown in figure 3.10 brings forth an interesting observation apart from the fact that guided models converge faster than the baselines. Even the guided (or learned) model seems to be overfitting to the training. However that is not the case since the data is probabilistic and one input can have multiple outputs. Loss therefore is not the best benchmark for comparing the performance of the three models.



(a) Standard

(b) Short

(c) Repeat

(d) Long

Figure 3.10: Average NLL Loss over 50 runs of the selected configuration of all three models

Focusing now on the accuracy (symbol rewriting accuracy as described in section 3.3.3), in figure 3.11, we can see that our results for baseline are fairly consistent with the ones presented by Weber et al. [2018]. Baseline2 in fact is almost at par with guided model in the standard (same distribution as training data) and repeat(similar to standard with repetition of input symbols allowed) datasets, but it doesn't satisfy the second requirement outlined by the authors in the original paper. It can't generalize to test sets where the input length changes as compared to training data. Baseline1 on the other hand generalizes relatively better to all test sets, although the accuracy is still significantly low in comparison to guided model.

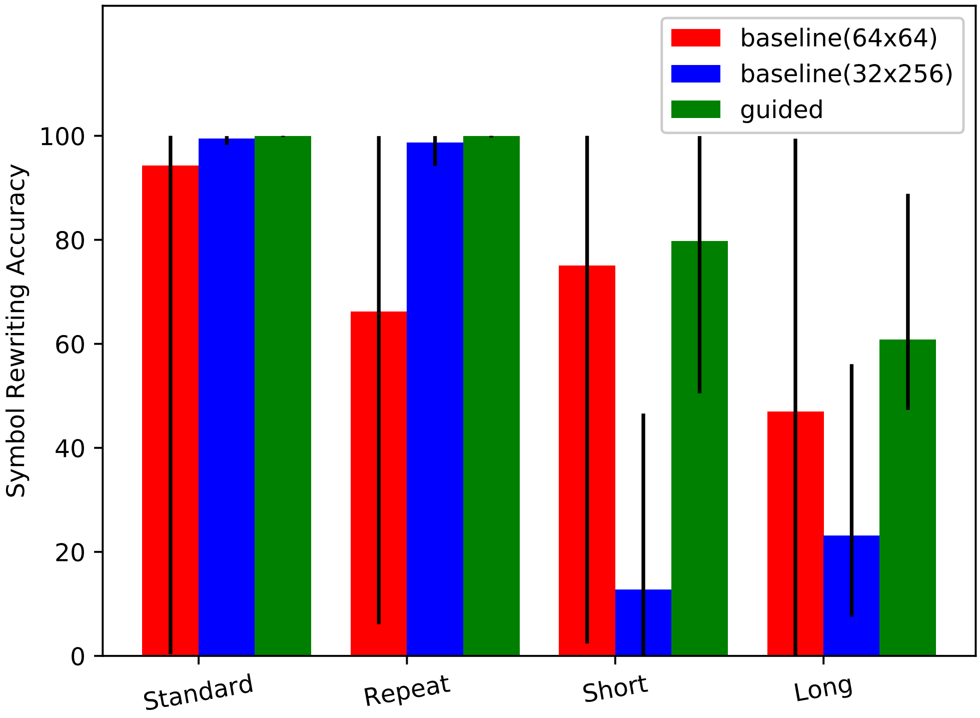
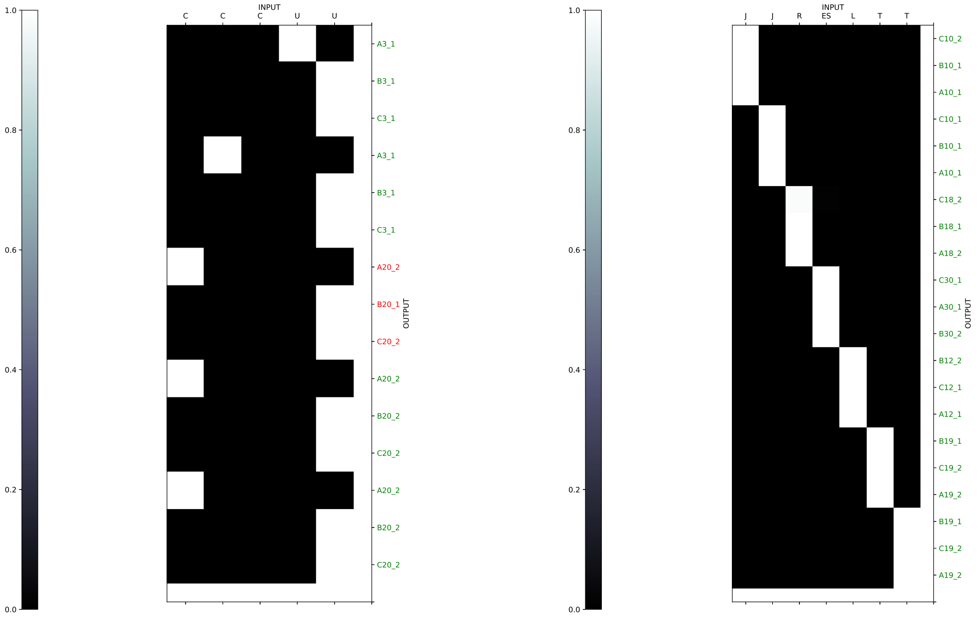


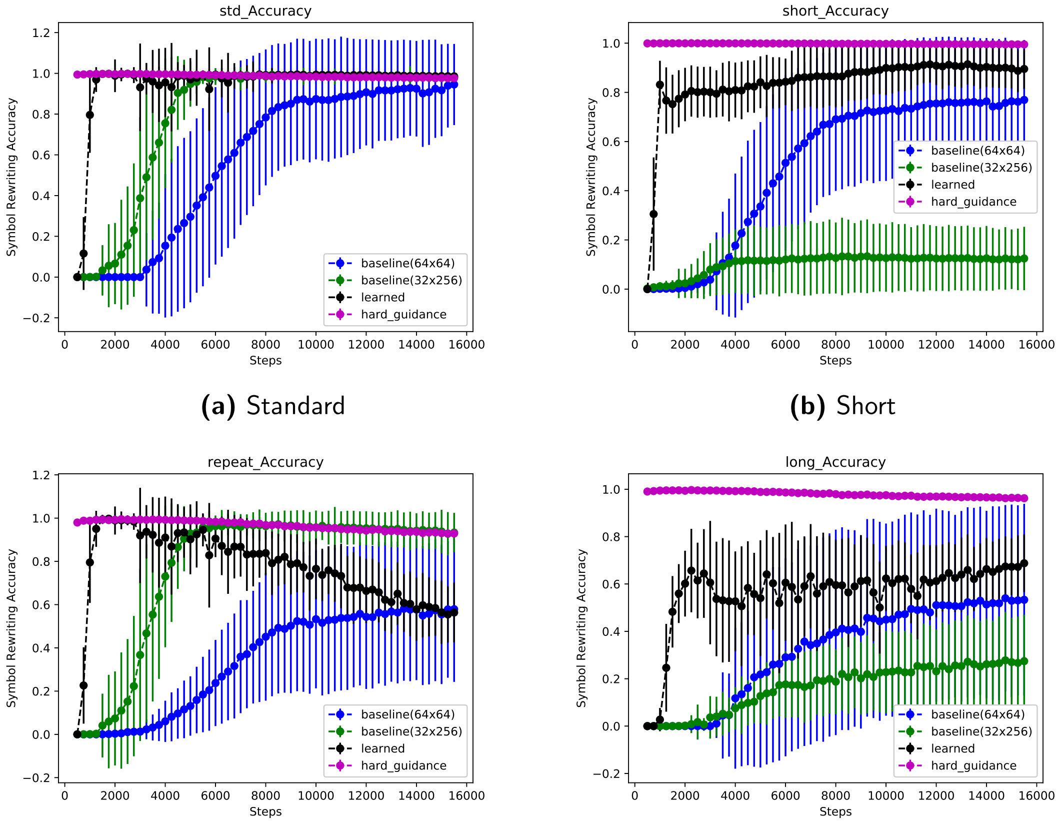
Figure 3.11: Sequence accuracies for symbol rewriting test splits. Error bars indicate maximum and minimum values achieved by the models.

Analysis of Attention: Similar to the lookup tables case the baseline attention is diffused over the entire input sequence in stark contrast to the attention of guided models, which have perfectly learned the trace for the symbol rewriting task as outlined in section 3.3.2. It can be argued by looking at figure 3.12, that the lack of attentive guidance causes baseline to memorize spurious patterns from the data and that's why after two successful production for the input symbol C it falters when it sees the same symbol for a third time.



(a) Baseline attention plot for a ‘repeat’ sample **(b)** AG attention plot for a ‘repeat’ sample

Figure 3.12: Attention plots for symbol rewriting



(a) Standard

(b) Short

(c) Repeat

(d) Long

Figure 3.13: Average accuracy over 50 runs of the selected configuration of all three models

Solving symbol rewriting with hard guidance: The guided models perform better on the test splits as compared to the baselines and as can be seen from figure 3.13, they do so while showing less variance than the baseline models. However they still can't overcome learning the dependencies between the input and output length and this is reflected in their poor performance on the longer test data. Models trained with hard guidance however were able to reach almost perfect accuracy on all the test splits as seen in figure 3.13.

3.4 Discussion

This chapter introduced the concept of Attentive Guidance which is a novel concept to *nudge* a seq2seq network in the direction of compositional solutions. The proposed method was tested in two domains, one of which involved sequential application of functions to solve a nested or composed function while the other one required a model to infer the rules of the grammar underlying a dataset in order to be able to generalize to test data which differs significantly from the training data while following the same rules. We saw that hard guidance was actually able to solve both the problems and generalize to unseen test data and this bolsters the argument that attention or more precisely the guidance of it is central to finding compositional solutions. At the same time, the inability of models trained with AG to scale up to complex datasets in a zero shot fashion indicates that the current architecture and the method of learning the guidance via. cross entropy loss isn't optimal and we need to come up with more robust mechanisms to guide attention.

Chapter 4

Micro Tasks

In chapter 3 we saw that AG exhibits compositional learning and can learn PCFG rules when explicitly asked to do so. We can now test if it can infer the rules of the grammar behind a formal language from the dataset and then generalize to unseen strings generated using the grammar in a zero-shot fashion. Since it has already been seen in chapter 1 that Lake and Baroni [2017] introduced the SCAN domain which is essentially a context free language and showed that vanilla seq2seq models can't learn the rules governing this dataset in a systematic fashion, it stands to reason that instead of directly testing the compositional abilities of my method on a context free language which is still higher up in the Chomsky Hierarchy, I start with the much simpler domain of sub-regular languages.

CommAI mini tasks [Baroni et al., 2017] are a set of strictly local and locally testable (please refer to section 2.6.2) languages designed to test the generalization capabilities of a compositional learner. Based on the mini tasks I propose a new dataset of sub-regular languages which I call the Micro Tasks. While the CommAI mini task position themselves as strictly local and locally testable tasks of progressive difficulty we add another layer of difficulty to the tasks by setting them up as either a decision problem (verify) or search problem (produce) in the same training domain.

Search vs Decision - It has been shown that for a general language $L \in NP$ search is more difficult than decision Bellare and Goldwasser [1994]. The *verification* task is clearly a decision problem while the *production* task is a search problem. Since Micro tasks fall into the domain of sub-regular languages they can be decided at-most in the linear time of length of string by a finite state automaton (section 2.6.1), they clearly are $\notin NP$. However it still makes sense to analyze whether for a prover (seq2seq network, lstm etc.) is the task of computing the witness for the membership in a $L \notin NP$ more difficult than establishing the existence of such a witness.

The **verify** task input is of the form '$\langle$ string of k-factors $\rangle$ verify $\langle$ witness $\rangle$' with the output being a standard classification token 'yes/no'. The witness string is of length $\geq$ the given k-factor string, and a model must verify if the k-factors exist in the witness. The **production** task consists of an input of the form '$\langle$ string of k-factors $\rangle$ produce' with the output being a satisfying assignment (equal to the length of the given string) to the criteria laid out in the input string.

Any given ⟨ k-factor string ⟩ in either of the verify or produce tasks can be one out of the following four types:

- **atomic**[1] - consists of a single word/k-factor/n-gram from the vocabulary and task of a learner is to check the existence of this k-factor in the witness (verify task). In the verification setting this task is strictly local in its description (section 2.6.2) and can be solved by a scanner with an access to the witness/lookup-table.
- **or** - consists of k-factors joined by the boolean operator 'or'. This task is again strictly local in its description albeit more memory intensive than the atomic task because it requires storage of multiple k-factors in memory.
- **and** - consists of k-factors joined by the boolean operator 'and'. Now a learner doesn't simply have to verify the existence of a single k-factor in the witness, instead for each k-factor it has to maintain a record of it's occurence/non-occurence in the witness, and thus requires store of all k-factors in memory. This task is therefore locally k-testable.
- **not** - consists of k-factors acted upon by both 'and ' as well as 'not ' operators. While the *conjunction* acts upon two k-factors, the *negation* only acts upon one. This task also has a locally k-testable description.

4.1 Data Structure

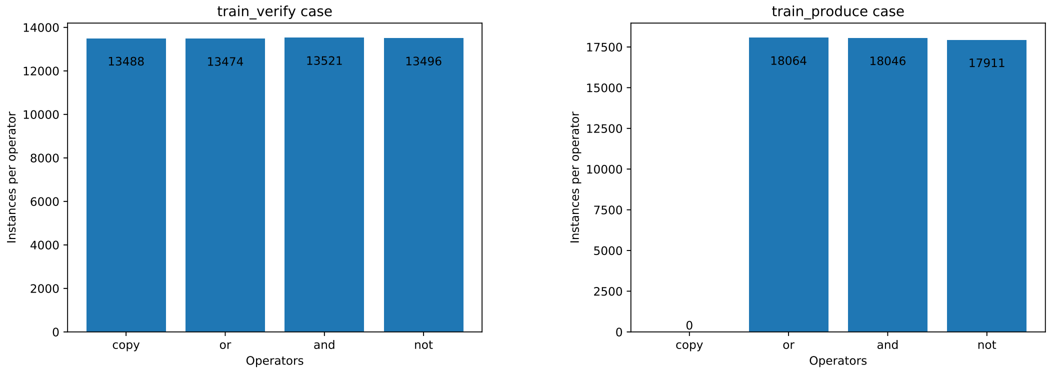
Out of the 128 ASCII characters, only 100 are printable and of those 100, 6 refer to newline, space, tab etc. which are not directly observable and therefore our vocabulary Σ consists of 94 printable ascii characters and two binary output tokens 'yes' and 'no'. The vocabulary is split into two equal disjoint subsets. The train and various test sets constructed from this vocabulary are as follows:

1. **train** consists of 120000 pairs of input output symbols with input composition length (vocabulary symbols excluding operators) ranging between ⟨ 2 and 4 ⟩ (limits inclusive). 10% of this data is used to create the validation set. Any particular input composition is constructed using words from only one of the subsets.
2. **unseen test** consists of 12000 i/o pairs with input composition length ranging between ⟨ 2 and 4 ⟩. The compositions are created by randomly sampling words from both subsets of the vocabulary i.e cross-composition between subsets is allowed
3. **longer test** consists of i/o pairs such that input composition length is between ⟨ 5 and 7 ⟩ (limits inclusive). cross-composition of words from two different subsets isn't allowed.

[1] This is present only in the verify task, since in produce it just becomes a unary k-factor without any operator and produces an output which is a copy of itself. It is like the 'and' task without any operator or additional symbols in the input.

4. **unseen longer test** consist of i/o pairs such that input composition length is between ⟨ 5 and 7 ⟩ and cross-composition is allowed.

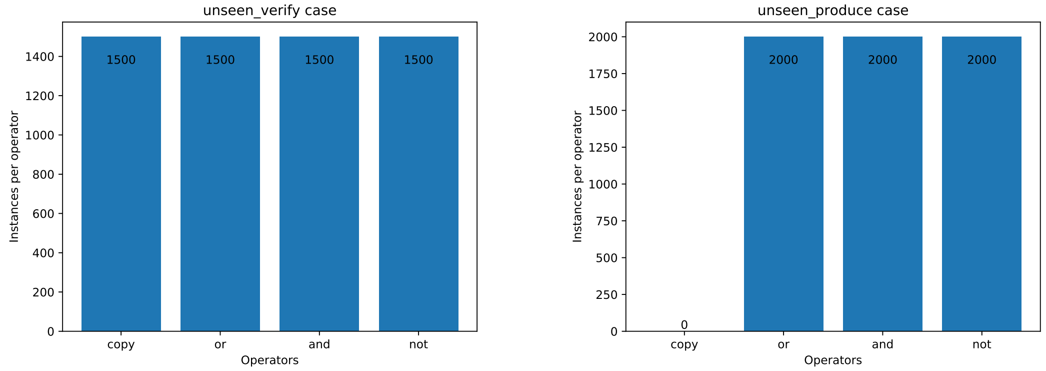
The distribution of the boolean operators (i.e. the number of tasks per operator) in the train and unseen test set (the distribution is same for every test set) can be seen in figures 4.1 and 4.2 respectively.



(a) Verification Task

(b) Production Task

Figure 4.1: Micro Tasks - Training Data



(a) Verification Task

(b) Production Task

Figure 4.2: Micro Tasks - Unseen Data

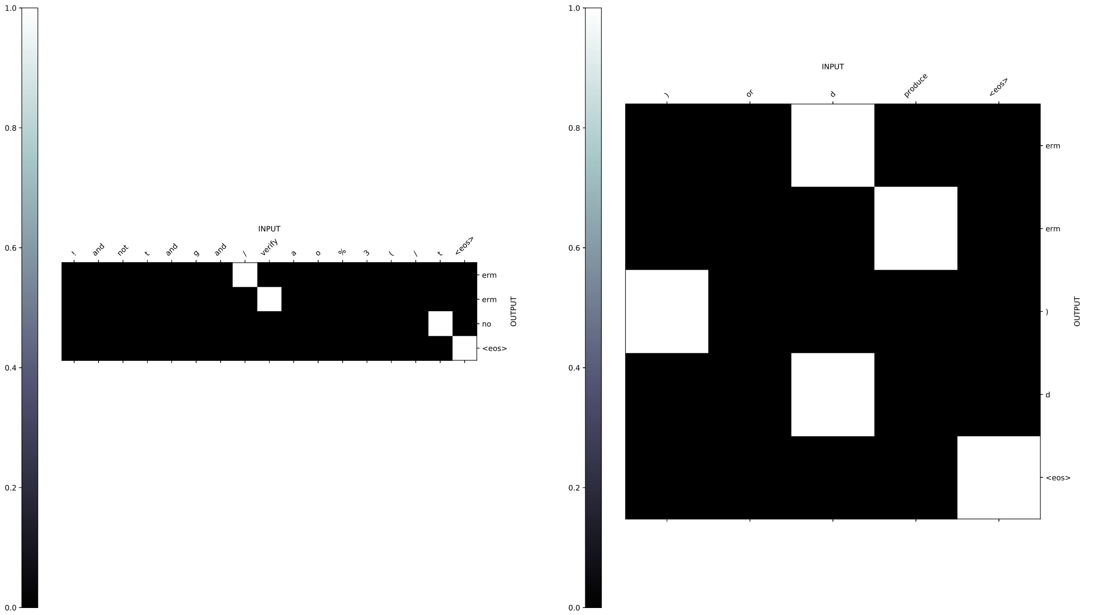
4.2 Experimental Setup

4.2.1 AG Trace

The trace for the hard guidance of micro-tasks is different for verify and produce cases owing to the fact that verify has a single emission (yes/no token) while produce has multiple emissions. Furthermore pondering is explicitly mocked here by augmenting the output with a ponder token. The ponder token helps the network in focusing on the summarization of the entire string and the meta-information i.e. verify/produce, before moving on to the emission of the actual targets. The details of the trace are as follows:

Verify. Here we have a trace of length 3 with the focus at end of string, meta-information and end of witness respectively. While the first two indices of the trace correspond to the ponder tokens in the output, the last index corresponds to the decision token which is the actual target. The trace is visualized in figure 4.3a.

Produce.In the case of produce while the ponder tokens and the trace corresponding to them remain the same as in the case of verify the trace corresponding to the emissions changes. The trace corresponds to the index in input string for an emission in output string. In the case of not, every emission in target which isn't in input is traced in ascending order of index of the words in input which are acted upon by a not operator. The trace is visualized in figure 4.3b



(a) AG Trace for Verification Task

(b) AG Trace for Production Task

Figure 4.3: Micro Tasks - Attentive Guidance Trace

4.2.2 Micro task Accuracy

Similar to the symbol rewriting dataset (section 3.3) micro tasks is a probabilistic dataset in both the number and order of emissions. Therefore, just as we had to define a symbol rewriting metric in section 3.3.3 we need to define a new accuracy measure for microtasks.

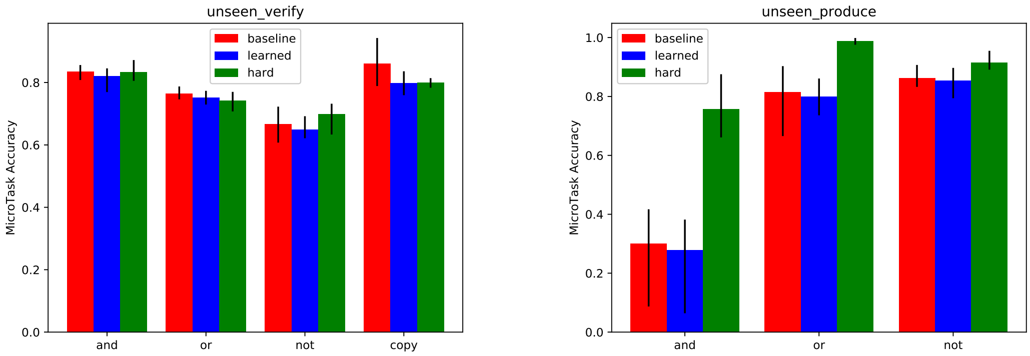
While the microtask accuracy for verify task is simply contingent on the final emission, which is binary, the accuracy for ‘or ’, ‘and ’, ‘not ’, tasks are separately defined as follows:

- **or** - by checking the presence of any of the input token in predicted sequence.
- **and** - by checking the presence of all of the input tokens in predicted sequence.
- **not** - by ensuring that the set of all the input tokens preceded by negation and the predicted sequence are disjoint. This is followed by the same test as above for all the tokens not preceded by negation.

Hyperparameters Three different kindof models- baseline(vanila seq2seq), learned guidance(attentive guidance) and hard guidance (attention vectors explicitly provided at decoding) were trained on this dataset using same hyperparameters. The hyperparameters used for the experiments were: rnn cell = lstm, hidden layer size = embedding size = 128, attention =pre-rnn [Bahdanau et al., 2014], alignment measure = mlp, optimizer=Adam [Kingma and Ba, 2014], learning rate = 0.001.

4.3 Micro Tasks - Results

Since the dataset is probabilistic in nature, it is reasonable to create multiple splits of the data and train different models on each split. I generated 5 different train and test splits of the data and trained separate baselines, learned and hard guided models on each of these splits. Each of these trained models were then used to get the accuracies on the 5 test splits. The mean of these accuracy values along with the error bars (max, min) values is presented for each operation individually in for both verify and produce tasks.

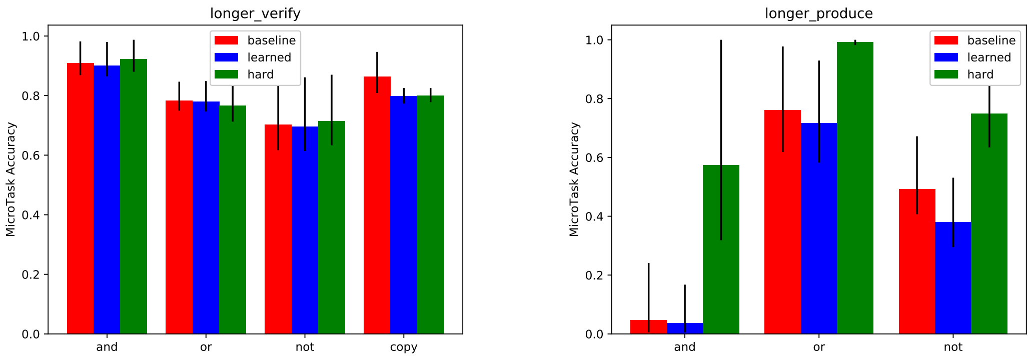


(a) Accuracies for verify task

(b) Accuracies for produce task

Figure 4.4: Micro Task accuracies per operation for unseen test

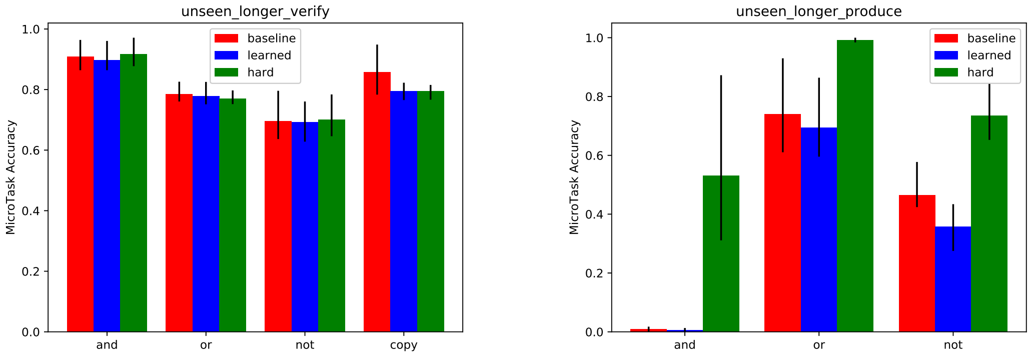
A general observation from these results is that on the verify task all the models have similar optimum values, for each of the operations. The best results are for the ‘and’ & ‘copy/atomic’ tasks. It can be argued that these two operations exhibit lower stochasticity than the remaining two operations in the sense that the symbols/morphemes that the model is expected to look for remain fixed in the witness string. This isn't the case for the 'or' & 'not' tasks and thus for a verifier 'not' is arguably the most difficult operation in the verification task. The results on each of the test splits (figures 4.4a,4.5a and 4.6a) expectedly concur with this hypothesis.



(a) Accuracies for verify task

(b) Accuracies for produce task

Figure 4.5: Micro Task accuracies per operation for longer test

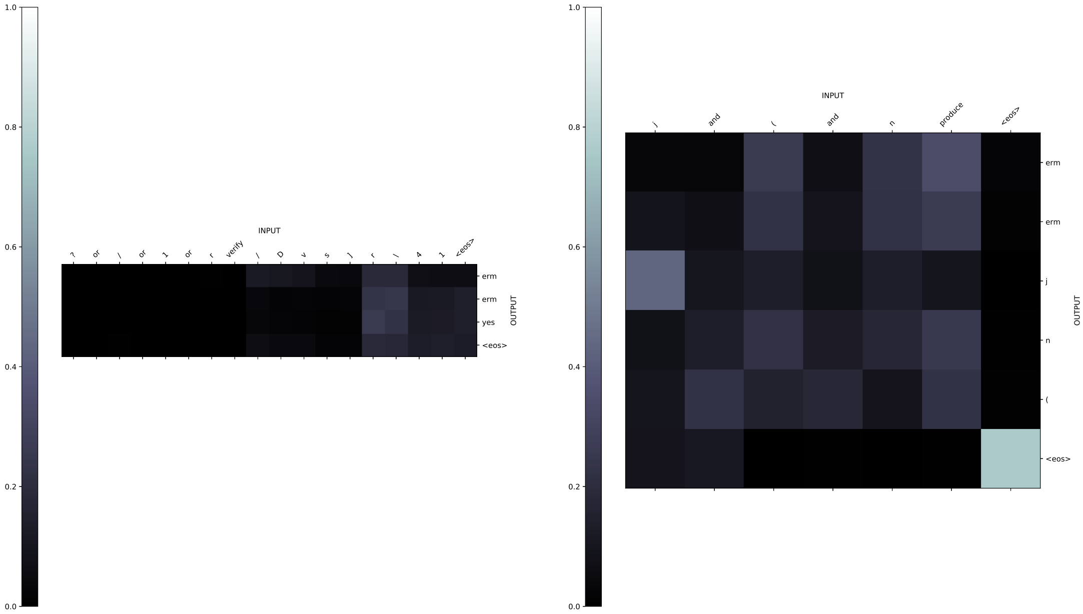


(a) Accuracies for verify task

(b) Accuracies for produce task

Figure 4.6: Micro Task accuracies per operation for unseen longer test

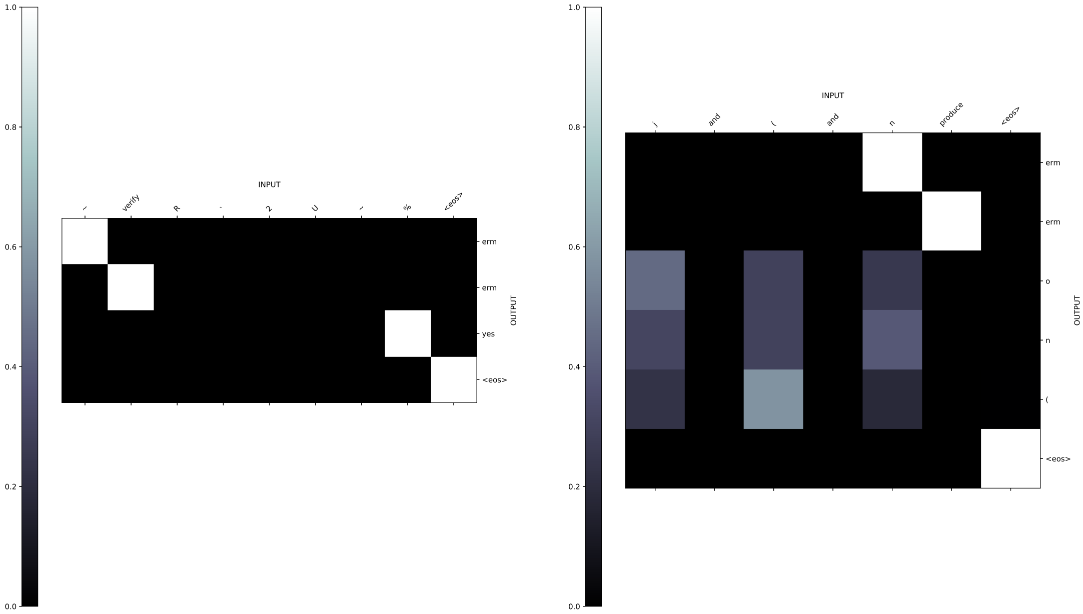
The baseline clearly fails on the production tasks, as does the learned guidance. Unsurprisingly 'or' is the easiest operation production task even for longer and unseen-longer test splits. (figures 4.4b, 4.5b and 4.6b). A probable explanation for this the fact that a model can solve an 'or' task just by emitting a subsection of the longer string and thus increasing the length is inconsequential. The reason(s) behind the models performing worse on 'and' as compared to 'not' is not clear and is left as future work.



(a) Verify

(b) Produce

Figure 4.7: Attention plots for baseline on both verify and produce tasks.



(a) Verify

(b) Produce

Figure 4.8: Attention plots for AG model on both verify and produce tasks.

4.4 Discussion

The huge chasm between the performance of the models with attentive guidance and hard guided models again brings to fore the argument I raised in section 3.4, that the current architectural setting is not optimal for learning the correct trace. This is substantiated by the attention plot for learned model in figure 4.8b. The model's attention is diffused over the three target emissions and it leads to one wrong emission in the sequence. One striking observation from the attention plot of baseline for verification task (figure 4.7a) is that it utilizes only the witness string to make its decision.

It has already been discussed in the introduction to this chapter that the language L proposed in this dataset $\notin NP$. However even for this language the results indicate that search (production) is more difficult than decision (verification) for a prover (a vanilla seq2seq model in our case). But the same vanilla seq2seq with additional hardcoded attention vectors can solve the search problem to a reasonable degree.

Chapter 5

Conclusions and Future Work

In this thesis, I have introduced the concept of attentive guidance (AG) which is a way of biasing a seq2seq model to learn in a systematic fashion. This form of systematic compositionality is at the core of how humans learn [Marcus, 2003] and gives them the ability to one shot generalize to new concepts [Lake et al., 2016]. Thanks to AG, I test the claim that when explicitly forced to focus on the primitives building up a complex expression, seq2seq models without additional modification are able to generalize to new concepts. The results presented on two datasets, one which exhibits hierarchical and functional compositionality and another which is built using the production rules of a PCFG, substantiate the aforementioned claim. The role of perfect attention in a compositional learner has further been substantiated by attempting to solve a new dataset introduced in this thesis. The dataset has been built on the concept of subregular language hierarchy (section 2.6.2) and experiments were carried out using vanilla seq2seq model and a hard guidance model. It was observed from the results obtained on this dataset that while vanilla seq2seq can solve the decision task, it failed on the more difficult search task, whereas hard guided model succeeded on both. This warrants additional investigation for effectiveness of hard guidance in languages which are higher up in the polynomial hierarchy [Arora and Barak, 2009].

However, AG comes with its own set of pitfalls and they become evident as I move towards more difficult forms of zero shot generalization (longer samples in case of lookup tables and symbol rewriting tasks). This is in contrast to the performance that is seen with hard guidance, i.e. when we explicitly provide the target attention vector as input to the decoder. This presents a case for using alternative paradigms for learning the guidance, as compared to supervised learning. The decoder can be tasked with finding the ideal attention vectors from a controlled search space of possible attention vectors in a reinforcement learning [Sutton et al., 2018] setting. Similarly, the pondering that is currently 'mocked' in some cases (sections 3.2.2 4.2.1)by AG, can be learned and this would make the decoder more adaptive to different sequence lengths or unseen input prompts by allowing it to ponder over them for more steps. In the current architectural setup, the burden of learning the attention falls on the decoder while the encoder remains the same as in the case of a vanilla seq2seq model. Another possible area of investigation could be tasking the encoder with presenting the information to the decoder in the most efficient fashion or by allowing dynamic encoder outputs based on an on-line loss feedback from the decoder at each step.

Finally, revisiting the information retrieval viewpoint presented by Vaswani et al. [2017], it can be argued that AG leads to compositional solutions via efficient information retrieval, i.e. giving the decoder access to the most informative encoder outputs at each step. However, another salient feature of human beings is their capacity for learning rich representation and then transferring that knowledge to new concepts [Lake et al., 2015]. Therefore, a parallel line of research could be creating latent space representations in encoder decoder models, which are more in sync with the structure of the data. Recently, Nickel and Kiela [2017] proposed embedding hierarchical data such as complex networks into hyperbolic (Poincaré ball) spaces to better capture the tree like structure in the data. They showed that Poincaré embeddings outperform Euclidean embeddings on data exhibiting latent hierarchies. Capturing the latent hierarchies of a data in a non-euclidean latent space presents an interesting line of research for (compositional) representation learning.

References

Jiasen Lu, Caiming Xiong, Devi Parikh, and Richard Socher. Knowing when to look: Adaptive attention via a visual sentinel for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 6, page 2, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. 2014a. ISSN 09205691. doi: 10.3115/v1/D14-1179.

Kyunghyun Cho, Bart Van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation : Encoder Decoder Approaches. *Ssst-2014*, pages 103–111, 2014b. doi: 10.3115/v1/W14-4012.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet: A large-scale hierarchical image database. *Cvpr*, pages 248–255, 2009. ISSN 1063-6919. doi: 10.1109/CVPRW.2009.5206848.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building Machines That Learn and Think Like People. *Behavioral and Brain Sciences*, (2012):1–101, 2016. ISSN 14691825. doi: 10.1017/S0140525X16001837.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. ISSN 10959203. doi: 10.1126/science.aab3050.

Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. 2017.

Christopher Olah. Understanding LSTM Networks – colah's blog, 2015. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. IT Pro. Cambridge University Press, 2009. ISBN 9780511533815.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Dec 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.4.541.

G. Frege and J.L. Austin. *The Foundations of Arithmetic: A Logico-Mathematical Enquiry Into the Concept of Number*. Northwestern University Press, 1968. ISBN 9780810106055.

I. Neath and A. Surprenant. *Human Memory*. Cengage Learning, 2013. ISBN 9781111834807.

JL Elman. Finding structure in time* 1. *Cognitive science*, 211(1 990):1–28, 1990. ISSN 0364-0213.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, may 2015. ISSN 0028-0836.

Noah Weber, Leena Shekhar, and Niranjan Balasubramanian. The Fine Line between Linguistic Generalization and Failure in Seq2Seq-Attention Models. 2018.

Adam Liška, Germán Kruszewski, and Marco Baroni. Memorize or generalize? Searching for a compositional RNN in a haystack. 2018. doi: 10.1145/nnnnnnn.nnnnnnn.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. 2015. ISSN 10495258. doi: 10.18653/v1/D15-1166.

E.T.F.I.M. Craik, E. TULVING, F.I.M. Craik, Oxford University Press, and University of Cambridge. *The Oxford Handbook of Memory*. Oxford Library of Psychology Series. Oxford University Press, 2000. ISBN 9780195122657.

Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3 – 71, 1988. ISSN 0010-0277. doi: https://doi.org/10.1016/0010-0277(88)90031-5.

Zoltn Gendler Szabo. Compositionality. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2017 edition, 2017.

Huiting Zheng, Jiabin Yuan, and Long Chen. Short-Term Load Forecasting Using EMD-LSTM neural networks with a xgboost algorithm for feature importance evaluation. *Energies*, 10(8), 2017. ISSN 19961073. doi: 10.3390/en10081168.

Gerhard Jäger and James Rogers. Formal language theory: Refining the Chomsky hierarchy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367(1598): 1956–1970, 2012. ISSN 14712970. doi: 10.1098/rstb.2012.0077.

G Miller. The Magic Number Seven, Plus or Minus Two: Some Limits on our Capcity for Processing Information. *The Psychological Review*, 63(2):81–97, 1956.

D. Purves, R. Cabeza, S. Huettel, K. LaBar, M.L. Platt, and M. Woldorff. *Principles of Cognitive Neuroscience*. Sinauer, 2013. ISBN 9780878935734.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. ISSN 00368075. doi: 10.1126/science.1127647.

G.F. Marcus. *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. A Bradford book. MIT Press, 2003. ISBN 9780262632683.

R.S. Sutton, A.G. Barto, and F. Bach. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, 2018. ISBN 9780262039246.

Gary Marcus. Deep Learning: A Critical Appraisal. *arXiv preprint arXiv:1801.00631*, pages 1–27, 2018.

Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. pages 1–19, 2016. ISSN 0927-7099. doi: 10.475/123.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013.

Eric Schulz, Josh Tenenbaum, David K Duvenaud, Maarten Speekenbrink, and Samuel J Gershman. Probing the Compositionality of Intuitive Functions. In D D Lee, M Sugiyama, U V Luxburg, I Guyon, and R Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3729–3737. Curran Associates, Inc., 2016.

Paul J. Werbos. Backpropagation Through Time: What It Does and How to Do It. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. ISSN 15582256. doi: 10.1109/5.58337.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. ISSN 14764687. doi: 10.1038/nature16961.

Marco Baroni, Armand Joulin, Allan Jabri, Germàn Kruszewski, Angeliki Lazaridou, Klemen Simonic, and Tomas Mikolov. CommAI: Evaluating the first steps towards a useful general AI. pages 1–9, 2017.

Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learning long-term dependencies in recurrent networks. *IEEE International Conference on Neural Networks - Conference Proceedings*, 1993-January:1183–1188, 1993. ISSN 10987576. doi: 10.1109/ICNN.1993.298725.

Dieuwke Hupkes, Anand Singh, Kris Korrel, German Kruszewski, and Elia Bruni. Learning compositionally through attentive guidance. 2018.

Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks arXiv:1311.2901v3 [cs.CV] 28 Nov 2013. *Computer VisionECCV 2014*, 8689: 818–833, 2014. ISSN 978-3-319-10589-5. doi: 10.1007/978-3-319-10590-1_53.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Maximilian Nickel and Douwe Kiela. Poincar\'e Embeddings for Learning Hierarchical Representations. 2017.

N. Chomsky. Formal properties of language. In D. Luce, editor, *Handbook of Mathematical Psychology*, page 2. John Wiley & Sons., 1963.

Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956. ISSN 21682712. doi: 10.1109/TIT.1956.1056813.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June-2015:3156–3164, 2015. ISSN 10636919. doi: 10.1109/CVPR.2015.7298935.

Mihir Bellare and Shafi Goldwasser. The Complexity of Decision Versus Search. *SIAM Journal on Computing*, 23(1):97–119, 1994. ISSN 0097-5397. doi: 10.1137/S0097539792228289.

Justin Johnson, Li Fei-Fei, Bharath Hariharan, C. Lawrence Zitnick, Laurens Van Der Maaten, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January:1988–1997, 2017. doi: 10.1109/CVPR.2017.215.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. (Nips), 2017. ISSN 0140-525X. doi: 10.1017/S0140525X16001837.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks. (2), 2012. ISSN 1045-9227. doi: 10.1109/72.279181.

D. Jurafsky, J.H. Martin, P. Norvig, and S. Russell. *Speech and Language Processing*. Pearson Education, 2014. ISBN 9780133252934.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. ISSN 0027-8424. doi: 10.1073/pnas.79.8.2554.

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Manish Chablani. Sequence to sequence model: Introduction and concepts, 2017. URL https://towardsdatascience.com/@ManishChablani.

Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 114–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015425.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. pages 1–15, 2014. ISSN 0147-006X. doi: 10.1146/annurev.neuro.26.041002.131047.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation (No. ICS-8506). *California Univ San Diego La Jolla Inst For Cognitive Science*, 1:318–362, 1986. ISSN 1-55860-013-2. doi: 10.1016/B978-1-4832-1446-7.50035-2.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, 2014. ISSN 09205691. doi: 10.1007/s10107-014-0839-0.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602. doi: 10.1007/BF02478259.

F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books, 1962.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012. ISSN 10495258. doi: http://dx.doi.org/10.1016/j.protcy.2014.09.007.

Sepp Hochreiter and Jurgen Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1–32, 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

Appendix A

Algorithm for balancing lookup tables train set

Since the dataset for the lookup tables is small – even if length 2 compositions was created using tables *t1* to *t8*, there could only be 64 compositions which would bijectively map 8 '3 bit inputs to 3 bit outputs' thereby leading to a total of 512 samples only – we try to ensure that the compositions present in the training data are uniformly distributed while simultaneously keeping the distribution of the output strings in the training data uniform as well. This ensures that the model doesn't get biased towards any particular composition or an output string. As has been explained in section 3.2.1, the heldout inputs set is created by randomly taking out 2 inputs for each of the 28 compositions in training. Therefore the resultant training set should have the following properties:

- The total number of data points $= 28 * 6 = 168$.
- There are 8 outputs. The total number of compositions that lead to a particular output $= 168/8 = 21$.

.

Algorithm 1 Create training with uniform distribution of both compositions and outputs

```
We start with 28 compositions with 8 inputs each.
Create an empty dataframe of dimension (21,8) with the 8 output strings as it's columns.

Create a dictionary Y with compositions as it's keys and values=0.
for i in range [0, 21) do
  for output in dataframe column do
    Sample composition
    if composition not in row[i] AND composition not in output column AND
    Y[composition] < 21 then
      dataframe[i][output] = composition
      Y[composition] += 1
    else
      continue
    end if
  end for
end for
```