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
Department Of
Computational Science

The undersigned hereby certify that they have read and recommend to the Department of Computational Science for acceptance a thesis titled **“Pondering in Artificial Neural Networks”** by **Anand Kumar Singh** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: Thursday 16th August, 2018

Supervisor: ______________________
Dr. ir. Elia Bruni

Supervisor: ______________________
Ir. Dieuwke Hupkes

Abstract

Abstract

Acknowledgements

Acknowledge

Amsterdam, The Netherlands
Thursday 16th August, 2018

Anand Kumar Singh

Contents

List of Figures

List of Tables

Nomenclature

Latin Symbols

μ_f	dynamic viscosity of fluid
ν_f	kinematic viscosity of fluid
ρ_f	density of fluid
ρ_f	density of particle
$C_{p,f}$	specific heat capacity of fluid
$C_{p,p}$	specific heat capacity of particle
k_f	thermal conductivity of fluid

Chapter 1

Introduction

1.1 Motivation

Motivation

1.2 Objectives

Objectives

1.3 Outline

Thesis outline

Chapter 2

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.

2.1 RNN - A non linear dynamical system

We frequently encounter data that is temporal in nature. A few obvious examples would be, audio signals, videos signals, time series of a stock price and natural language (the holy grail of AI!). While traditional feed-forward neural networks 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. Human beings solve such problems by compressing and storing the previous states in memory and then using that information for predicting the state T_t. A conventional feed-forward network such as a MLP is inherently memoryless.

Recurrent neural networks overcome 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 very 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. Fig 2.1 shows a rolled and unrolled recurrent network.

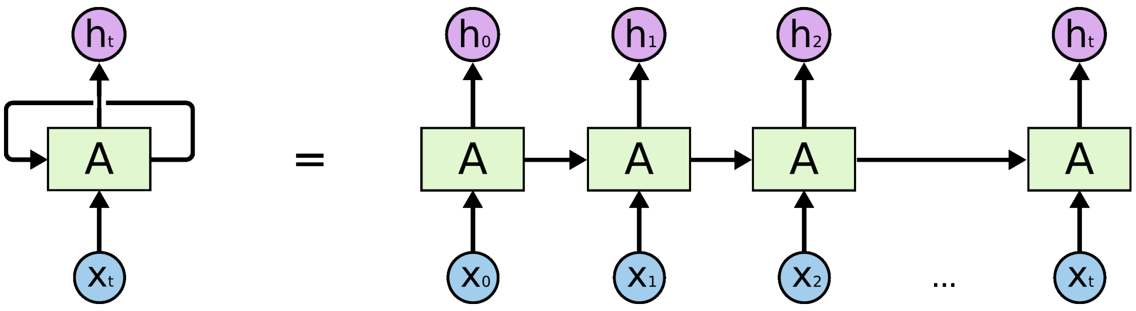


Figure 2.1: Schematic of a RNN Olah [2015]

2.1.1 Non linear dynamics inside a RNN cell

equations
presence of strange attractors.

$$h^{(t)} = f(x^{(t)}, h^{(t-1)}) \tag{2.1}$$

$$c_t = tanh(Ux_t + Wc_{t-1}) \tag{2.2}$$

$$y_t = sotmax(Vc_t) \tag{2.3}$$

2.2 BPTT and Vanishing Gradients

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) 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

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\sum_t (y_t \log \hat{y}_t) \tag{2.4}$$

Since the weight W_{oh} is shared across all time steps, adding the derivatives across the sequence we get:

$$\frac{\partial \mathcal{L}}{\partial W_{oh}} = \sum_t \frac{\partial \mathcal{L}}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial W_{oh}} \tag{2.5}$$

Now for time-step t $\rightarrow$ t+1:

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

Again since W_{hh} is shared across time steps, the gradient at t+1 will have contribution from the previous time steps as well. Summing over all the time steps we get:

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

Aggregating over the whole sequence we get:

$$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \sum_{t} \sum_{\tau=1}^{t+1} \frac{\partial \mathcal{L}(t+1)}{\partial \hat{y}_{t+1}} \frac{\partial \hat{y}_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_{\tau}} \frac{\partial \mathbf{h}_{\tau}}{\partial W_{hh}} \tag{2.8}$$

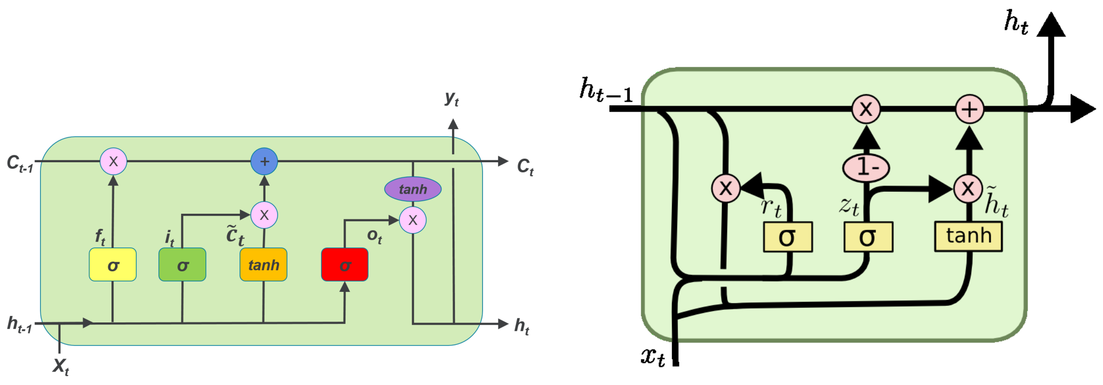
Looking at equation 2.8 we can see that the RNN gradient is a recursive product of $\frac{\partial h_t}{\partial h_{t-1}}$ and **in the event of this derivative being $\ll 1$ or $\gg 1$, the RNN gradient would vanish or explode respectively** when the network is trained over longer time-steps. In the former case the training would freeze while in the latter it would never converge. Therefore a vanilla RNN can't keep track of long term dependencies which is critical for tasks such as speech synthesis, music composition etc. 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

Looking at equation 2.1 we can see that RNN is an example of an iterated function and can therefore exhibit complex and chaotic behavior(substantiate this in later sections). 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 (rephrase this and substantiate with equation/previous equations). Now while an identity initialization of recurrent weights by itself isn't very 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.

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.



(a) LSTM **(b)** GRU

Figure 2.2: Gated RNNs

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

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

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

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

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

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

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

The hidden state of a LSTM cell is the output which is then multiplied with the output weights followed by the softmax operation in order to yield the final output. The cell state is the memory of the LSTM unit. Furthermore the sigmoid activation for each one of the input, forget and output gates serves as a switch such that multiplying them element wise with other vector decides how much of the vector to filter out.

- **input modulation gate $g^{(t)}$:** The input modulation gate is computed based on the present input and the previous hidden state (which is exposed to the output). It yields a candidate memory for the cell state. Since it doesn't determine how much of a particular information to retain, it doesn't have the sigmoid activation. The tanh layer serves the purpose of creating candidate memories for the new cell state.

- **input gate $i^{(t)}$:** The input first computes a new cell state based on the new input and the previous hidden state and then decides how much of this information to "let through". A sum of the hadamard products of the (input gate,modulation gate) and (forget gate, previous cell state) tuples respectively yields the new cell state.
- **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 $o^{(t)}$:** A tanh on the new cell state yields the candidate hidden state. The output gate then determines determines how much of this internal memory to expose to the top layers (and subsequent timesteps) of the network.

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:

$$m_t = (1 - z_t)m_{t-1} + z_t\widetilde{m_t} \tag{2.15}$$

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

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

$$r_t = \sigma(x_t U^{(r)}, m_{t-1}W^{(r)}) \tag{2.18}$$

- **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.
- **modulation gate $g_{(t)}$:** Just as in the case of the LSTM the modulation gate serves the purpose of yielding candidate memories for the new cell state.
- **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.
- **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 complex in comparison to a LSTM.

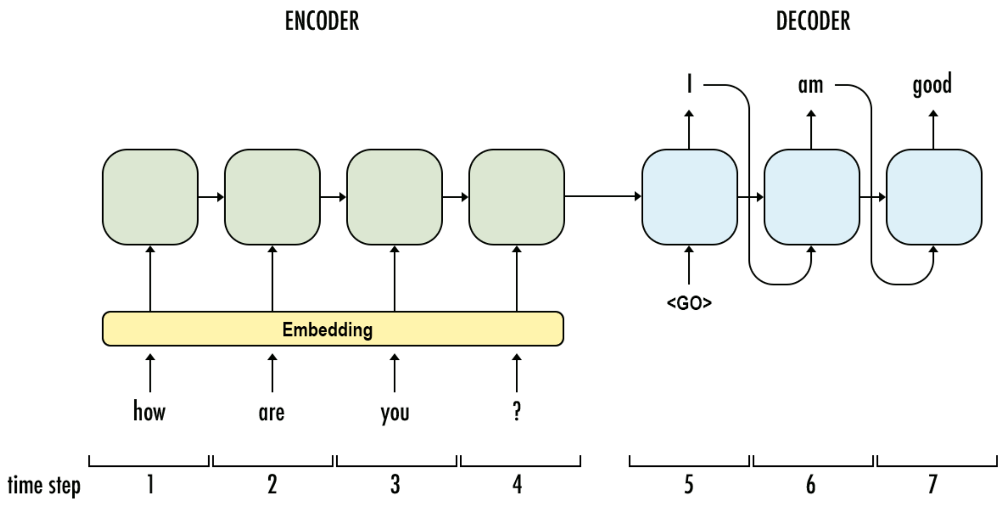


Figure 2.3: Schematic of a Seq2Seq

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, text summarization etc.

Neural networks have been shown to be excellent at learning rich representation from data without the need for extensive feature engineering (give some refs here). 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.

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

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

Chapter 3

Motivation

What I am actually looking at - Compositionality, Problems of RNN- SCAN, Lookup Tables. Why should it be compositional.

Pondering should be here as well.

3.1 Systematic Compositionality

Natural language and human reasoning arguably exhibit compositionality. Compositionality is the principle of understanding a complex expression through the meaning and syntactic combination of its morphemes. One of the strongest arguments in favor of compositionality is the presence of systematicty in natural language.

Systematicity. Provided [E1, E3] of same grammatical category and [E2,E4] of same grammatical category. If [E1,E2] can combine syntactically then so can [E1, E4], [E3, E2] and [E3,E4] and if one can understand ([E1,E2]) and ([E3, E4]) then they can also understand ([E1,E4]) and ([E3,E2]) provided that the is well formed.

The definition of compositionality presented above does not directly imply dependence on 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 [Szab, 2017]. Aritifical languages (such as the ones we'll encounter in chapter 4) on the other hand can be constructed to strictly follow the principles of systematic compositionality. Given human propensity for compositional solutions [Schulz et al., 2016] these datasets provide excellent test-beds for validating the existence of compositional skills in neural networks.

3.1.1 Compositionality in Seq2Seq Models

Lake and Baroni concluded that seq2seq is still doesn't exhibit systematicity.

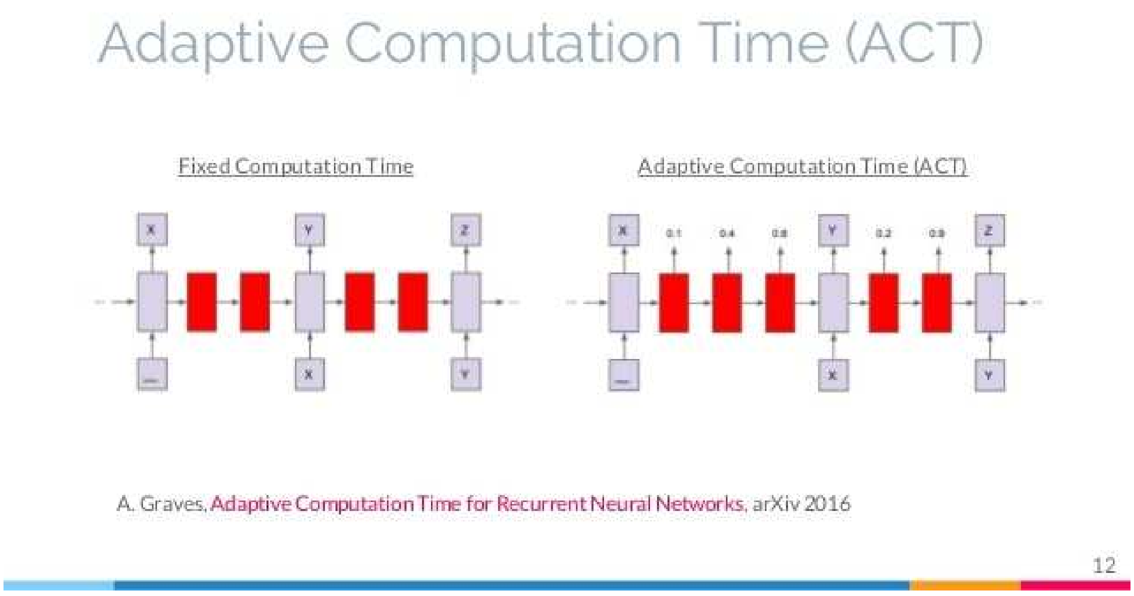

Adaptive Computation Time (ACT)
Fixed Computation Time
Adaptive Computation Time (ACT)
X
Y
Z
_
X
Y
0.1
0.4
0.8
0.2
0.8
A. Graves, Adaptive Computation Time for Recurrent Neural Networks, arXiv 2016
12

Figure 3.1: Adaptive Computation Time

Zero, One and Few shot generalization

3.2 Attention

3.2.1 Attention in Humans

Attention from a cognitive neuroscience perspective. Attentional blink

3.2.2 Se2Seq with Attention

It was shown by Cho et al. [2014b] that the performance of a basic encoder-decoder models as explained in 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, 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 NMT leads to improved performance.

How is attention calculated? Equations? Plots? Also cover this from a key-value pair pov.

3.3 Pondering

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 very 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 of 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:

$$\begin{aligned} h_t &= f(Ux_t + Wc_{t-1}), \\ y_t &= g(Vc_t). \end{aligned} \tag{3.1}$$

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:

$$\begin{aligned} c_t^n &= \begin{cases} f(Ux_t^1 + Wc_{t-1}) \text{ if } n = 1 \\ f(Ux_t^n + Wc_t^{n-1}) \text{ if } n \neq 1 \end{cases}, \\ y_t^n &= g(Vc_t^n). \end{aligned} \tag{3.2}$$

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:

$$\begin{aligned} h_t^n &= \sigma(Sc_t^n), \\ p_t^n &= \begin{cases} R(t) \text{ if } n = N(t) \\ h_t^n \text{ if } n \neq N(t) \end{cases}, \end{aligned} \tag{3.3}$$

where:

$$\begin{aligned} N(t) &= min\{m : \sum_{n=1}^{m} h_t^n \geq 1 - \epsilon\}, \\ R(t) &= 1 - \sum_{n=1}^{N(t)-1} h_t^n, \end{aligned} \tag{3.4}$$

and ϵ is a small constant.

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

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 3.1 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.

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:

$$\rho_t = N(t)/+\ R(t) \tag{3.5}$$

$$\begin{aligned} \mathcal{P}(x) &= \sum_{t=1}^{T} \rho_t, \\ \widetilde{\mathcal{L}}(x, y) &= \mathcal{L}(x, y) + \tau \mathcal{P}(x), \end{aligned} \tag{3.6}$$

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

3.4 Formal Language Theory

The field of formal language theory (FLT) concerns itself with the syntactic structure of natural language without much emphasis on the semantics. More precisely a formal language L is a sequence 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 the vocabulary as previously defined, NT is the set of non-terminals, S the start symbol and R the set of rules. A rule can be expressed as $a \rightarrow b$ and can be understood as a substitution of a with b with a, b coming from Σ and/or NT. 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.

3.4.1 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 3.2. The different classes of grammar are

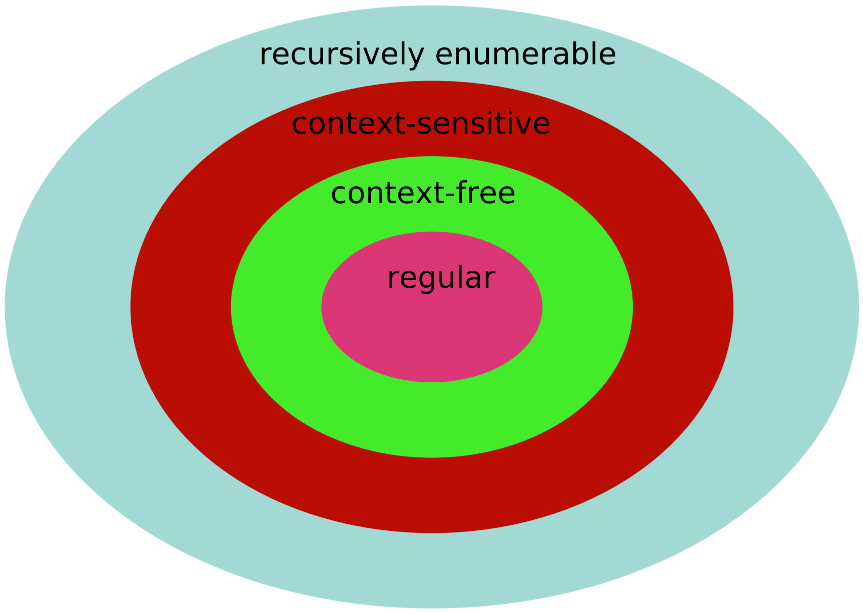


Figure 3.2: Chomsky Hierarchy

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 them. 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. Any grammar decidable by an unrestricted Turning machine with no constraints on the production rules $a \rightarrow b$. (Compare to halting problem.)

Context-Sensitive. (Don't understand too well yet.)

Context-Free. Such a grammar is described by production rules of the form $A \rightarrow \alpha$ where $A \in NT$ and $\alpha \in (\Sigma \cup NT)$. Context free grammar lead to context free languages (CFL) which are hierarchical in structure, albeit 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 psuh 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. Characterized by production rules of the form $A \rightarrow \alpha; A \rightarrow \alpha B$ 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 a FSA.

3.4.2 Subregular Hierarchy

The simplest class of languages encountered in section 3.4.1 were regular languages that can be described using a FSA. 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, is included in the aforementioned lookup-table.

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 won window size K slides over the string and records for every k-factor in vocabulary it's occurrence or nonoccurence in the string. The output of this scanner is then fed to a boolean network which verifies the k-expressions.

Chapter 4

Datasets

4.1 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 $001t1t2 = 010$. Since the i/o strings are three bit, there can be a maximum of 8 input/output strings.

As is very 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 3.1. 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.

4.1.1 Data Structure

We 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 we want our model to not simply memorize the compositions but rather to land on a compositional solution, we 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 4.1

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 balance w.r.t. the compositions as well as the output strings. (Details of this sudoku algorithm in the appendix)

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 we ensure 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.

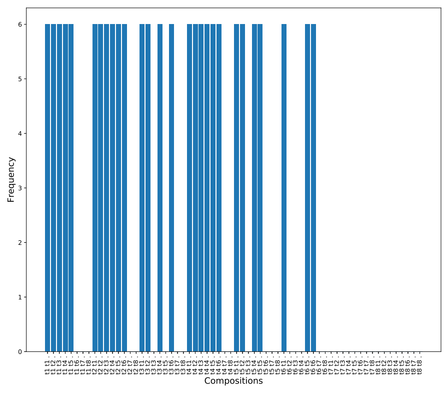
	Example	Size
train	t1 t2 011	232
heldout inputs	t1 t2 001	40
heldout compositions	t1 t3 110	64
heldout tables	t1 t8 111	192
new compositions	t7 t8 101	32

Table 4.1: Lookup Table Splits

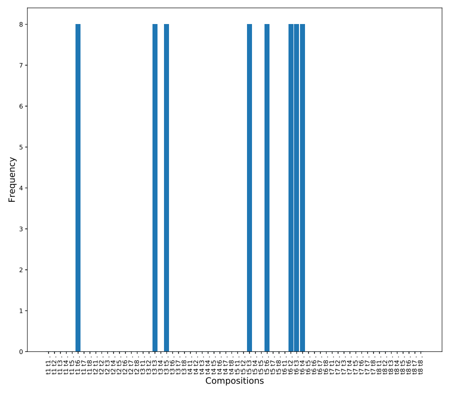
In accordance with the data split described above we present the distribution of all compositions in the train and various test sets. It can be seen that the test sets 'heldout_tables' and 'new_compositions ' are the most difficult and require zero-shot generalisation owing to their significantly different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

4.2 Symbol Rewriting

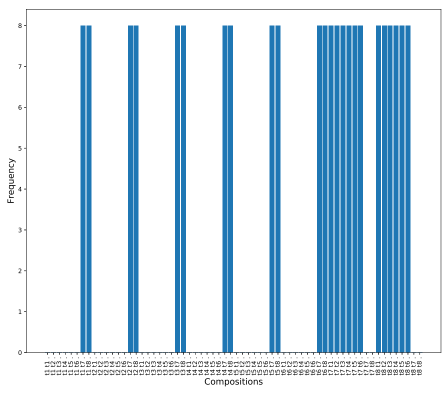
Introduced by Weber et al. [2018] the symbol rewriting dataset is essentially a probabilistic context free grammar (PCFG). It consists of a rewriting a set of input symbols to a set of output symbols based on this grammar. Before proceeding further with the task description,I'll elaborate on PCGFs briefly.



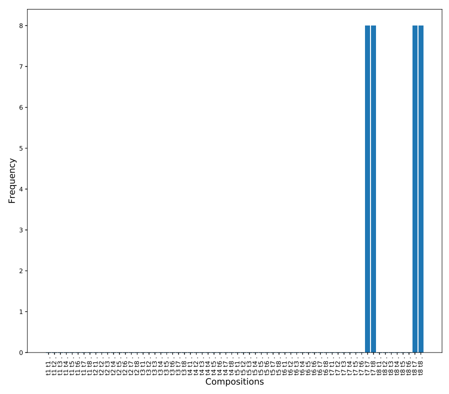
(a) Train



(b) Heldout Compositions



(c) Heldout Tables



(d) New Compositions

Figure 4.1: Data distribution of train and test sets

PCFGs are the stochastic version of CFGs (Context free grammars) that we encountered in section 3.4.1. The addition of this stochasticity was motivated by the non-uniformity of words in natural language. Assigning probabilities to production rules, lead to a grammar more in line with the Zipfian distribution of words in natural language. A PCFG consists of:

1. A CGF $\mathcal{G} = \langle \Sigma, N, S, R \rangle$ where the symbols have the same meaning as defined in **??**.
2. A probability parameter $p(a \to b) \mid \sum_{a \to b | a \in N} p(a \to b) = 1$
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 **??** illustrates the production rule for one input symbol. The grammar consists of 41 such symbols each following similar production rules. Weber et al. [2018] showed using this dataset while 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) that are atypical of the training distribution while still conforming to the underlying grammar. They however show that this indeed **is not** the case.

4.2.1 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 4.2

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 4.2: Symbol Rewriting Splits

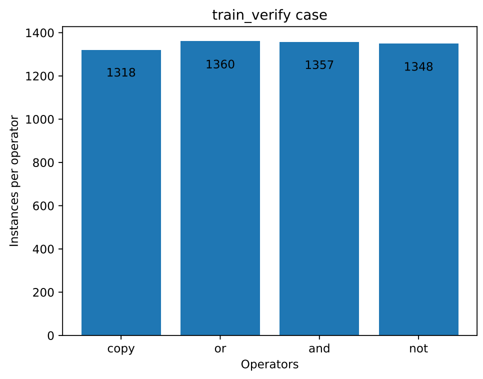
The datasets *repeat, short and long* come from distributions which are different from the training distribution on which the model has learned the data structure. It is expected of a compositional learner that given the sufficient size of the training data it would be able to infer a pattern which is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

4.3 Micro Tasks

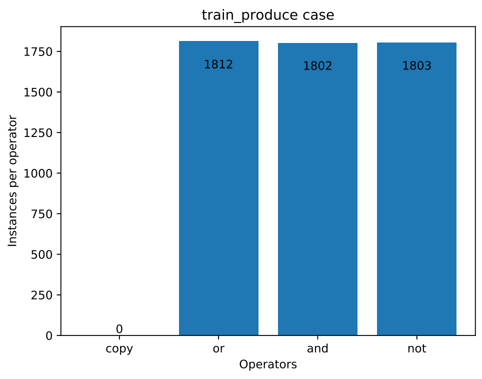
CommAI mini tasks [Baroni et al., 2017] are a set of strictly local and locally testable (please refer to section 3.4.2) languages designed to test the generalization capabilities of a compositional learner. Based on the mini tasks we propose a new dataset of sub-regular languages which we call he Micro Tasks. The language encompasses a vocabulary of all printable ascii characters, except printable but invisible characters i.e. tab, space, newline etc.

- Define verification and production tasks
- The vocabulary is acted upon by the boolean operators (or, and and not) in the case of both verification and production tasks. In case of verification we have an additional operation copy.
- Describe as in CommAI how copy and or are strictly local while and+not constitute locally k testable languages.

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.

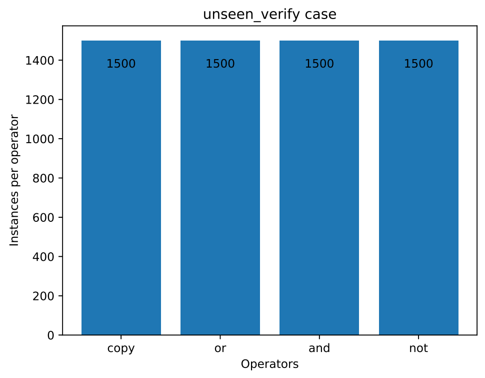


(a) Verification Task

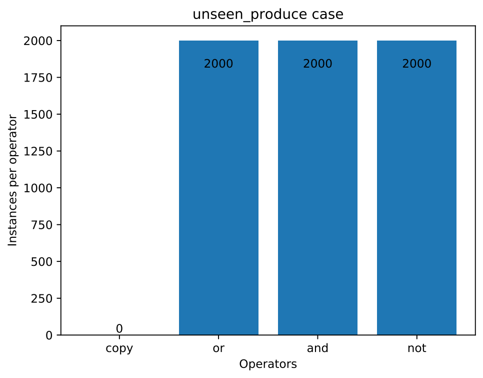


(b) Production Task

Figure 4.2: Micro Tasks - Training Data



(a) Verification Task



(b) Production Task

Figure 4.3: Micro Tasks - Unseen Data

Chapter 5

Proposed Models

The biggest criticism of deep learning is its overt dependence on voluminous data. For human level concept learning and generalization, deep learning models need to become more data efficient. I would like to break down the task of compositionality into two parallel sub-tasks as follows:

Storing Information meaningfully and efficiently

1. A more efficient data representation.
2. The new RNN cell designed with efficient memory allocation considerations.
3. Information Compression.

Retrieving Information efficiently

1. Attentive Guidance
2. Pondering
3. Curriculum Learning (Understander Executor).

Lake's work as an inspiration should come here, something about learning the trace of a program. What Liska et al showed on lookup tables.

Chapter 6

Experimental Setup

Set up to solve lookup tables tasks with attentive guidance. Set up to show how symbol rewriting was solved with attention guidance. Set up to solve MicroTasks with baseline. Baseline solved decision problem but not search. Trace for MicroTasks Trace for SCAN.

Maths. Merge with previous chapter

Chapter 7

Results and Discussions

Results

Chapter 8

Conclusions and Future Work

8.1 Conclusions

AG works learning AG is hard

8.2 Future Work

Understander executor Representation learning Latent space

References

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. URL http://arxiv.org/abs/1406.1078.

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.

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

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

Noah Weber, Leena Shekhar, and Niranjan Balasubramanian. The Fine Line between Linguistic Generalization and Failure in Seq2Seq-Attention Models. 2018. URL http://arxiv.org/abs/1805.01445.

Adam Liška, Germán Kruszewski, and Marco Baroni. Memorize or generalize? Searching for a compositional RNN in a haystack. 2018. doi: 10.1145/nnnnnnn.nnnnnnn. URL http://arxiv.org/abs/1802.06467.

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. URL http://arxiv.org/abs/1508.04025.

Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. pages 1–19, 2016. ISSN 0927-7099. doi: 10.475/123. URL http://arxiv.org/abs/1603.08983.

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. URL `http://papers.nips.cc/paper/6130-probing-the-compositionality-of-intuitive-functions.p`

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. URL `http://arxiv.org/abs/1701.08954`.

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.

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. URL `http://arxiv.org/abs/1409.0473`.

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. URL `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural`.

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. URL `http://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735`.