DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD
DASH.HARVARD.EDU

HARVARD LIBRARY

LSTM Networks Can Perform Dynamic Counting

Citation

Suzgun, Mirac, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M. Shieber. 2019. LSTM Networks Can Perform Dynamic Counting. Proceedings of the ACL 2019 Workshop on Deep Learning and Formal Languages, Florence, Italy, August 2, 2019.

Link

http://nrs.harvard.edu/urn-3:HUL.InstRepos:40827356

Terms of use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material (LAA), as set forth at
https://harvardwiki.atlassian.net/wiki/external/NGY5NDE4ZjgzNTc5NDQzMGIzZWZhMGFlOWI2M2EwYTg

Accessibility

https://accessibility.huit.harvard.edu/digital-accessibility-policy

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. Submit a story

LSTM Networks Can Perform Dynamic Counting

Mirac Suzgun[1] **Sebastian Gehrmann**[1] **Yonatan Belinkov**[12] **Stuart M. Shieber**[1]

[1] Harvard John A. Paulson School of Engineering and Applied Sciences
[2] MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA, USA
{msuzgun@college,{gehrmann,belinkov,shieber}@seas}.harvard.edu

Abstract

In this paper, we systematically assess the ability of standard recurrent networks to perform dynamic counting and to encode hierarchical representations. All the neural models in our experiments are designed to be small-sized networks both to prevent them from memorizing the training sets and to visualize and interpret their behaviour at test time. Our results demonstrate that the Long Short-Term Memory (LSTM) networks can learn to recognize the well-balanced parenthesis language (Dyck-1) and the shuffles of multiple Dyck-1 languages, each defined over different parenthesis-pairs, by emulating simple real-time k-counter machines. To the best of our knowledge, this work is the first study to introduce the shuffle languages to analyze the computational power of neural networks. We also show that a single-layer LSTM with only one hidden unit is practically sufficient for recognizing the Dyck-1 language. However, none of our recurrent networks was able to yield a good performance on the Dyck-2 language learning task, which requires a model to have a stack-like mechanism for recognition.

1 Introduction

Recurrent Neural Networks (RNNs) are known to capture long-distance and complex dependencies within sequential data. In recent years, RNN-based architectures have emerged as a powerful and effective architecture choice for language modeling (Mikolov et al., 2010). When equipped with infinite precision and rational state weights, RNN models are known to be theoretically Turing-complete (Siegelmann and Sontag, 1995). However, there still remain some fundamental questions regarding the practical computational expressivity of RNNs with finite precision.

Weiss et al. (2018) have recently demonstrated that Long Short-Term Memory (LSTM) models (Hochreiter and Schmidhuber, 1997), a popular variant of RNNs, can, theoretically, emulate a simple real-time k-counter machine, which can be described as a finite state controller with k separate counters, each containing integer values and capable of manipulating their content by adding ± 1 or 0 at each time step (Fischer et al., 1968). The authors further tested their theoretical result by training the LSTM networks to learn $a^n b^n$ and $a^n b^n c^n$. Their examination of the cell state dynamics of the models exhibited the existence of simple counting mechanisms in the cell states. Nonetheless, these two formal languages can be captured by a particularly simple form of automaton, a deterministic one-turn two-counter automaton (Ginsburg and Spanier, 1966). Hence, there is still an open question of whether the LSTMs can empirically learn to emulate more general finite-state automata equipped with multiple counters capable of performing an arbitrary number of turns.

In the present paper, we answer this question in the affirmative. We assess the empirical performance of three types of recurrent networks—Elman-RNNs (or RNNs, in short), LSTMs, and Gated Recurrent Units (GRUs)—to perform *dynamic counting* by training them to learn the Dyck-1 language. Our results demonstrate that the LSTMs with only a single hidden unit perform with perfect accuracy on the Dyck-1 learning task, and successfully generalize far beyond the training set. Furthermore, we show that the LSTMs can learn the shuffles of multiple Dyck-1 languages, defined over disjoint parenthesis-pairs, which require the emulation of multiple-counter arbitrary-turn machines. Our results corroborate the theoretical findings of Weiss et al. (2018), while extending their empirical observations. On the other hand, when trained to learn the Dyck-2 language, which is a strictly context-free language, all our recurrent models failed to learn the language.

2 Preliminaries

We start by defining several subclasses of deterministic pushdown automata (DPA). Following Valiant and Paterson (1975), we define a deterministic one-counter automaton (DCA_1) to be a DPA with a stack alphabet consisting of only one symbol. Traditionally, this construction allows ϵ-moves (that is, executing actions on the stack without the observance of any inputs), but we restrict our attention to simple DCA_1s without ϵ-moves in the rest of this paper. Similarly, we call a DPA that contains k separate stacks, with each stack using only one stack symbol, a deterministic k-counter automaton (DCA_k).[1]

One can impose a further restriction on the direction of stack movement of a DPA. This notion leads to the definition of a deterministic n-turn pushdown automaton (or n-turn DPA, in short) and is well-studied by Ginsburg and Spanier (1966) and Valiant (1974): A DPA is said to be an n-turn DPA if the total number of direction changes in the stack movement of the DPA is at most n for each stack. Note that a one-turn DCA_1 can recognize $a^n b^n$ (Valiant, 1973), whereas a one-turn DCA_2 can recognize $a^n b^n c^n$. We say that a DCA_k with no limit on the number of turns can perform *dynamic counting*.

3 Related Work

Formal languages have long been used to demonstrate the computational power of neural networks. Early studies (Steijvers, 1996; Tonkes and Wiles, 1997; Rodriguez and Wiles, 1998; Bodén et al., 1999; Bodén and Wiles, 2000; Rodriguez, 2001) employed Elman-style RNNs (Elman, 1990) to recognize simple context-free and context-sensitive languages, such as $a^n b^n$, $a^n b^n c^n$, and $a^n b^n c b^m a^m$. Most of these architectures, however, suffered from the vanishing gradient problem (Hochreiter, 1998) and could not generalize far beyond their training sets.

Using LSTMs (Hochreiter and Schmidhuber, 1997), Gers and Schmidhuber (2001) showed that their models could learn two strictly context-free languages and one strictly context-sensitive language by effectively using their gating mechanisms. In contrast, Das et al. (1992) proposed an RNN model with an external stack memory, named Recurrent Neural Network Pushdown Automaton (NNPDA), to learn basic context-free grammars.

More recently, Joulin and Mikolov (2015) introduced simple RNN models equipped with differentiable stack modules, called Stack-RNN, to infer algorithmic patterns, and showed that their model could successfully learn various formal languages, in particular $a^n b^n$, $a^n b^n c^n$, $a^n b^n c^n d^n$, $a^n b^{2n}$. Inspired by the early model design of NNPDAs, Grefenstette et al. (2015) also proposed memory-augmented recurrent networks (Neural Stacks, Queues, and DeQues), which are RNNs equipped with unbounded differentiable memory modules, to perform sequence-to-sequence transduction tasks that require specific data structures.

Deleu and Dureau (2016) investigated the ability of Neural Turing Machines (NTMs; Graves et al. (2014)) to capture long-distance dependencies in the Dyck-1 language. Their empirical findings demonstrated that an NTM can recognize this language by emulating a DPA. Similarly, Sennhauser and Berwick (2018), Bernardy (2018), and Hao et al. (2018) conducted experiments on the Dyck languages to explore whether recurrent networks can learn nested structures. These studies assessed the performance of their recurrent models to predict the next possible parenthesis, assuming that it is a closing parenthesis.[2] In fact, Bernardy (2018) used a purpose-designed architecture, called RUSS, which contains recurrent units with stack-like states, to perform the closing-parenthesis-completion task. Though the RUSS model had no trouble generalizing to longer and deeper sequences, as the author mentions, the specificity of the architecture disqualifies it as a practical model choice for natural language modeling tasks. Additionally, Skachkova et al. (2018) trained recurrent networks to predict the last appropriate closing parenthesis, given a Dyck-2 sequence without its last symbol. They showed that their GRU and LSTM models performed with almost full accuracy on this parenthesis-completion task, but their task does not illustrate that these RNN models can recognize the Dyck language.

Most recently, Weiss et al. (2018) and Suzgun et al. (2019) showed that the LSTM networks can develop natural counting mechanisms to rec-

[1] Weiss et al. (2018) call such a construction a k-counter machine. Previous papers provide a detailed investigation of the complexity of counting machines (Minsky, 1967; Fischer et al., 1968; Valiant and Paterson, 1975; Valiant, 1975).

[2] Their approach is slightly different than ours in the sense that we always try to predict the set of all the possible opening and closing parentheses at each time step.

ognize simple context-free and context-sensitive languages, particularly $a^n b^n$, $a^n b^n c^n$, $a^n b^n c^n d^n$. Their examination of the cell states of the LSTMs revealed that the models learned to emulate simple one- and two-turn counters to recognize these formal languages, but the authors did not conduct any experiments on tasks that require counters to perform arbitrary number of turns.

4 Models

All the models in this paper are recurrent neural architectures and known to capture long-distance relationships in sequential data. We would like to compare and contrast their ability to perform dynamic counting to recognize simple counting languages. We further investigate whether they can learn the Dyck-2 language by emulating a DPA.

A simple RNN architecture (Elman, 1990) is a recurrent model that takes an input x_t and a previous hidden state representation h_{t-1} to produce the next hidden state representation h_t, that is:

$$h_t = f(\mathbf{W}_{ih} x_t + \mathbf{b}_{ih} + \mathbf{W}_{hh} h_{t-1} + \mathbf{b}_{hh})$$
$$y_t = \sigma(\mathbf{W}_y h_t)$$

where $x_t \in \mathbb{R}^D$ is the input, $h_t \in \mathbb{R}^H$ the hidden state, $y_t \in \mathbb{R}^D$ the output at time t, $\mathbf{W}_y \in \mathbb{R}^{D \times H}$ the linear output layer, f an activation function[3] and σ an elementwise logistic sigmoid function.

In theory, it is known that RNNs with infinite precision and rational state weights are computationally universal models (Siegelmann and Sontag, 1995). However, in practice, the exact computational power of RNNs with finite precision is still unknown. Empirically, RNNs suffer from the vanishing or exploding gradient problem, as the length of the input sequences grow (Hochreiter, 1998). To address this issue, different neural architectures have been proposed over the years. Here, we focus on two popular RNN variants with similar gating mechanism, namely LSTMs and GRUs.

The LSTM model was introduced by Hochreiter and Schmidhuber (1997) to capture long-distance dependencies more accurately than simple RNNs. It contains additional gating components to facilitate the flow of gradients during back-propagation.

The GRU model was proposed by Cho et al. (2014) as an alternative to LSTM. GRUs are similar to LSTMs in their design, but do not contain an additional memory unit.

[3] In our experiments, we used the tanh function.

5 Experimental Setup

To evaluate the capability of RNN-based architectures to perform dynamic counting and to encode hierarchical representations, we conducted experiments on four different synthetic sequence prediction tasks. Each task was designed to highlight some particular feature of recurrent networks. All the tasks were formulated as supervised learning problems with discrete k-hot targets and mean-squared-error loss under the sequence prediction framework, defined next. We repeated each experiment ten times but used the same random seed across each run for each of the tasks to ensure comparability of RNN, GRU, and LSTM models.

5.1 The Sequence Prediction Task

Following Gers and Schmidhuber (2001), we trained the models as follows: Given a sequence in the language, we presented one character at each time step to the network and trained the network to predict the set of next possible characters in the language, based on the current input character and the prior hidden state. We used a one-hot representation to encode the inputs and a k-hot representation to encode the outputs. In all the experiments, the objective was to minimize the mean-squared error of the sequence predictions. We used an output threshold criterion of 0.5 for the sigmoid layer to indicate which characters were predicted by the model. Finally, we turned this sequence prediction task into a sequence classification task by *accepting* a sequence if the model predicted *all* of its output values correctly and *rejecting* it otherwise.

5.2 Training Details

Given the nature of the four languages that we will describe shortly, if recurrent models can learn them, then they should be able to do so with reasonably few hidden units and without the employment of any embedding layer or the dropout operation.[4] To that end, all the recurrent models used in our experiments were single-layer networks containing less than 10 hidden units. The number of hidden units that the networks contained for the Dyck-1, Dyck-2, Shuffle-2, and Shuffle-6 experiments were 3, 4, 4, and 8, respectively. (In Section 7, we describe further experiments with as few as a single hidden unit.) In all our experiments, we used the Adam optimizer (Kingma and Ba, 2014) with hyperparameter $\alpha = 0.001$.

[4] Some previous studies used embeddings (Sennhauser and Berwick, 2018; Bernardy, 2018) and the dropout oper-

Sample	(( )) ( ) ( )	([( )]) [([ ])] [ ]
Input	(( )) ( ) ( )	([( )]) [([ ])] [ ]
Output	1 1 1 0 1 0 1 0	1 2 1 2 1 0 2 1 2 1 2 0 2 0

Table 1: Example input-output pairs for the Dyck-1 (left) and Dyck-2 (right) languages.

Sample	([( ))] [([ ]])	[{ ( ) }] ⟨ ⌈ ⌉ ⟩
Input	([( ))] [([ ]])	[{ ( ) }] ⟨ ⌈ ⌉ ⟩
Output	1 3 3 3 2 0 2 3 3 3 1 0	2 6 7 6 2 0 8 24 8 0

Table 2: Example input-output pairs for the Shuffle-2 (left) and Shuffle-6 (right) languages.

6 Experiments

The first language, Dyck-1 (or D_1), consists of well-balanced sequences of opening and closing parentheses. Recall that a neural network need not be equipped with a stack-like mechanism to recognize the Dyck-1 language under the sequence prediction paradigm; a single counter DCA_1 is sufficient. However, dynamic counting is required to capture the language.

The next two languages are the shuffles of two and six Dyck-1 languages, each defined over disjoint parentheses; we refer to these two languages as *Shuffle*-2 and *Shuffle*-6, respectively. These two tasks are formulated to investigate whether recurrent networks can emulate deterministic k-counter automata by performing dynamic counting, separately counting the number of opening and closing parentheses for each of the distinct parenthesis-pairs and predicting the closing parentheses for the pairs for which the counters are non-zero, in addition to the opening parentheses. In contrast, the final language, Dyck-2, is a context-free language which cannot be captured by a simple counting mechanism; a model capable of recognizing the Dyck-2 language must contain a stack-like component (Sennhauser and Berwick, 2018).

Tables 1 and 2 provide example input-output pairs for the four languages under the sequence-prediction task. For purposes of presentation only, we use a simple binary encoding of the output sets to concisely represent the output. In all of the languages we investigate in this paper, the open parentheses are always allowable as next symbol; we assign the set of open parentheses the number 0. Each closing parenthesis is assigned a different power of 2:) is assigned to 1,] to 2, } to 4, ⟩ to 8, ⌉ to 16, and ⌋ to 32. (These latter closing parentheses are needed for the Shuffle-6 language below.) The set of predicted symbols is then the sum of the associated numbers. For instance, an output 3 represents the prediction of any of the open parentheses as well as) and].

We note that even though an input sequence might appear in two different languages, it might have different target representations. This observation is important especially when making comparisons between the Dyck-2 and the Shuffle-2 languages. For instance, the output sequence for ([ ]) in the Dyck-2 language is 1 2 1 0, whereas the output sequence for ([ ]) in the Shuffle-2 language is 1 3 1 0.

6.1 The Dyck-1 Language

The Dyck-1 language, or the well-balanced parenthesis language, arises naturally in enumerative combinatorics, statistics, and formal language theory. A sequence in the Dyck-1 language needs to contain an equal number of opening and closing parentheses, with the constraint that at each time step the total number of opening parentheses must be greater than or equal to the total number of closing parentheses so far. In other words, for a given sequence $s = s_1 \cdots s_{2n}$ of length $2n$ in the Dyck-1 language over the alphabet $\Sigma = \{(,)\}$, if we have a function f that assigns value $+1$ to '(' and value -1 to ')', then we know that it is always true that $\sum_{i=1}^{j} f(s_i) \geq 0$ with strict equality when $j = 2n$, for all $j \in [1, \ldots, 2n]$. Therefore, a model with a single unbounded counter can recognize this language.

A probabilistic context-free grammar for the Dyck-1 language can be written as follows:

$$S \rightarrow \begin{cases} (S) & \text{with probability } p \\ SS & \text{with probability } q \\ \varepsilon & \text{with probability } 1-(p+q) \end{cases}$$

where $0 < p, q < 1$ and $(p + q) < 1$.

ation (Bernardy, 2018) in their experiments.

Task	Model	Training Set			Short Test Set			Long Test Set		
		Min	Max	Med	Min	Max	Med	Min	Max	Med
Dyck-1	RNN	0.45	76.17	46.96	0.28	73.62	41.89	0.06	24.44	7.19
Dyck-1	GRU	99.37	100	100	99.34	100	100	67.68	95.58	84.38
Dyck-1	**LSTM**	**100**	**100**	**100**	**100**	**100**	**100**	**99.98**	**100**	**100**
Shuffle-2	RNN	33.68	87.77	58.16	29.42	86.48	55.37	0.54	25.58	2.75
Shuffle-2	GRU	99.73	100	99.97	99.62	99.98	99.93	83.70	95.18	93.12
Shuffle-2	**LSTM**	**100**	**100**	**100**	**100**	**100**	**100**	**96.84**	**99.98**	**99.35**
Shuffle-6	RNN	0.26	57.39	44.54	0.16	54.80	41.38	0	0.64	0.15
Shuffle-6	GRU	96.32	99.98	99.78	96.08	99.98	99.81	51.96	97.30	85.14
Shuffle-6	**LSTM**	**99.68**	**100**	**100**	**99.74**	**100**	**100**	**82.92**	**99.72**	**98.14**
Dyck-2	RNN	4.37	31.67	14.74	2.96	27.46	12.21	0	0.46	0.01
Dyck-2	GRU	7.78	53.34	28.71	5.38	49.06	25.08	0	1.56	0.05
Dyck-2	**LSTM**	**19.76**	**52.13**	**35.82**	**16.58**	**48.24**	**31.29**	**0**	**1.46**	**0.20**

Table 3: The performances of the RNN, GRU, and LSTM models on four language modeling tasks. Shuffle-2 denotes the shuffle of two Dyck-1 languages defined over different alphabets, and similarly Shuffle-6 denotes the shuffle of six Dyck-1 languages defined over different alphabets. Min/Max/Median results were obtained from 10 different runs of each model with the same random seed across each run. We note that the LSTM models not only outperformed the RNN and GRU models but also often achieved full accuracy on the short test set in all the "counting" tasks. Nevertheless, even the LSTMs were not able to yield a good performance on the Dyck-2 language modeling task, which requires a stack-like mechanism.

Setting $p = \frac{1}{2}$ and $q = \frac{1}{4}$, we generated $10,000$ distinct Dyck sequences, whose lengths were bounded to $[2, 50]$, for the training set. We used two test sets: The "short" test set contained $5,000$ distinct Dyck words defined in the same length interval as the training set but distinct from it. The "long" test set contained $5,000$ distinct Dyck words defined in the interval $[52, 100]$. Hence, there was no overlap between any of the training and test sets.

Results: Table 3 lists the performances of the RNN, GRU, and LSTM models on the Dyck-1 language. First, we highlight that all the LSTM models obtained full accuracy on the training set and short test set, whose sequences were bounded to $[2, 50]$, in all the ten trials. They were also able to easily generalize to longer and deeper sequences in the long test set: They obtained perfect accuracy in nine out of ten trials and 99.98% accuracy (only 1 incorrect prediction) in the remaining trial. These results exhibit that the LSTMs can indeed perform unbounded dynamic counting.

The GRUs yielded an almost similar qualitative performance on the training and first test sets; however, they could not generalize well to longer and deeper sequences. On the other hand, the RNN models performed significantly worse than the first two recurrent models, in terms of their median accuracy rate. We note that similar empirical observations about the performance-level differences between the RNNs, GRUs, and LSTMs for other simple formal languages were also reported by Weiss et al. (2018) and Bernardy (2018).

6.2 The Shuffle-k Language

Next, we consider two *shuffle* languages, which are both generated by the Dyck-1 language. Before describing each task in detail, let us first define the notion of *shuffling* formally. The shuffling operation $|| : \Sigma^* \times \Sigma^* \rightarrow \mathcal{P}(\Sigma^*)$ can be inductively defined as follows:[5]

- $u||\varepsilon = \varepsilon||u = \{u\}$
- $\alpha u||\beta v = \alpha(u||\beta v) \cup \beta(\alpha u||v)$

for any $\alpha, \beta \in \Sigma$ and $u, v \in \Sigma^*$. For instance, the shuffling of ab and cd would be:

$$ab||cd = \{abcd, acbd, acdb, cabd, cadb, cdab\}$$

There is a natural extension of the shuffling operation $||$ to languages. The *shuffle* of two languages $\mathcal{L}_1$ and $\mathcal{L}_2$, denoted $\mathcal{L}_1||\mathcal{L}_2$, is defined as the set of all the possible interleavings of the elements of $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively, that is:

$$\mathcal{L}_1||\mathcal{L}_2 = \bigcup_{u \in \mathcal{L}_1,\, v \in \mathcal{L}_2} u||v$$

[5] We abuse notation by allowing a string to stand for its own singleton set.

Given a language $\mathcal{L}$, we define its self-shuffling $\mathcal{L}||^2$ to be $\mathcal{L}||\sigma(\mathcal{L})$, where σ is an isomorphism on the vocabulary of $\mathcal{L}$ to a disjoint vocabulary. More generally, we define the k-self-shuffling

$$\mathcal{L}||^k = \begin{cases} \{\varepsilon\} & \text{if } k = 0 \\ \mathcal{L}||\sigma(\mathcal{L}||^{k-1}) & \text{otherwise} \end{cases} .$$

The Shuffle-2 Language: The first language is $D_1||^2$, the shuffle of D_1 and D_1, where the first D_1 is over the alphabet $\{(,)\}$ and the second over the alphabet $\{[,]\}$. For instance, the sequence ([ )] is in $D_1||^2$ but not in D_2, whereas ( ] [ ) is in neither $D_1||^2$ nor D_2. Note that D_2 is a subset of $D_1||^2$, but not the other way around.[6]

To generate the training and test corpora, we used a probabilistic context-free grammar for the Dyck-2 language, which we will describe shortly, but considered correct target values for the sequences interpreted as per the Shuffle-2 language. The training set contained $10,000$ distinct sequences of lengths in $[2, 50]$. As before, the short test set had $5,000$ distinct samples defined over the same length interval but disjoint from the training set, and the long test set had $5,000$ distinct samples, whose lengths were bounded to $[52, 100]$.

The Shuffle-6 Language: The second shuffle language is $D_1||^6$, the shuffle of six Dyck-1 languages, each defined over different parenthesis-pairs. Concretely, we used the following pairs: $(\ ), [\ ], \{ \ \}, \langle \ \rangle, \lceil \ \rceil, \lfloor \ \rfloor$. In theory, an automaton with six separate unbounded-turn counters (DCA_6) can recognize this language. Hence, we wanted to explore whether our recurrent networks can learn to emulate a dynamic-counting 6-counter machine.

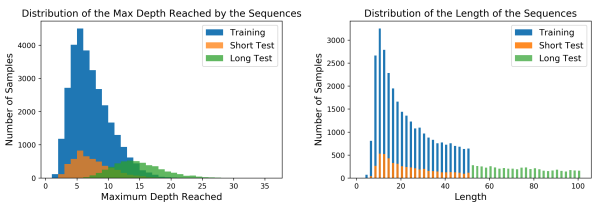

Figure 1: Length and maximum depth distributions of training/test sets for an example Shuffle-6 experiment.

The training and two test corpora were generated in the same style as the previous sequence prediction task; however, we included $30,000$ samples in the training set for this language, due to the complexity of the language. Figure 1 shows the length and maximum depth distributions of the training and test sets for one of the Shuffle-6 experiments.

Results: As shown shown in Table 3, the LSTM models achieved a median accuracy of 100% on the training and short test sets in both of the shuffle language variants. Furthermore, they were able to generalize well to longer and deeper sequences in both shuffle languages, achieving an almost perfect median accuracy score on the long test set. In contrast, the GRU models performed slightly worse than the LSTM models on the training and short test sets, but the GRUs did not yield the same performance as the LSTMs on the long test set, obtaining median scores of 93.12% and 85.14% in the Shuffle-2 and Shuffle-6 languages, respectively. Additionally, the simple RNN models always performed much worse than the GRU and LSTM models and could not even learn the training sets in either of the shuffle languages. These empirical findings show that the LSTM models can successfully emulate a DCA_k, a deterministic (real-time) automaton with k-counters, each capable of performing an arbitrary number of turns.

6.3 The Dyck-2 Language

The generalized Dyck language, D_n, represents the core of the notion of context-freeness by virtue of the Characterization Theorem of Chomsky and Schützenberger (1963), which provides a natural way to characterize the CFL class:

Theorem 6.1. *Any language in CFL can be represented as a homomorphic image of the intersection of a Dyck language D_n and a regular language R.*

Furthermore, D_n can be reduced to D_2 at the expense of increasing the depth and length of the original sequences in the former language.

Proposition 6.2. *D_n is reducible to D_2.*

Proof. [7] Let D_n be the Dyck language with n distinct pairs of parentheses. Let us further suppose that $p = \{p_1, p_2, p_3, \ldots, p_n\}$ are the opening parentheses and that $\bar{p} = \{\overline{p_1}, \overline{p_2}, \overline{p_3}, \ldots, \overline{p_n}\}$ are their corresponding closing parentheses. We set $m = \lceil \log_2 n \rceil$ and encode each opening and closing parenthesis in D_n with m bits using either (and [or) and]. Furthermore, we map empty string to empty string.

[6] On the other hand, we highlight once again that the target sequences in $D_1||^2$ are often different from those in D_2.

[7] A similar reduction is also provided by Magniez et al. (2014).

Given an open parenthesis p_i, we first determine the m-digit binary representation of the number $i-1$ and then use (to encode 0's and [to encode 1's in this representation. Given a closing parenthesis $\overline{p_i}$, we determine the m-digit binary representation of the number $i-1$, write the binary number *in the reverse order*, and then use) to encode 0's and] to encode 1's. That is,

$$\begin{aligned}\Gamma : p \cup \bar{p} \cup \{\varepsilon\} &\to \mathbb{S}^m \cup \{\varepsilon\} \\ p_i &\mapsto \mathbf{Enc}_{([} \circ \text{Bin}(i-1) \\ \overline{p_i} &\mapsto \mathbf{Enc}_{)]} \circ \text{Rev} \circ \text{Bin}(i-1) \\ \varepsilon &\mapsto \varepsilon\end{aligned}$$

where $\mathbb{S} = \{(,[,),]\}$, the parentheses in D_2. We note that $s = s_1 s_2 s_3 \cdots s_k$ is in D_n if and only if $\Gamma(s) = \Gamma(s_1)\Gamma(s_2)\Gamma(s_3)\cdots\Gamma(s_k)$ is in D_2, completing the reduction. □

The previous proposition simply shows that we can map an expression in D_n to an expression in D_2 at the expense of creating a deeper structure in the latter language by a factor of $m = \lceil \log_2 n \rceil$. For instance, if an expression s in D_n has a maximum depth of k, then the expression generated by the mapping above would have a maximum depth of $k \times m$ in D_2.

Motivated by context-free-language universality (Sennhauser and Berwick, 2018), we therefore experimented with the Dyck-2 language defined over two types of parenthesis-pairs, namely $\{(,)\}$ and $\{[,]\}$, as well. The recognition of the Dyck-2 language requires a model to possess a stack-like component; real-time primitive counting does not enable us to capture the Dyck-2 language. Hence, if an RNN-based architecture learns to recognize this language, we can conclude that RNNs with finite precision can actually learn complex deeply nested representations.

A probabilistic context-free grammar for the Dyck-2 language can be written as follows:

$$S \to \begin{cases} (S) & \text{with probability } \frac{p}{2} \\ [S] & \text{with probability } \frac{p}{2} \\ SS & \text{with probability } q \\ \varepsilon & \text{with probability } 1-(p+q) \end{cases}$$

where $0 < p, q < 1$ and $(p+q) < 1$.

Setting $p = \frac{1}{2}$ and $q = \frac{1}{4}$, we generated $10,000$ distinct sequences, whose lengths were bounded to $[2, 50]$, for the training set. Again, we generated $5,000$ other distinct Dyck-2 sequences of lengths defined in the interval $[2, 50]$ for the first test set and $5,000$ distinct sequences of lengths defined in the interval $[52, 100]$ for the second test set. As in the previous case, there was no overlap between the training and test sets.

Results: As shown in Table 3, we found that none of our RNNs was able to emulate a DPA to recognize the Dyck-2 language, a context-free language that requires a model to contain a stack-like mechanism for recognition. Overall, the LSTM models had the best performances among all the networks, but they still failed to employ a stack-based strategy to learn the Dyck-2 language. Even the best LSTM model could achieve only 48.24% and 1.46% accuracy scores on the short and long test sets, respectively.

7 Discussion and Analysis

7.1 Visualization of Hidden+Cell States

Our empirical results on the Dyck-1 and Shuffle languages suggest that our LSTM models were performing dynamic counting to recognize these languages. In order to validate our hypothesis, we visualized the hidden and cell states of some of our LSTM models that achieved full accuracy on the test sets.

Figure 2 illustrates that our LSTM is able to recognize the samples in $D_1||^6$ by emulating a DCA_6. In fact, the discrete even transitions in the cell state dynamics of the model reveal that six out of eight hidden units in the model are acting like separate counters. In some cases, we further discovered that certain units learned to count the length of the input sequences. Such length counting behaviours are also observed in machine translation (Shi et al., 2016; Bau et al., 2019; Dalvi et al., 2019) when the LSTMs are trained on a fixed-length training corpus.[8]

On the other hand, Figure 3 provides visualizations of the hidden and cell state dynamics of one of our single-layer LSTM models with four hidden units when the model was presented two sequences in the Dyck-2 language. Both sequences have some noticeable patterns and were chosen to explore whether the model behaves differently in repeated (or similar) subsequences. It seems that the LSTM model is trying to employ a complex counting strategy to learn the Dyck-2 language but failing to accomplish this task.

[8]The visualizations for the Dyck-1 and Shuffle-2 languages were qualitatively similar.

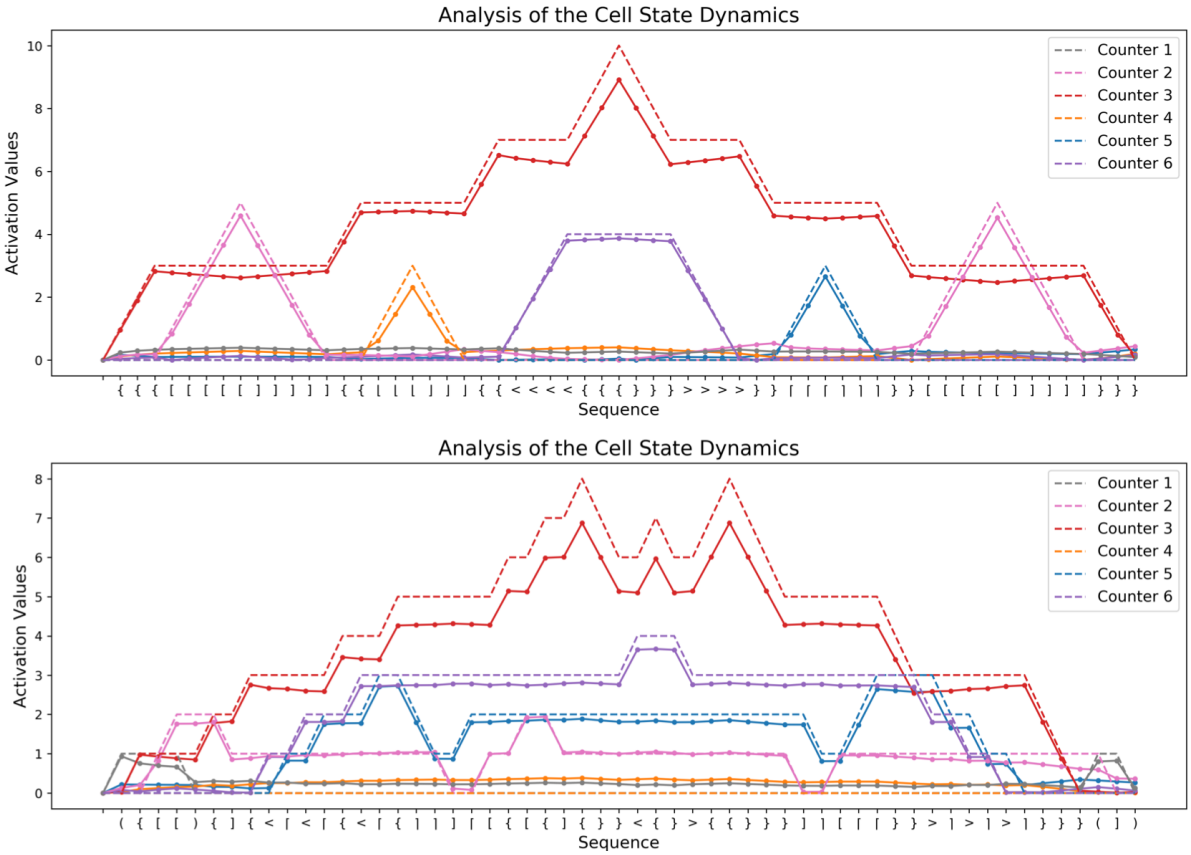

Figure 2: Visualization of the cell state dynamics of one of the LSTM models trained to learn $D_1||^6$, the Shuffle-6 language. The solid lines show the values of the cell states of the six out of eight units in the model, whereas the dashed lines depict the current depth of each distinct parenthesis-pair in $D_1||^6$. We highlight the striking parallelism between the solid lines and the dashed-lines. Our visualizations confirm that the LSTM models employ a simple counting mechanism to recognize the Shuffle languages.

7.2 LSTM with a Single Hidden Unit

In theory, a DCA_1 should be able to easily recognize Dyck-1, the well-balanced parenthesis language. Can an LSTM with one hidden unit learn Dyck-1? Our empirical results (Figure 4) confirmed that LSTMs can indeed learn this language by effectively using the single hidden unit to count up the total number of left and right parentheses in the sequence. Similarly, we found that an LSTM with only two hidden units can recognize $D_1||^2$.

7.3 Predicting the Last Closing Parenthesis

Following Skachkova et al. (2018), we also trained an LSTM model with four hidden units to learn to predict the last closing parenthesis in the Dyck-2 language. The model learned the task in a couple of epochs and achieved perfect accuracy on the training and test sets. However, our simple analysis of the cell state dynamics of the LSTM in Figure 5 suggests that the model is doing some complex form of counting to perform the desired task, rather than learning the Dyck-2 language.

8 Conclusion

We investigated the ability of standard recurrent networks to perform dynamic counting and to encode hierarchical representations, by considering three simple counting languages and the Dyck-2 language. Our empirical results highlight the overall high-caliber performance of the LSTM models over the simple RNNs and GRUs, and further inflect our understanding of the limitations and strengths of these models.

9 Acknowledgement

The first author gratefully acknowledges the support of the Harvard College Research Program (HCRP). The third author was supported by the Harvard Mind, Brain, and Behavior Initiative. The computations in this paper were run on the Odyssey cluster supported by the FAS Division of Science, Research Computing Group at Harvard University.

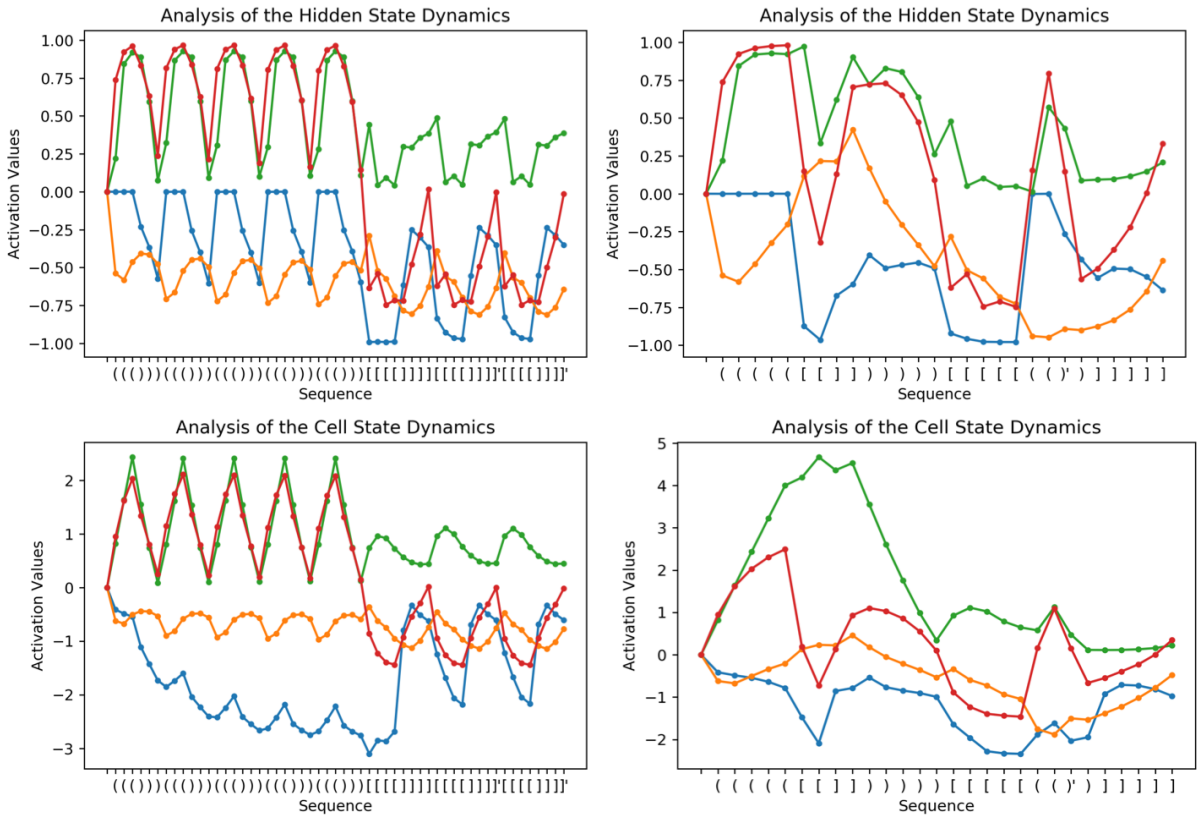

Figure 3: Visualization of the hidden and cell state dynamics of one of the LSTMs trained to learn the Dyck-2 language. The time steps at which the model made incorrect predictions are marked with an apostrophe in the horizontal axis. The plots on the left provide a demonstration of the periodic behaviour of the hidden and cell states of the model for a long sequence. Similarly, the plots on the right provide the complex counting behaviour of the model as it observes a nested sequence. We witnessed similar behaviours in our other models as well.

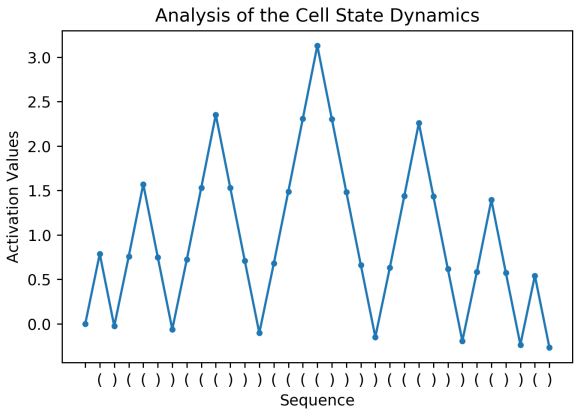

Figure 4: A single-layer LSTM with one hidden unit learns the Dyck-1 language by counting up upon the observance of (and down upon the observance of).

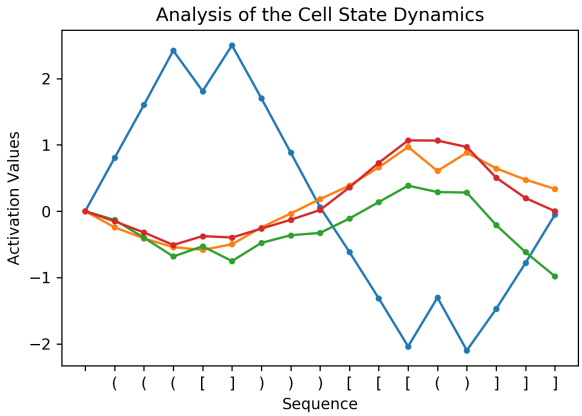

Figure 5: The cell state dynamics of one of the LSTM models trained to predict the last closing parenthesis. Our LSTMs often achieved full accuracy on this task.

References

Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. 2019. Identifying and controlling important neurons in neural machine translation. In *International Conference on Learning Representations*.

Jean-Philippe Bernardy. 2018. Can recurrent neural networks learn nested recursion? *LiLT (Linguistic Issues in Language Technology)*, 16(1).

Mikael Bodén and Janet Wiles. 2000. Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science*, 12(3-4):197–210.

Mikael Bodén, Janet Wiles, Bradley Tonkes, and Alan Blair. 1999. Learning to predict a context-free lan-

guage: Analysis of dynamics in recurrent hidden units.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Noam Chomsky and Marcel P Schützenberger. 1963. The algebraic theory of context-free languages. In *Studies in Logic and the Foundations of Mathematics*, volume 35, pages 118–161. Elsevier.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, D. Anthony Bau, and James Glass. 2019. What is one grain of sand in the desert? analyzing individual neurons in deep NLP models. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*.

Sreerupa Das, C Lee Giles, and Guo-Zheng Sun. 1992. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, page 14.

Tristan Deleu and Joseph Dureau. 2016. Learning operations on a stack with Neural Turing Machines. *arXiv preprint arXiv:1612.00827*.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Patrick C Fischer, Albert R Meyer, and Arnold L Rosenberg. 1968. Counter machines and counter languages. *Mathematical systems theory*, 2(3):265–283.

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

Seymour Ginsburg and Edwin H Spanier. 1966. Finite-turn pushdown automata. *SIAM Journal on Control*, 4(3):429–453.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural Turing Machines. *arXiv preprint arXiv:1410.5401*.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pages 1828–1836.

Yiding Hao, William Merrill, Dana Angluin, Robert Frank, Noah Amsel, Andrew Benz, and Simon Mendelsohn. 2018. Context-free transductions with neural stacks. *arXiv preprint arXiv:1809.02836*.

Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780.

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

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. 2014. Recognizing well-parenthesized expressions in the streaming model. *SIAM Journal on Computing*, 43(6):1880–1905.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

Marvin Lee Minsky. 1967. *Computation: Finite and Infinite Machines*. Prentice-Hall Englewood Cliffs.

Paul Rodriguez. 2001. Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural computation*, 13(9):2093–2118.

Paul Rodriguez and Janet Wiles. 1998. Recurrent neural networks can learn to implement symbol-sensitive counting. In *Advances in Neural Information Processing Systems*, pages 87–93.

Luzi Sennhauser and Robert Berwick. 2018. Evaluating the ability of LSTMs to learn context-free grammars. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 115–124.

Xing Shi, Kevin Knight, and Deniz Yuret. 2016. Why neural translations are the right length. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2278–2282.

Hava T Siegelmann and Eduardo D Sontag. 1995. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150.

Natalia Skachkova, Thomas Trost, and Dietrich Klakow. 2018. Closing brackets with recurrent neural networks. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 232–239.

Mark Steijvers. 1996. A recurrent network that performs a context-sensitive prediction task.

Mirac Suzgun, Yonatan Belinkov, and Stuart M Shieber. 2019. On evaluating the generalization of LSTM models in formal languages. *Proceedings of the Society for Computation in Linguistics (SCiL)*, pages 277–286.

Bradley Tonkes and Janet Wiles. 1997. Learning a context-free task with a recurrent neural network: An analysis of stability. In *In Proceedings of the Fourth Biennial Conference of the Australasian Cognitive Science Society*. Citeseer.

Leslie Valiant. 1973. *Decision Procedures for Families of Deterministic Pushdown Automata*. Ph.D. thesis, University of Warwick.

Leslie G Valiant. 1974. The equivalence problem for deterministic finite-turn pushdown automata. *Information and Control*, 25(2):123–133.

Leslie G Valiant. 1975. Regularity and related problems for deterministic pushdown automata. *Journal of the ACM (JACM)*, 22(1):1–10.

Leslie G Valiant and Michael S Paterson. 1975. Deterministic one-counter automata. *Journal of Computer and System Sciences*, 10(3):340–350.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 740–745.