Investigating Backpropagation Alternatives when Learning to Dynamically Count with Recurrent Neural Networks

Ankur Mali AAM35@PSU.EDU
The Pennsylvania State University
University Park, PA 16802 USA

Alexander Ororbia AGO@CS.RIT.EDU
Rochester Institute of Technology
Rochester, NY 14623 USA

Daniel Kifer DUK17@PSU.EDU
The Pennsylvania State University
University Park, PA 16802 USA

Lee Giles CLG20@PSU.EDU
The Pennsylvania State University
University Park, PA 16802 USA

Editors: Jane Chandlee, Rémi Eyraud, Jeffrey Heinz, Adam Jardine, and Menno van Zaanen

Abstract

Artificial neural networks, such as recurrent neural networks (RNNs) and transformers, have empirically demonstrated impressive performance across many natural language processing tasks. However, automated text processing at a deeper and more interpretable level arguably requires extracting intricate patterns such as underlying grammatical structures. As a result, correctly interpreting a neural language model would require an understanding of linguistic structure through formal language theory. Nonetheless, there is often a discrepancy between theoretical and practical findings that restrict models informed by formal language theory in real-life scenarios. For instance, while learning context-free grammars (CFGs), existing neural models fall short because they lack appropriate memory structures. In this work, we investigate how learning algorithms affect the generalization ability of RNNs that are designed to learn context-free languages (CFGs) as well as their ability to encode hierarchical representations. To do so, we investigate a range of learning algorithms on complex, context-free languages such as the Dyck languages, with a focus on the RNN's ability to generalize to longer sequences when processing a CFG. Our results demonstrate that a Long Short-term memory (LSTM) RNN equipped with second-order connections, trained with the sparse attentive backtracking (SAB) algorithm, performs stably across various Dyck languages and successfully emulates real-time-counter machines. We empirically show that RNNs without external memory are incapable of recognizing Dyck-2 languages, which require a stack-like structure. We finally investigate each learning algorithm's performance on real-world language modeling tasks using the Penn Tree Bank and text8 benchmarks. We further investigate how an increase in model parameters affects each RNN's stability and grammar recognition performance when trained using different learning algorithms.

Keywords: Automata theory, Dyck languages, context free languages, recurrent neural network, formal language, BPTT alternative, forward mode learning algorithms, self-attention

© 2021 A. Mali, A. Ororbia, D. Kifer & L. Giles.

1. Introduction

Artificial neural networks (ANNs), including recurrent neural networks (RNNs) and transformers, can be used to capture the long distance, complex dependencies inherent to sequential data. Recently, transformer-based architectures have become popular for natural language processing tasks (Vaswani et al., 2017), often outperforming RNNs. When equipped with infinite precision and rational state weights, transformer-based architectures have theoretical limits that make them incapable of recognizing context free grammars (CFGs) (Hahn, 2020). On the other hand, RNN-based models are known to be, theoretically, Turing-complete with infinite precision and weights (Siegelmann and Sontag, 1994; Korsky and Berwick, 2019). Recent findings show that, even with finite precision and weights, tensor RNNs are still capable of recognizing any CFG (Stogin et al., 2020)

Recent work in formally analyzing the computational power of RNNs (Weiss et al., 2018a; Merrill et al., 2020) has demonstrated that a popular variant of a first order RNN known as the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) can theoretically emulate a simple real-time k-counter machine (Fischer et al., 1968). Specifically, experiments using the LSTM on the $a^n b^n$ and $a^n b^n c^n$ grammars were conducted and the empirical findings correlated with proposed theory when the network's cell states were examined. However, these two formal grammars are simple forms of an automaton that can be easily captured by a deterministic, one-turn two-counter automaton (Ginsburg and Spanier, 1966). As such, it is still an open question as to how computationally capable RNNs are in understanding and recognizing more complex grammars.

Recently, (Suzgun et al., 2019a; Mali et al., 2020) examined the practical computational expressivity of RNNs and showed that an LSTM performed well compared to other RNN alternatives. Nevertheless, it is still debatable whether the algorithm that trains these ANNs, i.e., backpropagation of errors (BP), is capable of facilitating optimal behaviour given that it often struggles with the well-known problem of credit assignment (Mackintosh, 1975). As a result, it is often challenging to uncover which parts of a network affects model representation and overall performance. If a model's representation is compromised, its parameter optimization will follow a sub-optimal search trajectory towards a solution, thus hampering generalization on unseen data. We argue that to truly understand the practical computational expressivity and robustness of RNNs with finite precision, it is crucial to examine different learning algorithms (that manipulate model parameters) in order to find a viable pathway to building neural models with robust internal representations.

In seeking models with more robust representations, we investigate models that are structured more appropriately for formal languages. With respect to neural models, as shown in prior work (Giles et al., 1990; Omlin and Giles, 1996a; Mali et al., 2020, 2021c), higher-order synaptic connections provide better expressivity by naturally encoding grammatical structure and yielding stable representations. Unfortunately, these connections are computationally expensive to employ. Motivated by this finding and the desire to better ensure computational viability, we extend popular RNN variants with a rank-1 approximation of second-order weights. In addition, we examine how utilizing different training algorithms interact with these types of weights and affect an RNN's acquisition of stable, robust representations useful for formal learning recognition. As argued in prior work (Omlin and Giles, 1996b), if an internal representation is robust (consisting of ideal weights to recognize any given language) then one can stably extract minimal state machines from these trained RNNs, resulting in stable generalization across strings of varied length. As also evident from prior work Zeng et al. (1994), models equipped with an optimal set of weights generalize quite well. We believe that our study and findings will aid in determining whether RNNs empirically are capable of learning more general finite-state automata and shed light as to what their computational limitations might be. To our knowledge, this is the first work to analyze how the performance of RNNs are affected by different learning algorithms on the task of grammatical inference.

This work aims to investigate an alternative to BP for neural models with and without second-order connections and to answer the following questions: 1) can one of these algorithms can stably recognize a formal language?, 2) what advantages do these alternatives offer when learning a formal language?, and 3) which algorithm leads to better generalization and stable, internal representations when tested on longer sequences? In answering the questions above, this work serves as a stepping stone to a more complete understanding of the computational power underlying RNNs when learning formal languages and motivate the development of better processes and mechanisms to acquire latent representations that would prove useful for stably extracting automaton (a highly desirable and important application of language-recognition RNNs).

In the same spirit as prior effort (Suzgun et al., 2019a; Mali et al., 2020), we assess the empirical performance of three types of RNNs with and without second order connections—Elman-RNNs (or RNNs), LSTMs, and gated recurrent units (GRUs). We analyze a wide variety of learning algorithms such as BP through time (BPTT), sparse attentive backtracking (SAB) (Ke et al., 2017), difference target propagation (DTP) (Manchev and Spratling, 2020), unbiased online recurrent optimization (UORO) (Tallec and Ollivier, 2017b), and the Kronecker factors recurrent real-time learning algorithm (KL-RTRL) (Mujika et al., 2018) to perform *dynamic counting.* We train these models to learn bounded, context free languages such as the Dyck-1 language. To further understand the performance of each learning algorithm, we also evaluate them on a real world language modeling task using Penn Treebank and text-8. Our results align with prior findings (Suzgun et al., 2019a; Weiss et al., 2018a), demonstrating that LSTMs and multiplicative LSTMs (MI-LSTMs), with even a single neuron, can reach perfect accuracy and generalize better. We also find that LSTMs and their second order variants trained with DTP and UORO are consistent and robust across various seeds. However, we also find that the higher-order variant of the GRU trained with KF-RTRL and DTP can also perform dynamic counting on shuffles of multiple Dyck-1 languages, defined over disjoint parenthesis-pairs. On the other hand, when we train RNNs on unbounded or strictly CFGs, such as the Dyck-2 language, all models struggle to learn. Finally, we show that, as model complexity increases, BP starts to dominate in final performance. To this end, our main contributions are:

- To our knowledge, we are the first to examine the effect that learning algorithm alternatives to BP have on neural models trained for the task of grammatical inference.
- We extend all popular RNN variants to make them compatible with BP algorithm alternatives.
- We conduct the first analysis of these algorithms on large-scale language modeling problems.

2. Related Work

Historically, grammatical inference (Gold, 1967) has been at the core of formal language theory and could be considered to be a fundamental direction in understanding the structure of natural languages. A summary of much of the theoretical work done in formal languages can be found in (De la Higuera, 2010). Applications of formal language work have led to the development of methods for predicting and understanding sequences in diverse critical areas, including financial time series, genetics and bioinformatics (Searls, 1993; Sakakibara et al., 1994), and software data exchange (Giles et al., 2001; Wieczorek and Unold, 2016; Exler et al., 2018). Notably, RNNs have been trained to learn context free and context-sensitive counter languages (Gers and Schmidhuber, 2001; Bodén and Wiles, 2000; Tabor, 2000; Wiles and Elman, 1995; Sennhauser and Berwick, 2018; Nam et al., 2019; Wang and Niepert, 2019; Cleeremans et al., 1989; Kolen, 1994; Cleeremans et al., 1989; Weiss et al., 2018b; Mali et al., 2020). Despite positive results, these networks often fail to generalize when tested on real world scenarios and the computational capabilities of these models with limited resources is still unknown. Recently, (Gers and Schmidhuber, 2001; Mali et al., 2020; Suzgun et al., 2019b) tried to probe the computational capabilities of RNNs. From a theoretical perspective, RNNs augmented with external memory have historically been considered to be more capable of efficiently recognizing context free grammars (CFGs), i.e., tensor RNNs (Das et al., 1993; Pollack, 1990; Sun et al., 1997; Mali et al., 2021c) and, more recently, first order RNNs with various differentiable memory structures (Joulin and Mikolov, 2015; Grefenstette et al., 2015; Graves et al., 2014; Kurach et al., 2015; Zeng et al., 1994; Hao et al., 2018; Yogatama et al., 2018; Graves et al., 2016; Le et al., 2019; Arabshahi et al., 2019; Mali et al., 2021b). Despite the positive recognition results, prior work has not focused on examining generalization over very long strings and has also highlighted the difficulty in training memory augmented models. In contrast, (Zeng et al., 1994; Mali et al., 2021c; Suzgun et al., 2019b) and (Gers and Schmidhuber, 2001) presented promising results in generalizing beyond the training dataset with respect to simple CFGs.

Recent work on differentiable stacks (the stackRNN) (Joulin and Mikolov, 2015) and (stack-RNN) (Suzgun et al., 2019b), which are closely related to this work, have focused on language modeling and on learning simple algorithmic patterns. Note that these models were able to solve problems which required counting as well as memorization. Other work related to differentiable memory (Grefenstette et al., 2015) was motivated by the neural pushdown automaton, i.e., the NNPDA (Das et al., 1992, 1993; Mozer and Das, 1993; Sun et al., 1997), which extended RNNs to use an unbounded, external differentiable memory such as the stack, queue, and doubly-linked list. Few prior studies have focused on evaluating models on complex and long CFGs. While other memory-augmented models have been proposed to efficiently recognize CFGS, such as neural random access memory (NRAM) (Kurach et al., 2015) and the neural Turing machine (NTM) (Graves et al., 2014), these models suffer from instability issues and are quite difficult to train and scale to real world problems. Other approaches to tackling this problem include inductive logic programming Lin et al. (2014) or inductive functional logic programming Hofmann et al. (2009), which empirically provide better prior or structures to a neural model. We believe our findings complement these models and methods since representation plays a vital role in learning hierarchical languages. To truly evaluate internal representation and the generalization ability of RNNs and other memory-augmented models on CFGS, it is important to consider testing them on more complex, longer CFGs such as the Dyck languages (Suzgun et al., 2019a).

3. Recurrent Learning Algorithms

We train RNNs with various learning algorithms to analyze their effect on model performance on grammatical inference, with the hope of finding more robust models that efficiently solve the credit assignment problem. Below, we briefly describe the key learning algorithms tested in this work.

3.1. Kronecker Factors Real-Time Recurrent Learning

Before diving into the Kronecker factors version, we will discuss how real-time recurrent learning (RTRL) (Williams and Zipser, 1989) works. RTRL is a learning procedure for training RNNs online. Unlike BPTT, RTRL does forward-mode differentiation to calculate parameter gradients and thus does not require unrolling over time. The aim of RTRL is to optimize parameters, denoted as Θ, in order to minimize a total loss for a model with a state function defined as follows:

$$\mathbf{z}_{t+1} = F_{state}(\mathbf{x}_{t+1}, \mathbf{z}_t, \Theta). \tag{1}$$

Again, one key advantage of RTRL is that it computes the derivative of the states and the outputs in its forward computation. For next step prediction, the loss L to optimize, using RTRL, is simply:

$$\frac{\partial L_{t+1}}{\partial \Theta} = \frac{\partial L_{t+1}(\mathbf{y}_{t+1}, \mathbf{y}^*_{t+1})}{\partial \mathbf{y}} \otimes \left(\frac{\partial F_{\text{out}}(\mathbf{x}_{t+1}, \mathbf{z}_t, \Theta)}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \Theta} + \frac{\partial F_{\text{out}}(\mathbf{x}_{t+1}, \mathbf{z}_t, \Theta)}{\partial \Theta} \right). \tag{2}$$

If we differentiate Equation 1 with respect to Θ, we obtain:

$$\frac{\partial \mathbf{z}_t + 1}{\partial \Theta} = \frac{\partial F_{\text{state}}(\mathbf{x}_{t+1}, \mathbf{z}_t, \Theta)}{\partial \Theta} + \frac{\partial F_{\text{state}}(\mathbf{x}_{t+1}, \mathbf{z}_t, \Theta)}{\partial \mathbf{z}_t} \otimes \frac{\partial \mathbf{z}_t}{\partial \Theta} \tag{3}$$

where at each time step, we compute $\frac{\partial \mathbf{z}_t}{\partial \Theta}$ based on $\frac{\partial \mathbf{z}_{t-1}}{\partial \Theta}$ and then use these values to directly compute $\frac{\partial \mathbf{z}_{t+1}}{\partial \Theta}$. This is how RTRL calculates its gradients without unfolding. The shape/size of $\frac{\partial \mathbf{z}_t}{\partial \Theta}$ is equal to $|z| \times |\Theta|$, therefore, for standard RNNs with n hidden units, this calculation scales with n^4 time complexity (Williams and Zipser, 1995). This makes RTRL impractical for most problems despite its unique advantage of avoiding the need for unrolling the underlying computational graph.

KF-RTRL solves the impracticality of RTRL by approximating $\partial \mathbf{z}_t$, or $\frac{\partial \mathbf{z}_t}{\partial \Theta}$ (the derivative of the state with respect to parameters). It calculates an unbiased estimate $\hat{\partial \mathbf{z}_t}$ of $\partial \mathbf{z}_t$. At time-step t, KF-RTRL maintains vector u_t and matrix A_t, such that $\hat{z}_t = u_t \otimes A_t$ where $\otimes$ is the Kronecker product, satisfying a key condition for ensuring unbiased (gradient) estimates (Mujika et al., 2018).

3.2. Unbiased Online Recurrent Optimization

Unbiased Online Recurrent Optimization (UORO) (Tallec and Ollivier, 2017a), which is considered to be a noisy approximation of RTRL, uses a rank-1 trick to approximate the operations involved in RTRL. This helps in reducing the computational cost during training and, for online learning, UORO is faster than BPTT. For instance, for any given unbiased estimation of $\frac{\partial \mathbf{z}_t}{\partial \Theta}$, we can form a stochastic matrix $\tilde{Z}_t$ such that $\mathbf{E}(\tilde{Z}_t) = \frac{\partial \mathbf{z}_t}{\partial \Theta}$. Since Equations 2 and 3 are affine in $\frac{\partial \mathbf{z}_t}{\partial \theta}$, unbiasedness is preserved due to the linearity of the expectation. We compute the value of $\tilde{Z}_t$ and plug it into Equations 2 and 3 to calculate the value for $\frac{\partial \mathbf{L}_{t+1}}{\partial \Theta}$ and $\frac{\partial \mathbf{z}_{t+1}}{\partial \Theta}$. For a rank-one, unbiased approximation, $\tilde{Z}_t = \tilde{z}_t \otimes \tilde{\Theta}_t$ at step t .To calculate $\hat{Z}_{t+1}$ at $t+1$, we plug in $\tilde{Z}_t$ into Equation 3.

In order to obtain a proper rank-1 approximation, we use the efficient approximation technique proposed in (Ollivier et al., 2015) where we rewrite the above equation as follows:

$$\tilde{Z}_{t+1} = \left(\rho_0 \frac{\partial F_{\text{state}}(\mathbf{x}_{t+1}, \mathbf{z}_t, \theta)}{\partial \mathbf{z}} \tilde{\mathbf{z}}_t + \rho_1 \nu \right) \otimes \left(\frac{\tilde{\theta}_t}{\rho_0} + \frac{(\nu)^T}{\rho_1} \frac{\partial F_{\text{state}}(\mathbf{x}_{t+1}, \mathbf{z}_t, \theta)}{\partial \theta} \right) \tag{4}$$

where ν is a vector of independent, random signs. ρ contains k positive numbers and the rank-1 trick can be applied for any ρ. In UORO, ρ_0 and ρ_1 are meant to control the variance of the derivative approximations. In practice, we define ρ_0 and ρ_1 as:

$$\rho_0 = \sqrt{\frac{\|\tilde{\theta}_t\|}{\|\frac{\partial F_{state}(\mathbf{x}_{t+1}, \mathbf{z}_t, \theta)}{\partial \mathbf{z}} \tilde{\mathbf{z}}\|}}, \quad \text{and}, \quad \rho_1 = \sqrt{\frac{\|(\nu)^T \frac{\partial F_{state}(\mathbf{x}_{t+1}, \mathbf{z}_t, \theta)}{\theta}\|}{\|\nu\|}}. \tag{5}$$

Note that, initially, $\tilde{\mathbf{z}}_0 = 0$ and $\tilde{\Theta}_0 = 0$, which, as argued in (Tallec and Ollivier, 2017a), yields unbiased estimates at time $t = 0$. Given the construction of the UORO procedure, all subsequent estimates can be shown, by induction, to be unbiased as well.

3.3. Sparse Attentive Backtracking

Sparse attentive backtracking (SAB) (Ke et al., 2017) is an approach that incorporates a differentiable, sparse attention mechanism to select between previously computed states. SAB modifies the traditional BP backward pass as follows: 1) during the RNN's forward pass, the neural system should manage a memory that selects, at most, a sparse subset of past memories , which is referred to as “sparse retrieval”, and, 2) during the backward pass, gradients are only propagated via the sparse subset memory to surrounding units (referred to as “sparse replay”). As a result, these changes can yield better generalization with improved memorization ability (Ke et al., 2017).

3.4. Difference target propagation

Unlike other variants, difference target propagation (DTP) (Lee et al., 2015; Manchev and Spratling, 2020) tries to side-step the limitations of BP while working with precise derivatives and smooth networks to make credit assignment less troublesome. DTP calculates local losses under the total discrepancy framework (Ororbia and Mali, 2019). DTP first computes a global target with respect to a global Loss (L) which, for an n layer model, is calculated as:

$$\mathbf{\hat{z}_n} = \mathbf{z_n} - \hat{\alpha}\frac{\partial L}{\partial z_n}. \tag{6}$$

In the case of RNNs, DTP can be seen as setting a target for the final time step, replacing $\hat{z}_n$ with $\mathbf{\hat{z}}_{final_t}$ and $\mathbf{z}_n$ with $\mathbf{z}_{final_t}$, respectively. As a result, we get the update equation below:

$$\mathbf{\hat{z}_{final_t}} = \mathbf{z_{final_t}} - \hat{\alpha}\frac{\partial L}{\partial z_{final_t}} \tag{7}$$

which is used to compute targets for lower layers over local time-steps (backwards in time for RNNs). DTP propagates global error to intermediate blocks by creating local error signals or targets and, in RNNs, DTP performs local optimization to bring $\mathbf{\hat{z_t}}$ closer to $\mathbf{z_t}$. For vanilla RNNs, the function (F), takes input ($\mathbf{x}_t$) and hidden state at previous time ($\mathbf{z}_{t-1}$) and can thus be represented as $F(\mathbf{x}_t, \mathbf{z}_{t-1})$. Finally, the goal is to have create an inverse of F which we denote as G such that it takes $\mathbf{x}_t$ and $\mathbf{z}_t$ as input and approximates $\mathbf{z}_{t-1}$. Similarly, we approximate the gradients backward in time under the same scheme, yielding a useful method for conducting credit assignment.

4. Models

All of the models used in this paper are first-order RNNs that utilize a rank-1 approximation of second-order connections (Giles et al., 1990) (given that higher order weights and their approximations are argued to better capture the structural properties of grammar). As motivated at the start of this paper, we are interested in comparing the dynamic counting and next-step prediction ability of RNNs trained with different learning algorithms. Furthermore, we also test their ability to learn the Dyck-2 language and their capability to emulate the operation of the pushdown automaton (PDA).

An Elman (or vanilla) RNN architecture (Elman, 1993) is a function that takes an input $\mathbf{x}_t \in \mathcal{R}^{d\times 1}$ (input has d features) and a previous hidden state representation $\mathbf{z}_{t-1} \in \mathcal{R}^{m\times 1}$ (the hidden layer contains m neurons) to produce the next hidden state $\mathbf{z}_t \in \mathcal{R}^{m\times 1}$. Formally, this entails:

$$\mathbf{z}_t = f(W \cdot \mathbf{x}_t + U \cdot \mathbf{z}_{t-1} + \mathbf{b}) \tag{8}$$

where $\cdot$ denotes matrix/vector multiplication. Note that parameter dimensionalities are: $W \in \mathcal{R}^{m\times d}$ (the token embedding matrix), $U \in \mathcal{R}^{m\times m}$ (the recurrent memory matrix), and $\mathbf{b} \in \mathcal{R}^{m\times 1}$ (the hidden state bias). An important variant of the RNN, known as the multiplicative integration RNN (MI-RNN), replaces the operator $+$ with the Hadamard product $\odot$ to enhance the encoding capability of the RNN, making them suitable for learning complex grammars (Krause et al., 2016; Mali et al., 2020). As a result, the hidden state for the MI-RNN is calculated as follows:

$$\mathbf{z}_t = f(W \cdot \mathbf{x}_t \odot U \cdot \mathbf{z}_{t-1} + \mathbf{b}) \tag{9}$$

where f is an element-wise activation function.[1] The other models that we investigate are architectural modifications of the above base RNNs, designed to better handle vanishing/exploding gradients. Specifically, we employ the LSTM (Gers and Schmidhuber, 2001), the GRU (Chung et al., 2014), the MI-GRU (Krause et al., 2016), and the MI-LSTM (Krause et al., 2016).

5. Experimental Setup

We adopted the training scheme advocated in prior work (Suzgun et al., 2019a) and extended it to work with various learning algorithms and RNNs. We performed several experiments to evaluate the capability of each algorithm and each RNN model to perform dynamic counting and encode hierarchical representations. Specifically, we created four different synthetic sequence prediction task datasets. Each task was designed in such a way that it tests/examines different capabilities of an RNN. All models use mean square error (MSE) as an objective function and use 0.5 as a threshold criteria for prediction. Whenever we encounter the end of sequence symbol, the RNN is designed to classify the sequence, where the ground truth is a flag variable that indicates if the model prediction is valid or is invalid otherwise. All experiments are repeated 10 times with different random seeds. To ensure a fair and valid model comparison, we share the same seeds across all models.

5.1. Training Details

If we can represent a formal language using fewer hidden neurons such that each unit corresponds to individual transition states in a minimal automata that defines the (target) formal language, such a model is more interpretable (Omlin and Giles, 1996b). This design makes model interpretation and DFA extraction an efficient task since each unit belongs to one transition within an automaton.[2]

All RNNs trained in our experiments were single-layer networks with up to 10 hidden units. We conducted a grid search for the hidden layer size m over the set $[1, 2, 3, 5, 7, 8, 9, 10]$. For parameter optimization, we used a patience scheduling for the learning rate with an initial learning rate $= 0.2$ and divided this rate by two whenever (validation) performance did not increase. We also conducted a grid search to find the ideal optimizer [SGD, RmsProp, Adam, Momentum]. Parameter gradients, under any learning algorithm, were estimated with mini-batches of 50 samples. For the language modeling (LM) tasks, we used the dataset splits provided by (Ke et al., 2017) and performed a grid search for m over the set $[50, 100, 150, 300, 500, 750, 1000]$. Gradients for the LM models were calculated with mini-batches of size 64. All LM model parameters were optimized Adam Kingma and Ba (2014) and used patience scheduling for the learning rate with an initial value of 0.01.

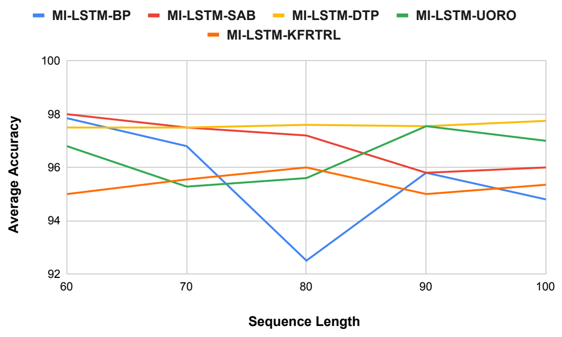
1. In our experiments, we used the scaled variant of the hyperbolic tangent (LeCun et al., 2012).
2. Some studies have used embeddings (Sennhauser and Berwick, 2018; Bernardy, 2018) and the dropout operator (Bernardy, 2018) to improve generalization on longer strings but, ultimately, this makes the model rather complex.

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

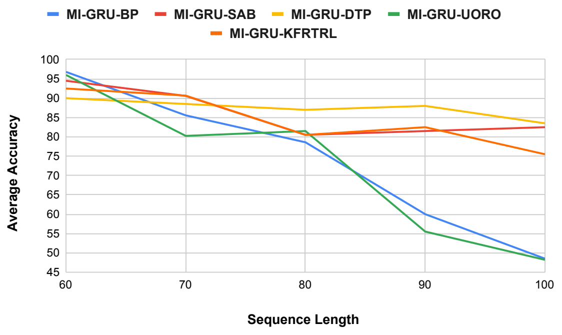
Table 1: Example input-output pairs for the Dyck-1 (left) and Dyck-2 (right) languages.

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.



(a)



(b)

Figure 1: For longer strings from Shuffle-6, a) Average accuracy of the multiplicative integration LSTM (MI-LSTM) with various learning approaches. b) Average accuracy of the multiplicative integration GRU (MI-GRU) with various learning approaches.

6. Datasets and Experimental Results

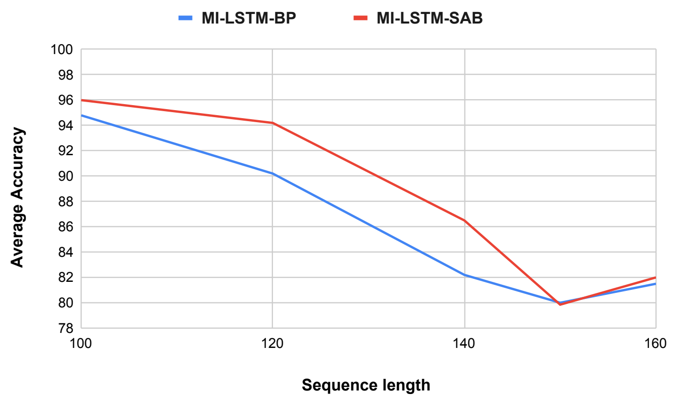
The synthetic dataset used in this work was mainly used to test the capability of RNNs to emulate deterministic k-counter automata. Furthermore, we examine the effect that different learning algorithms have on RNN model representation and generalization. Specifically, we generate hierarchical languages such as the Dyck languages, which have been shown

Task	Model	Training Set		Short Test Set		Long Test Set	
		Max	Med	Max	Med	Max	Med
Dyck-1	MI-LSTM-BP	100	100	100	100	100	100
Dyck-1	MI-LSTM-SAB	100	100	100	100	100	100
Dyck-1	**MI-LSTM-DTP**	**100**	**100**	**100**	**100**	**100**	**100**
Dyck-1	MI-LSTM-UORO	100	100	100	100	100	100
Dyck-1	MI-LSTM-KFRTRL	100	100	100	100	100	100
Shuffle-2	MI-LSTM-BP	100	100	100	100	99.94	99.25
Shuffle-2	MI-LSTM-SAB	100	100	100	100	99.93	99.27
Shuffle-2	**MI-LSTM-DTP**	**100**	**100**	**100**	**100**	**99.98**	**99.40**
Shuffle-2	MI-LSTM-UORO	100	100	100	100	99.95	99.36
Shuffle-2	MI-LSTM-KFRTRL	100	100	100	100	99.92	99.34
Shuffle-6	MI-LSTM-BP	100	100	100	100	99.70	98.10
Shuffle-6	MI-LSTM-SAB	100	100	100	100	99.65	98.00
Shuffle-6	**MI-LSTM-DTP**	**100**	**100**	**100**	**100**	**99.75**	**98.20**
Shuffle-6	MI-LSTM-UORO	100	100	100	100	99.72	97.55
Shuffle-6	MI-LSTM-KFRTRL	100	100	100	100	99.75	98.00
Dyck-2	MI-LSTM-BP	52.10	35.75	49.50	32.00	1.5	0.15
Dyck-2	MI-LSTM-SAB	52.12	35.80	51.50	**32.15**	1.2	0.09
Dyck-2	MI-LSTM-DTP	51.00	35.00	54.15	32.02	1.4	0.15
Dyck-2	**MI-LSTM-UORO**	53.00	36.50	**58.30**	32.00	**1.6**	**0.25**
Dyck-2	MI-LSTM-KFRTRL	**54.00**	**38.00**	49.80	30.00	1.2	0.08

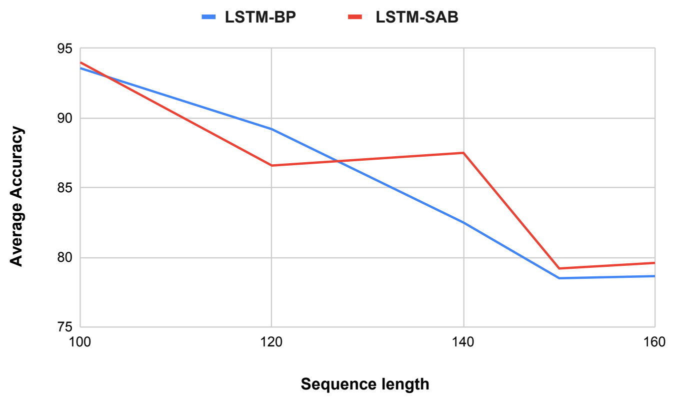
Table 3: The accuracy (correctly classifying a given string) of the MI-LSTM on four grammatical inference tasks (Dyck grammars). Shuffle-2 denotes the shuffle of two Dyck-1 languages defined over different alphabets, and, similarly, Shuffle-6 denotes the shuffle of six Dyck-1 languages defined over different alphabets. Max/median results were obtained from 10 different runs of each model (each run used a different random seed). We note that the MI-LSTM outperformed other RNNs and achieved full accuracy on the shorter test set besides the Dyck-2 grammar, which is strictly a grammar that requires a stack-like mechanism.

BPC	PTB	Text-8
MI-LSTM-BP	1.37	1.42
MI-LSTM-SAB	1.38	1.44
MI-LSTM-DTP	2.20	2.35
MI-LSTM-UORO	2.62	2.80
MI-LSTM-KF-RTRL	1.76	1.87

Table 4: Language modeling experimental results (BPC) with the MI-LSTM.



(a)



(b)

Figure 2: a) Average accuracy of MI-LSTM-BP & MI-LSTM-SAB when tested on Shuffle-6. b) Average accuracy of LSTM-BP & LSTM-SAB when tested on Shuffle-6.

Method	n =252	n =512
MILSTM-BP	80	78.5
MILSTM-SAB	82	81.50
MILSTM-DTP	100	100
MILSTM-UORO	100	95.00
MILSTM-KF-RTRL	92	90

Table 5: MI-LSTM performance with various learning algorithms on very long strings for Dyck-1 grammars. For all MI-LSTM models, we report best accuracy .

to be a promising benchmark (Suzgun et al., 2019b,a) for testing modern day RNNs on grammatical inference. The Dyck languages can be defined in the following manner: let $\Sigma = \{[,]\}$ be the alphabet consisting of symbols [and] and let Σ^* denote its Kleene closure.

A Dyck language is then specified as follows:

$$\{n \in \Sigma^* | \text{ all prefixes of } n \text{ contain no more symbol ']' than symbol '[' } \& \ cnt('[', n) = cnt(']', n)\}$$

where $cnt(<\text{ symbol }>, n)$ is the frequency of <symbol > in n. We can also define Dyck languages via CFGs with a single non-terminal S and the production rule: S $\rightarrow \epsilon|"["S"]"S$, where S is either the empty set or an element of the Dyck languages. A probabilistic CFG for D_2 can be written as:

$$S \rightarrow \begin{cases} (\,S\,) & \text{with probability } \frac{p}{2} \\ [\,S\,] & \text{with probability } \frac{p}{2} \\ S\,S & \text{with probability } p_1 \\ \epsilon & \text{with probability } 1-(p+p_1) \end{cases} \tag{10}$$

where ϵ, p, and p_1 are scalar values that are set externally. The data creation process is adopted from (Suzgun et al., 2019a) – we create four synthetic datasets, mainly using Dyck-1 (or D_1) languages, containing well-balanced sequences of opening and closing parenthesis. Next, we create two languages that are the "shuffle" of either two (different) D_1 languages, i.e., we call this the *Shuffle*-2 dataset, or six D_1 languages, i.e., we call this the *Shuffle*-6 dataset. Finally, we explore performance on a Dyck-2 (D_2) language, which is strictly a context-free language that would require an additional stack like mechanism in order to ensure correct recognition. In theory, all modern day RNNs without memory should fail to recognize D_2 languages (Suzgun et al., 2019b; Mali et al., 2020).

Tables 1 and 2 provide the grammar and training examples. The training set contained 10000 distinct sequences of lengths in the range $[2, 50]$, a short test set contained 5000 strings (in the interval $[2, 50]$), and a long test contained 5000 samples of lengths between $[51, 102]$. A very long test set contained 5000 strings in interval $[104, 160]$. Table 3 presents the main results of the best-performing RNN structure (MI-LSTM) trained with different learning algorithms for Dyck-1, Shuffle-2 and Shuffle-6, and the Dyck-2 grammar. Figures 1 and 2 provide a visual breakdown of the average accuracy for the MI-LSTM and MI-GRU models trained with different algorithms. In the appendix, we conduct an ablation study to understand how the model dynamics change when increasing the number of parameters and to further examine the (average) accuracy as a function of model complexity (in terms of the number of neurons). Finally, we conducted experiments in language modeling (LM) since BP is the driving force for large-scale LM models. We perform two experiments on character level Penn treebank (PTB) and text8 and report bits-per-character (BPC) as the performance metric. We followed the experimental setup provided by (Mikolov et al., 2010) for PTB and (Ke et al., 2017) for text8. For text8, we used the first 90M characters for training, the next 5M for validation, and the remaining 5M characters for testing. We trained our models on non-overlapping sequences of length 180. For all experiments, we report the best model (i.e., MI-LSTM) in the main paper in Table 4.

7. Results and Discussion

Our empirical results on the Dyck-1 and the Dyck Shuffle-2 and 6 languages suggest that the LSTM and its higher-order variants are stable across all BPTT alternatives. We remark that, in the presence of noise, models trained using DTP learned stable internal representations – see the appendix for the details related to this finding. Whenever a model achieves perfect accuracy, analysis of the cell and hidden states reveals the dynamic counting capability of the MI-LSTM and LSTM. To further test and probe the robustness of the acquired representations, we tested all variants of RNN, which achieved perfect accuracy on the training set, on long strings. Figure 1 shows a comparison of the MI-LSTM and MI-GRU on unseen Shuffle-6 long strings when trained using different algorithms.

Even though all algorithms achieve similar performance, difference target propagation (DTP) prediction yields the most consistent performance as the length of the sequence increases. In Table 5, we show the best performance of all models on the Dyck-1 grammar, when tested on two separate longer sequence test sets with intervals $[162 - 252]$ and $[253 - 512]$. Each of these test sets consisted of 50 longer strings not seen during training. This indicates that DTP is capable of creating robust and stable representations which we believe will facilitate a much more stable automaton extraction process from trained RNNs using methods such as the L-star algorithm (Weiss et al., 2018b) and K-means clustering Omlin and Giles (1996b). This is encouraging, offering evidence that aligns with the arguments of (Omlin and Giles, 1996b; Mali et al., 2021c).

It is also interesting to see that (as evidenced from the main results and those in the appendix) that, in some cases, the MI-LSTM and the MI-GRU learned to count input sequences; this behaviour is common in applications such as machine translation (Shi et al., 2016; Bau et al., 2018; Dalvi et al., 2019; Suzgun et al., 2019a) and in popular models such as transformers (Hahn, 2020). In Figure 1b, we see that, despite lacking a dynamic counting mechanism, the MI-GRU trained with various algorithms also generalizes better due to its use of second-order connections facilitating better memorization during the learning process. This can be seen as one of the reasons why the GRU achieves comparable results to the LSTM on various NLP tasks (Jozefowicz et al., 2015). Further analysis of the different learning approaches as well as the benefit of second-order connections are included in Appendix A.3.

Prior work in image compression has also shown that higher-order connections, as well as forward propagation learning algorithms (UORO, RTRL), can lead to useful distributed representations (Mali et al., 2021a). This aligns with our findings in Table 3, which shows results for our best model, i.e., the MI-LSTM, when trained on Dyck-2 CFGs; it is interesting to see that approximate higher-order weights do provide a more powerful encoding capability yielding a 10% boost in performance compared to vanilla LSTMs (when tested on the short test set). Since LSTMs are not theoretically capable of recognizing CFGs, it was no surprise that they struggle to recognize the Dyck-2 language, despite having the higher-order synapses. For all RNN variations, we report further experimental results in the appendix where we compare models, with and without higher-order connections, trained using all learning algorithms to further verify the difficulty in learning Dyck-2 (see Appendix A.3). We kept the training procedure identical for all cases in order to provide a fair comparison. In Figure 2, we compare sparse attentive backtracking (SAB) and backprop (BP) used as algorithms to train an approximate higher-order LSTM (MI-LSTM) and one first order variant (LSTM). We observe that the combination of SAB with higher order connections is important for obtaining consistently better performance, especially as the input grammar grows more complex.

Finally, upon examining model performance on the LM tasks, i.e., Table 4, we observe that BP still yields the best result (in terms of measured BPC) with SAB offering a competitive alternative. This result is sensible given that SAB aims to approximate BP but integrates a sparse attention mechanism over time. Unfortunately, the other algorithms, including DTP, struggle to yield good performance (though we note KF-RTRL is not too far away suggesting that it might be the most promising next candidate BP-alternative algorithm, provided that further extensions are developed to improve its overall performance).

8. Conclusion

This paper investigated neural model representation when learning formal languages. Specifically, we analyzed the interaction between approximate higher-order recurrent neural networks (RNNs) and different learning algorithms, explicitly focusing on the model's ability to dynamically count in the context of grammatical inference. Our experiments demonstrate that models trained using UORO and DTP on Dyck languages can achieve stable performance. Furthermore, our work shows that a higher-order LSTM can acquire robust representations consistently across learning algorithms – this has positive downstream implications for utilizing (classical) procedures for extracting viable, interpretable, and optimal state machines from a neural model trained to conduct grammatical inference. We also conducted experiments in language modeling and although our results did not uncover any improvement over backpropagation of errors (BP), we observed that sparse attentive backtracking (SAB) was competitive. If extended and improved upon in future research, the Kronecker factors real-time recurrent learning algorithm could also potentially be a sound online alternative to algorithms such as BP and SAB that unfold/unroll over time.

References

Forough Arabshahi, Zhichu Lu, Sameer Singh, and Animashree Anandkumar. Memory augmented recursive neural networks. *arXiv preprint arXiv:1911.01545*, 2019.

Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. Identifying and controlling important neurons in neural machine translation. *arXiv preprint arXiv:1811.01157*, 2018.

Jean-Phillipe Bernardy. Can recurrent neural networks learn nested recursion? In *Linguistic Issues in Language Technology, Volume 16, 2018*, 2018.

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

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Axel Cleeremans, David Servan-Schreiber, and James L McClelland. Finite state automata and simple recurrent networks. *Neural computation*, 1(3):372–381, 1989.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, Anthony Bau, and James Glass. What is one grain of sand in the desert? analyzing individual neurons in deep nlp models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6309–6317, 2019.

Sreerupa Das, C Lee Giles, and Guo-Zheng Sun. 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, 1992.

Sreerupa Das, C Lee Giles, and Guo-Zheng Sun. Using prior knowledge in a nnpda to learn context-free languages. In *Advances in neural information processing systems*, pages 65–72, 1993.

Colin De la Higuera. *Grammatical inference: learning automata and grammars.* Cambridge University Press, 2010.

Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

Markus Exler, Michael Moser, Josef Pichler, Günter Fleck, and Bernhard Dorninger. Grammatical inference from data exchange files: An experiment on engineering software. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 557–561. IEEE, 2018.

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

F. A. Gers and E. Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, Nov 2001. ISSN 1045-9227. doi: 10.1109/72.963769.

C Lee Giles, Guo-Zheng Sun, Hsing-Hen Chen, Yee-Chun Lee, and Dong Chen. Higher order recurrent networks and grammatical inference. In *Advances in neural information processing systems*, pages 380–387, 1990.

C Lee Giles, Steve Lawrence, and Ah Chung Tsoi. Noisy time series prediction using recurrent neural networks and grammatical inference. *Machine learning*, 44(1-2):161–183, 2001.

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

E Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

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

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in neural information processing systems*, pages 1828–1836, 2015.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Martin Hofmann, Emanuel Kitzelmann, and Ute Schmid. A unifying framework for analysis and evaluation of inductive programming systems. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 55–60. Citeseer, 2009.

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

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International conference on machine learning*, pages 2342–2350, 2015.

Nan Rosemary Ke, A. G., Olexa Bilaniuk, Jonathan Binas, Laurent Charlin, Chris Pal, and Yoshua Bengio. Sparse attentive backtracking: Long-range credit assignment in recurrent networks. *arXiv preprint arXiv:1711.02326*, 2017.

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

John F Kolen. Recurrent networks: State machines or iterated function systems. In *Proceedings of the 1993 Connectionist Models Summer School*, pages 203–210. Hillsdale NJ, 1994.

Samuel A. Korsky and Robert C. Berwick. On the computational power of rnns. *CoRR*, abs/1906.06349, 2019. URL http://arxiv.org/abs/1906.06349.

Ben Krause, Liang Lu, Iain Murray, and Steve Renals. Multiplicative lstm for sequence modelling. *arXiv preprint arXiv:1609.07959*, 2016.

Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. Neural random-access machines. *arXiv preprint arXiv:1511.06392*, 2015.

Hung Le, Truyen Tran, and Svetha Venkatesh. Neural stored-program memory. *arXiv preprint arXiv:1906.08862*, 2019.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 498–515. Springer, 2015.

Dianhuan Lin, Eyal Dechter, Kevin Ellis, Joshua B Tenenbaum, and Stephen H Muggleton. Bias reformulation for one-shot function induction. 2014.

NJ Mackintosh. Blocking of conditioned suppression: Role of the first compound trial. *Journal of Experimental Psychology: Animal Behavior Processes*, 1(4):335, 1975.

Ankur Mali, Alexander Ororbia, Daniel Kifer, and Clyde Lee Giles. Recognizing long grammatical sequences using recurrent networks augmented with an external differentiable stack. *arXiv preprint arXiv:2004.07623*, 2020.

Ankur Mali, Alexander G. Ororbia, Dan Kifer, and C. Lee Giles. An empirical analysis of recurrent learning algorithms in neural lossy image compression systems. In *2021 Data Compression Conference (DCC)*, pages 356–356, 2021a. doi: 10.1109/DCC50243.2021.00073.

Ankur Mali, Alexander G Ororbia, Daniel Kifer, and C Lee Giles. Recognizing and verifying mathematical equations using multiplicative differential neural units. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5006–5015, 2021b.

Ankur Arjun Mali, Alexander G Ororbia II, and C Lee Giles. A neural state pushdown automata. *IEEE Transactions on Artificial Intelligence*, 2021c.

Nikolay Manchev and Michael W Spratling. Target propagation in recurrent neural networks. *Journal of Machine Learning Research*, 21(7):1–33, 2020.

William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A Smith, and Eran Yahav. A formal hierarchy of rnn architectures. *arXiv preprint arXiv:2004.08500*, 2020.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

Michael C Mozer and Sreerupa Das. A connectionist symbol manipulator that discovers the structure of context-free languages. In *Advances in neural information processing systems*, pages 863–870, 1993.

Asier Mujika, Florian Meier, and Angelika Steger. Approximating real-time recurrent learning with random kronecker factors. In *Advances in Neural Information Processing Systems*, pages 6594–6603, 2018.

Hyoungwook Nam, Segwang Kim, and Kyomin Jung. Number sequence prediction problems for evaluating computational powers of neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4626–4633, 2019.

Yann Ollivier, Corentin Tallec, and Guillaume Charpiat. Training recurrent networks online without backtracking. *arXiv preprint arXiv:1507.07680*, 2015.

Christian W Omlin and C Lee Giles. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM (JACM)*, 43(6):937–972, 1996a.

Christian W Omlin and C Lee Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural networks*, 9(1):41–52, 1996b.

Alexander G Ororbia and Ankur Mali. Biologically motivated algorithms for propagating local target representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4651–4658, 2019.

Jordan B Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1-2): 77–105, 1990.

Yasubumi Sakakibara, Michael Brown, Richard Hughey, I Saira Mian, Kimmen Sjölander, Rebecca C Underwood, and David Haussler. Stochastic context-free grammers for trna modeling. *Nucleic acids research*, 22(23):5112–5120, 1994.

David B Searls. The computational linguistics of biological sequences. *Artificial intelligence and molecular biology*, 2:47–120, 1993.

Luzi Sennhauser and Robert C Berwick. Evaluating the ability of lstms to learn context-free grammars. *arXiv preprint arXiv:1811.02611*, 2018.

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

Hava T. Siegelmann and Eduardo D. Sontag. Analog computation via neural networks. *Theor. Comput. Sci.*, 131(2):331–360, 1994.

Dr Stogin, Ankur Mali, Dr Giles, et al. Provably stable interpretable encodings of context free grammars in rnns with a differentiable stack. *arXiv preprint arXiv:2006.03651*, 2020.

Guo-Zheng Sun, C Lee Giles, and Hsing-Hen Chen. The neural network pushdown automaton: Architecture, dynamics and training. In *International School on Neural Networks, Initiated by IIASS and EMFCSC*, pages 296–345. Springer, 1997.

Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M. Shieber. LSTM networks can perform dynamic counting. *CoRR*, abs/1906.03648, 2019a. URL `http://arxiv.org/abs/1906.03648`.

Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M. Shieber. Memory-augmented recurrent neural networks can learn generalized dyck languages, 2019b.

Whitney Tabor. Fractal encoding of context-free grammars in connectionist networks. *Expert Systems*, 17(1):41–56, 2000.

Corentin Tallec and Yann Ollivier. Unbiased online recurrent optimization, 2017a.

Corentin Tallec and Yann Ollivier. Unbiased online recurrent optimization. *CoRR*, abs/1702.05043, 2017b. URL http://arxiv.org/abs/1702.05043.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Cheng Wang and Mathias Niepert. State-regularized recurrent neural networks. In *International Conference on Machine Learning*, pages 6596–6606, 2019.

Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision rnns for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 740–745, 2018a.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *International Conference on Machine Learning*, pages 5247–5256, 2018b.

Wojciech Wieczorek and Olgierd Unold. Use of a novel grammatical inference approach in classification of amyloidogenic hexapeptides. *Computational and mathematical methods in medicine*, 2016, 2016.

Janet Wiles and Jeff Elman. Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of the seventeenth annual conference of the cognitive science society*, number s 482, page 487. Erlbaum Hillsdale, NJ, 1995.

Ronald J Williams and David Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111, 1989.

Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 433, 1995.

Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. Memory architectures in recurrent neural network language models. In *International Conference on Learning Representations*, 2018.

Zheng Zeng, Rodney M Goodman, and Padhraic Smyth. Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks*, 5(2):320–330, 1994.

Appendix A. Ablation Study and Additional Results

In this section, we provide additional results to support the experiments of the main paper. First, we examine the importance of model parameters. Second, we show that adding a small amount of noise can corrupt the internal representation of the recurrent network model and we examine which learning algorithms are resistant to small amounts of noise. Finally, we show the importance of higher-order connections and how other variants of RNNs perform on Dyck grammars.

A.1. On the Importance of Model Parameters

BPTT alternatives, especially DTP and UORO, offer promising performance and stay consistent even when tested over longer strings. Figure 3 examines the changes in performance of our best model (i.e., MI-LSTM) trained using various learning algorithms. We increased model capacity and tested how this hampered the performance when using the different learning algorithms. Based on our experiments, BPTT-based (BP) models struggle initially; however, as network capacity increased, these models start to overcome their initial failures. Nevertheless, this does not mean that simply adding more parameters will yield a better result. In fact, in a few cases, more parameters resulted in poorer generalization. Nevertheless, UORO and DTP resulted in stable performance across trials and model capacity but failed, unfortunately, to match BPTT performance at the highest model capacity.

A.2. Injecting Noise into Internal Representations

We also experimented with injecting Gaussian noise, i.e., $\epsilon \sim \mathcal{N}(\mu = 0, \sigma^2 = 0.2)$ (for each dimension of the model state $\mathbf{z}_t$), into the hidden states whenever the network reached perfect accuracy on the training set in order to test how small perturbations in the model's state signals would affect the overall generalization over longer strings. In Table 6, we show how various learning approaches behave with and without the injection of noise. Surprisingly, DTP and KF-RTRL predictions are rather robust when noise is injected into the RNN's states (with DTP performing the best).

Method	Without Noise	With Noise
MILSTM-BP	100	94.50
MILSTM-SAB	100	93.00
MILSTM-DTP	100	99.60
MILSTM-UORO	100	97.50
MILSTM-KF-RTRL	100	99.00

Table 6: MI-LSTM performance with various learning algorithms on longer strings for Dyck-1 grammars. All MI-LSTM based models are tested with and without gaussian noise .

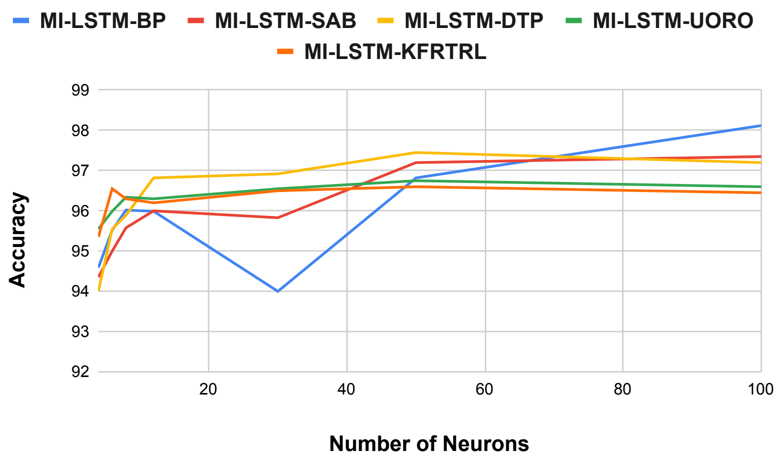


Figure 3: Accuracy as a function of the number of neurons (network capacity). We report mean accuracy for the long string test set (samples from the Shuffle-6 dataset).

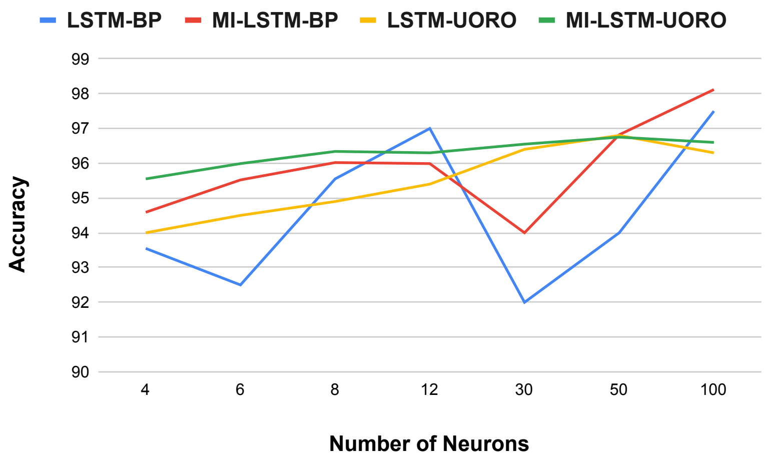


Figure 4: Accuracy as a function of the number of neurons in the hidden layer for BP and UORO. We report mean accuracy for the long string test set (samples from the Shuffle-6 dataset). We compare the LSTM to its multiplicative variant, the MI-LSTM.

A.3. On the Importance of Higher Order Connections

Vanilla (Elman) RNNs and their multiplicative variants failed to match the LSTM and GRU performance, sometimes only achieving 8% accuracy on the simplest Dyck-1 grammar; hence, we excluded them in our plots. Figure 4 shows how second-order weights, when coupled with different learning algorithms, can lead to a steady increase in performance compared to baseline approaches. This is an important finding, which measures the relationship between the choice of internal model structure and the choice of learning algorithm when dealing with complex grammars. Similarly, Table 7 demonstrates the importance of higher-order connections and how they help the GRU match the performance of the LSTM in the task of grammatical inference.

		Training Set		**Short Test Set**		**Long Test Set**	
Task	**Model**	**Max**	**Med**	**Max**	**Med**	**Max**	**Med**
Dyck-1	MI-GRU-BP	100	100	100	100	98.00	85.00
Dyck-1	MI-GRU-SAB	100	100	100	100	96.00	84.50
Dyck-1	**MI-GRU-DTP**	**100**	**100**	**100**	**100**	**100**	**88.50**
Dyck-1	MI-GRU-UORO	100	100	100	100	99	86
Dyck-1	MI-GRU-KFRTRL	100	100	100	100	98.90	87.50
Shuffle-2	MI-GRU-BP	100	100	100	100	94.00	93.25
Shuffle-2	MI-GRU-SAB	100	100	100	100	94.93	92.27
Shuffle-2	**MI-GRU-DTP**	**100**	**100**	**100**	**100**	**97.98**	**95.00**
Shuffle-2	MI-GRU-UORO	100	100	100	100	97.95	95.36
Shuffle-2	MI-GRU-KFRTRL	100	100	100	100	95.92	95.34
Shuffle-6	MI-GRU-BP	100	100	100	96.00	95.70	84.10
Shuffle-6	MI-GRU-SAB	100	100	99.99	97.50	92.65	82.00
Shuffle-6	**MI-GRU-DTP**	**100**	**100**	**100**	**99.99**	**97.75**	**87.50**
Shuffle-6	MI-GRU-UORO	100	100	100	94.00	97.50	87.45
Shuffle-6	MI-GRU-KFRTRL	100	100	100	92	93.75	83.00
Dyck-2	MI-GRU-BP	55.20	32.55	49.50	30.00	1.55	0.09
Dyck-2	MI-GRU-SAB	55.00	30.85	47.00	**34.15**	1.5	0.08
Dyck-2	MI-GRU-DTP	54.00	34.00	52.45	32.08	1.4	0.14
Dyck-2	**MI-GRU-UORO**	**59.00**	35.50	**52.30**	34.00	**1.65**	**0.20**
Dyck-2	MI-GRU-KFRTRL	59.00	**36.00**	55.00	31.20	1.3	0.12

Table 7: The performance (accuracy) of the MI-GRU across four grammatical inference 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. Max/median results were obtained from 10 runs of each model with a different random seed used for each run.