Proceedings of Machine Learning Research 153:130–153, 2021 Proceedings of the 15th ICGI

Recognizing Long Grammatical Sequences using Recurrent Networks Augmented with an External Differentiable Stack

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

Recurrent neural networks (RNNs) are widely used for sequence modeling, generation, and prediction. Despite success in applications such as machine translation and voice recognition, these stateful models have several critical shortcomings. Specifically, RNNs struggle to recognize very long sequences, which limits their applicability in many important temporal processing and time series forecasting problems. For example, RNNs struggle in recognizing complex context free languages (CFLs), unable to reach 100% accuracy on the training set. One way to address these shortcomings is to couple an RNN with an external, differentiable memory structure, such as a stack. However, differentiable memories in prior work have neither been extensively studied on complex CFLs nor tested on very long sequences that have been seen on training. In fact, earlier work has shown that continuous differentiable memory structures struggle in recognizing complex CFLs over very long sequences. In this paper, we improve the memory-augmented RNN with new architectural and state updating mechanisms that learn to properly balance the use of latent states with external memory. Our improved RNN models exhibit improved generalization and are able to classify long strings generated by complex hierarchical context free grammars (CFGs). We evaluate our models on CFGs, including the Dyck languages, as well as on the Penn Treebank language modelling dataset, achieving stable, robust performance on these benchmarks. Furthermore, we show that our proposed memory-augmented networks are able retain information over long sequences leading to improved generalization for strings up to length 160.

Keywords: Recurrent neural network, Memory augmented network, Dyck languages, Context free language, Pushdown automata

1. Introduction

Recurrent neural networks (RNNs) are stateful models that extract temporal dependencies from sequential data. They have been widely used in various natural language processing (NLP) and computer vision tasks, such as machine translation (Kalchbrenner and Blunsom, 2013; Bahdanau et al., 2015; Sutskever et al., 2014), language modeling (Mikolov et al., 2010; Sundermeyer et al., 2012), multimodal language modelling (Kiros et al., 2014; Ororbia et al., 2019) and speech recognition (Graves et al., 2013). In theory, first order RNNs with the desired weights (arranged as 2D weight matrices) and infinite precision have been shown to be as powerful as a pushdown automaton (PDA) (Korsky and Berwick, 2019; Kremer, 1995; Frasconi and Gori, 1996) and can be considered to be computationally universal models (Siegelmann and Sontag, 1994, 1995). However, a higher order model (such as second order or tensor RNNs) can directly map higher order recurrent weights to state machine transition rules, thus offering more interpretable representations. This is also known as structural encoding, and tensor RNNs are viable solution for same. (Omlin and Giles, 1996; Horne and Hush, 1994; Carrasco et al., 2000) proved that any deterministic finite state machine can be stably constructed with a 2nd order or tensor RNN.

To this day, languages that are recognizable by first order RNNs and even transformers, as well as the limits of their computational power, are still being explored. Recently, (Merrill, 2019) theoretically demonstrated the expressiveness of RNNs with finite precision under asymptotic conditions. Empirically, it has been demonstrated that LSTMs possess the capability of recognizing simple context free languages (such as $a^n b^n$) and context-sensitive languages (such as $a^n b^n c^n$) by using a complex counting mechanism (Gers and Schmidhuber, 2001; Weiss et al., 2018a) whereas GRUs struggle to perform dynamic counting (Weiss et al., 2018a). While some of the languages from the Chomsky hierarchy (Chomsky and Lightfoot, 2002) can be modelled using counting, counting does not capture all of the structural properties that underlie natural language, i.e., languages with hierarchical or nested structure. (Chomsky and Lightfoot, 2002; Chomsky, 1962) categorized these nested languages as context free languages (CFLs). To recognize CFLs, a counting mechanism with finite memory is insufficient and a stack is required in order to correctly recognize these grammars. Based on formal language theory, Dyck languages D_n, where $n > 1$, offer nested structure and are categorized as CFLs(Nivat, 1970). (Suzgun et al., 2019a) showed that first order RNNs, e.g., GRU/LSTM, perform well on Dyck languages (D_1) (and various combinations) when $n = 1$. However, these models struggle when $n > 1$. In contrast, theoretically, a pushdown automaton with a stack can correctly recognize any CFG that is also known to be the homomorphic image of a regular D_n language (Chomsky and Schützenberger, 1959; Chomsky, 1962).

Prior work has focused on using first order RNNs, with and without external memory, to learn Dyck languages up to length 2 (Zaremba et al., 2016; Hao et al., 2018; Deleu and Dureau, 2016) but none have achieved reasonable performance on long sequences. In work similar to ours, (Suzgun et al., 2019b) proposed a mechanism for training memory-augmented RNNs for the Dyck languages. Other work has focused on a different class of RNNs, often referred to as tensor RNNs, which are able to learn CFLs, even D_2 languages, on long sequences not seen during training (Mali et al., 2019). These models appear to offer better encoding/representations for recognizing context free languages (Mali et al., 2019;

Das et al., 1993, 1992). However, due to their computational complexity and difficulty in implementation/training, higher order RNNs are not widely used in practice. The recent success of higher order RNNs with memory (Mali et al., 2019) in recognizing CFGs motivates our current work. Here, we focus on maintaining the representational capability of tensor RNNs while minimizing computational overhead. To do this, we propose a first order approximation of higher order weights as well as their updates, reducing computational cost. To better understand this approach, we empirically investigate the computational capabilities of first order RNNs used in practice, with and without external memory, for the task of recognizing complex CFLs. Then, we extend RNNs to those with higher order weights and propose several new improved memory-augmented RNN models. We call this family of models that DiffStk-RNN family and show that these models offer improved performance in recognizing complex CFLs. The contributions of this paper are:

- We introduce five new stack-augmented, recurrent neural networks and, within a non-traditional framework of next-step prediction, we develop a schema that facilitates stronger, iterative error-correction for improved training of CFG recognition models.
- We show that approximating higher order RNNs with stack memory offer the best results when classifying long CFL strings while reducing computation time.
- We investigate, for the first time, the performance of first order RNNs on complex, long string CFLS and show the importance of using negative string examples when training for CFG recognition.
- We show that our models, with only finite precision, can emulate the dynamics of a PDA to effectively learn long string CFLs, mainly $D_>1$ languages, and furthermore perform well on language modeling using the Penn Treebank benchmark dataset.
- We propose an efficient scheme for training stack-augmented RNNs that helps preserve memory over longer string lengths. In addition, we demonstrate the importance of noise and the need for improved handling of NO-OP operations on the RNN's stack when preserving memory over long time spans.

2. Related Work

Historically, grammatical inference (Gold, 1967) has been at the core of formal language theory and could be considered to be one of the most fundamental pathways towards understanding the important properties of natural languages. A summary of most 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 areas ranging from financial time series to genetics and bioinformatics (Searls, 1993; Sakakibara et al., 1994) to software data exchange (Giles et al., 2001; Wieczorek and Unold, 2016; Exler et al., 2018). In recent times, state-less models such as transformers have demonstrated positive results in recognizing formal languages Bhattamishra et al. (2020); Weiss et al. (2021), however theoretical capability in recognizing complex grammars of such models are often debatable Hahn (2020). In contrast many stateful neural models take the form of a first order recurrent neural network (RNN) and are taught 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). However, from a theoretical perspective, RNNs augmented with an external memory have historically been considered efficient in recognizing context free languages (CFLs), such as tensor RNNs (Das et al., 1993; Pollack, 1990; Sun et al., 1997; Mali et al., 2019), or, more recently, first order RNNs augmented 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). Despite the positive recognition results, prior work has not focused on examining generalization over very long strings but has highlighted difficulty in training memory-augmented models. In contrast, (Zeng et al., 1994; Mali et al., 2019; Suzgun et al., 2019b) and (Gers and Schmidhuber, 2001) demonstrated promising results in generalizing beyond the training dataset with simple CFLs. Simple CFLs are bounded context-free languages that can be easily recognized using Finite automata or counter automata by increasing the model capacity without requiring any external memory. In contrast, complex CFLs require external memory to capture hierarchical representation in languages and cannot be easily recognized by finite machines.

Recent work on differentiable stacks (the stack-RNN) (Joulin and Mikolov, 2015), which is closely related to this work, tests on real natural language modeling tasks and on learning simple algorithmic patterns. 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 including stacks, queues, and doubly-linked lists. Nonetheless, none of these prior efforts have focused on evaluating models on complex and long CFLs. While other memory-augmented models have been proposed to solve CFLS, such as neural random access memory (NRAM) (Kurach et al., 2015) and the neural Turing machine (NTM) (Graves et al., 2014), these models face instability issues and are quite difficult to train and scale to real world problems. To truly evaluate the generalization ability of RNNs and other memory-augmented models on CFLs, it is critical that we test them on more complex and longer CFLs such as the Dyck languages (Suzgun et al., 2019a).

3. Recurrent Neural Networks Augmented with an External Stack

In this section, we describe a set of stack-augmented RNNs, motivated by neural systems designed to emulate state machines with pushdown dynamics (Das et al., 1993; Mali et al., 2019; Joulin and Mikolov, 2015; Grefenstette et al., 2015; Hao et al., 2018; Suzgun et al., 2019b). Unlike previously proposed models, ours are simple and easy to train.

We approach the problem of CFL recognition in a way different to how it has been approached historically. Classically, an RNN designed for recognizing CFLs first encodes the entire string sequence in its internal state and then subsequently classifies it as correct or incorrect, i.e., the input to the recognition (binary) classifier was the RNN's final computed hidden vector summary of the sequence. In contrast, we design a next-step prediction scheme that entails: 1) predicting the recognition label y of the string at each time step while measuring efficacy with a mean squared error (MSE) loss, and 2) averaging step-wise predictions once the end symbol/token is encountered. Whenever the predicted value is

< 0.5, the string is considered invalid otherwise it is flagged as valid. Our intuition behind using next step prediction was to create a better training scheme for CFL recognition, allowing the RNN to recover from previous mistakes when iteratively recognizing a sequence. Our experiments suggest this helps in learning complex and longer sequences for CFLs.

Ideally, since we want all of our networks to operate like PDAs, we formulate our model in the spirit of classical pushdown networks. As such, we first start by defining an M-state PDA as a 7-tuple $(Q, \Sigma, \Gamma, \delta, q^0, \perp, F)$ where:

- $\Sigma = \{a^1, \cdots, a^l, \cdots, a^L\}$ is the input alphabet
- $Q = \{s^1, \cdots, s^m, \cdots, s^M\}$ is the finite set of states
- Γ is known as stack alphabet (a finite set of tokens)
- q^0 is the start state
- $\perp$ is the initial stack symbol
- $F \subseteq Q$ is the set of accepting states
- $\delta \subseteq Q \times (\Sigma \cup) | \times \Gamma \rightarrow Q \times \Gamma^*)$ is the state transition.

3.1. The DiffStk-RNN Pushdown Network

Our pushdown network will be referred to as a "Differentiable Stack-RNN", in brief we call it the DiffStk-RNN (to distinguish it from the original model "StackRNN"(Joulin and Mikolov, 2015) and the recent (Suzgun et al., 2019b) model). DiffStk-RNN learns to control an external stack data structure and consists of an input layer, a hidden layer with recurrent connections, and an output layer. The RNN processes, at each time step t in a sequence of length T, a symbol/token $\mathbf{x}_t \in \{0,1\}^{d \times 1}$ (the one-hot encoding of the symbol) where d is the total number of unique symbols in the dataset/corpus, i.e., the size of the alphabet. In essence, its goal is to jointly predict $\mathbf{x}_t$ and the global validity flag $y_t \in \{0,1\}$ (a binary value) given a history of symbols observed thus far $\mathbf{x}_{<t}$ (using the cross-entropy loss that is characteristic of RNN language models). Assuming the CFL-recognizing RNN's prediction (at step t) of the recognition label is $\hat{y}_t \in [0,1]$ (a scalar) and its prediction of the next token is $\hat{\mathbf{x}}_t$ (a probability simplex), then the complete loss for a CFL sample sequence string is:

$$\mathcal{L}(\Theta) = \sum_{t=1}^{T} \beta_x \left[\sum_i -(\mathbf{x}_t \otimes \log(\hat{\mathbf{x}}_t))[i] \right] + \beta_y \left[\frac{1}{2}(\hat{y}_t - y)^2 \right] \tag{1}$$

where i indicates retrieval of the ith scalar value in a vector. β_x controls the importance of the symbol-prediction term of the loss while β_y controls the importance of the validity prediction term (we set both $\beta_x = \beta_y = 1$ in this paper). As the RNN processes symbols one by one, it computes its hidden state as follows (biases omitted for clarity):

$$\mathbf{z}_t = f_1(U \cdot \mathbf{x}_t + R \cdot \mathbf{z}_{t-1}) \tag{2}$$

where $\cdot$ indicates matrix/vector multiplication. Note that $f_1(\circ)$ is hyperbolic tangent (tanh) activation function, specifically, the scaled variant of it (LeCun et al., 2012), i.e., $f_1(x) = 1.7519 \times tanh(2/3 \times x)$, applied coordinate-wise. Note that U is a $m \times d$ token embedding matrix and R is a $m \times m$ recurrent weight matrix. The number of different tokens is d while m is the number of hidden units in the state $\mathbf{z}_t$. Based on its current state at step t, the

RNN outputs probability distributions $\hat{\mathbf{x}}_t$ and $\hat{y}_t$ for the next step in the following manner:

$$\hat{\mathbf{x}}_t = softmax(V \cdot \mathbf{z}_t) = \frac{\exp(V \cdot \mathbf{z}_t)}{\sum_i \exp(V\mathbf{z}_t)[i]}, \text{ and, } \hat{y} = \sigma(Q \cdot \mathbf{z}_t) \tag{3}$$

where $\sigma(x) = 1/(1 + exp(-x))$ is the logistic sigmoid, $V \in \mathcal{R}^{d \times m}$ is the token prediction matrix, and $Q \in \mathcal{R}^{1 \times m}$ is the validity prediction vector.

As discussed earlier in the paper, pure stateful 1st order RNNs (such as the one presented in Equation 2), struggle to recognize complex CFLs. If we were to raise the order of the network and equip it with an external memory, however, then it would be theoretically possible to recognize any CFL in general (Giles et al., 1992; Mali et al., 2019; Zeng et al., 1994). To raise the order of the network without incurring too great a computational cost, we introduce an approximation of higher order tensor behavior through multiplicative operations – we provide the details of this approximation in the appendix. To address the memory issue, we augment the RNN with a stack, which serves as an external persistent memory structure which has only its topmost element readily accessible to the RNN controller. In this case, such an RNN has basic three operations it may perform on the stack: 1) PUSH: adds elements, 2) POP: removes elements, and 3) NO-OP (NoOP): does nothing. In essence, we want the stack to carry information to the hidden layer of an RNN (acting as the controller). While the original StackRNN does this according to Equation 2, our DiffStk-RNN makes use of an additional term added to the hidden state update equation, plus other improvements such as better handling of NoOP and negative sampling to make it stable. Specifically, we compute the next hidden state of our model according to the following equation:

$$\hat{\mathbf{z}}_{t-1} = \mathbf{z}_{t-1} + P \cdot \mathbf{S}_{t-1}[0], \quad \mathbf{z}_t = f_1(U \cdot \mathbf{x}_t + R \cdot \hat{\mathbf{z}}_{t-1}) \tag{4}$$

where P is $m \times 3$ recurrent matrix and $\mathbf{S}_{t-1}[0]$ is the top-most element of stack $\mathbf{S}$. This shows how the stack-augmented term is integrated into the hidden state calculation. To specify the stack itself, we start by denoting a_t as a 3-dimensional variable representing each action taken on the stack, which is dependent on hidden state $\mathbf{z}_t$

$$\mathbf{a}_t = softmax(A \cdot \mathbf{z}_t) \tag{5}$$

where A is a $3 \times m$ state-to-action matrix. We store the top element of the stack at position 0, with value $\mathbf{S}_t[0]$, via the following:

$$\mathbf{S}_t[0] = \mathbf{a}_t[PUSH]\sigma(D \cdot \mathbf{z}_t) + \mathbf{a}_t[POP]\mathbf{S}_{t-1}[1] + \mathbf{a}_t[NoOP]\mathbf{S}_{t-1}[0] \tag{6}$$

where the symbols PUSH, POP, and NoOP correspond to the unique integer indices 0, 1, and 2 that access the specific action value in their respective slot. D is a $1 \times m$ matrix. If $\mathbf{a}_t[PUSH] = 1$ we add element to the top of the stack and if $\mathbf{a}_t[POP] = 1$ we remove the element at top of the stack and move the stack upwards. Similarly for elements stored at depth $i > 0$ in the stack, the rule is as follows:

$$\mathbf{S}_t[i] = \mathbf{a}_t[PUSH]\mathbf{S}_{t-1}[i-1] + \mathbf{a}_t[POP]\mathbf{S}_{t-1}[i+1]. \tag{7}$$

Figure 1 shows the architecture of our base DiffStk network. Note that for more complex hidden state functions used to compute $\mathbf{z}_t$ (such as an LSTM state function), integration with a stack is similar – details of these models in the family are provided in the appendix.

Adding Noise to the Hidden Weights: Noise plays a crucial role in regularizing RNNs. Besides regularization, it has also been shown to improve the model's forecasting horizon yielding better generalization (Jim et al., 1996; Goodfellow et al., 2016; Noh et al., 2017; Mali et al., 2020; Mali et al., 2019). Motivated by this result, we modify the state update equation as follows:

$$\hat{\mathbf{z}}_{t-1} = \mathbf{z}_{t-1} + P \cdot \mathbf{s}_{t-1}[0] + \epsilon, \quad \forall i, \epsilon[i] \sim \mathcal{N}(\mu, \sigma^2) \tag{8}$$

where μ (mean) and σ^2 (variance) are user-set meta-parameters to control the strength of the additive Gaussian regularization noise. Based on our experiments, we found that injecting a small amount of noise into the state improves the model's generalization although it can sometimes slow down convergence. [1]

Carry Forward State Update: Whenever there exists a symbol that leads to more than one No-OP operation on the stack, we propose carrying forward the previous hidden state. Empirically, we found that this helps whenever the RNN encounters too many No-OPs in the input strings. This can be considered as approximation of a "reject" state, which have been shown to stabilize the learning process of higher order RNNs (Das et al., 1993; Sun et al., 1998). We keep a counter which indicates how many NoOPs the network has seen so far, which is used to trigger the update equation. With this counter-based trigger integrated, the full hidden state update equation can then be written as follows:

$$\mathbf{z}_t = u_t f_1(U \cdot \mathbf{x}_t + R \cdot \hat{\mathbf{z}}_{t-1}) + (1 - u_t)\hat{\mathbf{z}}_{t-1}, \quad u_t = \begin{cases} u_t = 0 \text{ and } c_t > 1, & \text{No-OP} \\ u_t = 1, & \text{PUSH/POP} \end{cases} \tag{9}$$

where u_t is a scalar: 0 indicates that the RNN is to copy the previous state while 1 indicates that the state is to be updated/overwritten and c_t is the counter which keeps track of No-oP operations [2]. In appendix, we show that models using noise in conjunction with carry forward states work better than models that do not. We report accuracy over 10 trials on a D_2 grammar. Observe that our approach consistently demonstrates improved memory retention over longer pattern sequence lengths.

Negative Sampling: Based on preliminary experimentation, we found it important to provide negative samples to a CFG-recognition network during both the training and testing phases (whether using our next-step prediction scheme or the classical scheme of predicting the label at the end). While the rough ratio of positive to negative samples can be controlled when first sampling a target CFG to create a training dataset, most of the negative samples produced by this process are simply too different from the positive cases, or rather, are simply too easy to distinguish from positive cases. In order to truly improve the generalization ability of our RNN models, we mix with the sampled negative cases

1. Other regularization schemes were tried, such as drop-out, batch normalization, and layer normalization. However, these did not help generalization performance and thus we focus on state-injected additive noise.
2. We maintain counter variable c_t just to keep track of NoOPs which helps facilitate decision-making in the RNN and is based on S_t

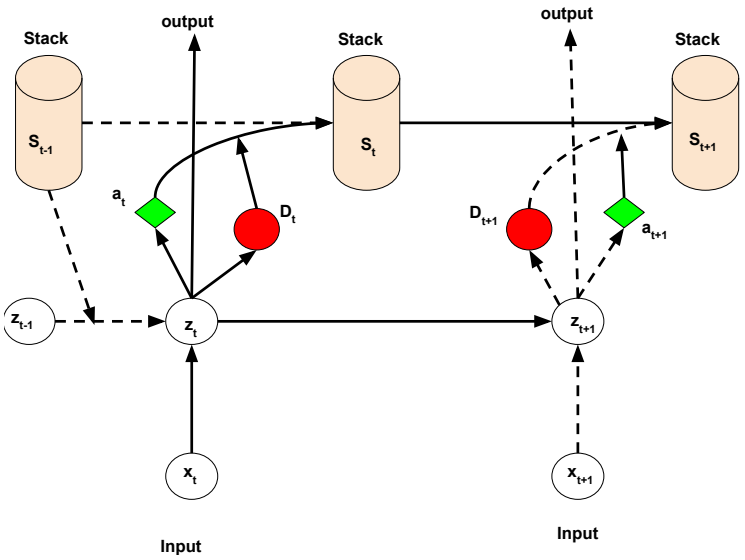

Figure 1: Architecture of the DiffStk-RNN network shown in the act of processing data, updating, and correcting the stack over 2 time steps. $\mathbf{z}_t$ is the hidden state of RNN, $\mathbf{S}_t$ is the stack, $\mathbf{a}_t$ is the action neuron, and dotted lines represents past and future updates.

some number of "difficult" (negative) samples, which are similar/close to positive strings but would still be rejected by the target CFG. For instance for grammar $a^n b^n$, aaaaabbbab, aaaaaaabbbbabb could be considered as difficult negative examples, since only one character changes the notion. In contrast strings such as baaa, aaabaa are much simpler one can be recognized easily with some prior structures knowledge.

In order to facilitate the generation of high-difficulty negative samples, we utilize an oracle which recognizes any given language. We then randomly swap around 1 or 3 characters (randomly selected uniformly) in the input strings and use the oracle to check whether or not the generated grammar is negative (generating the correct rejection label). We then generate and mix in these synthetic negative samples to the dataset while still preserving a balance between the number of positive and negative examples in the original training set. Specifically, we generate a new number of difficult negative samples (Ns) and swap out the same number from the original pool of negative strings. Specifically, prior to training, we stochastically set Ns to be between 15-30% of the total number of original negative samples, generate these new examples based on the scheme above, and swap out the same number of randomly chosen original negative strings with the new, more difficult ones.

Based on our experiments, we find that training with our form of negative sampling improves the performance of an RNN model on longer strings. We provide detailed ablation study in Appendix B and show how adding these components helps in model generalization. In Appendix A, we provide further details on how to extend modern day RNNs with our version of the differentiable stack and conduct ablation study to show the importance of each component we introduce.

4. Datasets

As noted earlier, $D >_1$ denotes all complex context free languages (CFLs) that require a stack-like structure in order to ensure complete and accurate recognition. Furthermore, note that a simple counting mechanism (which is what most first-order RNNs implement) is insufficient to recognize these languages. In light of this, we created various test sets in order to understand the true limits of first order RNNs when recognizing complex CFLs and to test our proposed models so as to observe potential gains in generalization.

4.1. Context-Free Languages

We experimented with various complex grammars such as the Dyck languages. The Dyck languages can be defined in the following manner. Let $\Sigma = [,]$ be the alphabet consisting of symbol [and] and let Σ^* denote its Kleene closure. A Dyck language is defined as:

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

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. From this perspective, the production rule is defined in the following manner. S $\rightarrow \epsilon |$"["S"]"S, where S is either empty set or an element of the Dyck languages. A probabilistic, context-free grammar for D_2 can then 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 set externally.

We tested various Dyck languages such as D_2, D_3, and D_6. For the D_2 and D_3 grammars, we created a dataset containing 6230 training samples (of length T less than or equal to 55), 1000 for validation ($20 < T \leq 70$, and 3000 samples for testing ($55 < T \leq 102$). However, for D_6, which is a much more complex grammar, we created a dataset containing far more training samples - 15000 for training with 2000 set aside for validation and 4000 for test. The proportions of string lengths in all splits for D_6 was kept identical. Following prior work, we also tested our models on the palindrome language (Zeng et al., 1994; Giles et al., 1992; Gers and Schmidhuber, 2001; Mali et al., 2019).

4.2. Language Modeling

In addition to CFGs, we evaluate all of our stack-augmented models on the Penn TreeBank word level language modeling task (Mikolov et al., 2010). All models trained on this dataset consisted of 100 hidden units and were trained over 50 epochs using the Adam optimizer (also with patience scheduling for the learning rate) [3]. Dataset splits and settings was chosen based on prior related work (Joulin and Mikolov, 2015). Our models were able to

3. We ensured that memory-augmented RNN parameters were comparable with the LSTM

match the performance of the LSTM and SRCN (Joulin and Mikolov, 2015). It is important to note that this experiment is not designed to obtain state of the art results but rather to show how various architectures work with a similar number of parameters.

5. Experiments

To evaluate our models and empirically investigate the computational capabilities of first order RNNs, we tested all models on the Dyck languages, which need automata with memory in order to ensure successful recognition of strings of arbitrary length. We created 2000 samples per set (train/dev/test) for any given Dyck grammar. We perform binary classification and test whether any string is correctly or incorrectly classified. To achieve this, we collect predictions using a trained RNN (performing overall inference character for any particular grammar), and when we reach the end of the string, we average over each prediction made per character in the sequence. When the model correctly predicts the input symbol/character, we assign +1 or 0 and collect this until the end of the symbol is encountered. If the mean prediction is greater than 0.5, the string is valid, else it is invalid [4].

In addition, we also tested these models on a real-world problem, i.e., the Penn Treebank language modeling benchmark. To our knowledge, this is the first work to extensively compare a large variety of first order RNNs on complex longer CFLs task as well as on a real world benchmark. We followed two training processes, one was sequential (Gers and Schmidhuber, 2001; Mikolov et al., 2010), which is widely used in natural language processing related tasks, while the other was based on incremental learning (Giles et al., 1992; Mali et al., 2019) or curriculum learning (Bengio et al., 2009), which is an alternative training scheme for tensor RNNs. We compare these two variants and show that sequential prediction has a slight advantage over incremental learning.

5.1. Training

All RNN models trained (baselines included the GRU, LSTM, the Δ-RNN (Ororbia II et al., 2017), the baby neural turing machine (Suzgun et al., 2019b), and previously-proposed stack RNN models, e.g., Stack RNN (Joulin and Mikolov, 2015) and Stack-RNN+Softmax (Suzgun et al., 2019b)) were designed to have a single layer with 8 hidden units. RNN weights were adjusted using gradients computed via back-propagation through time (BPTT) (with a look-back that extended 50 steps) similar to that of (Mikolov et al., 2010). Gradients were hard-clipped to have a maximum magnitude of 15 to ensure that they did not explode. Adam (Kingma and Ba, 2014) was used to update the weights using an initial learning rate $2e-3$. A patience schedule was used to adjust the learning rate; if a gain was not observed in validation within the last three times it was checked, the learning rate was halved. Training hyper-parameters match the setting proposed by (Joulin and Mikolov, 2015) and by running multiple simulations to find optimal parameters. Optimal settings are extracted by observing loss over validation for k simulations, where k ranges between 10 and 35. We ran experimental simulations 10 times, i.e., each run used a unique seed.

4. Since this is a supervised learning task, we have access to ground truth at each instance; this gives ground truth access at each step during the post-processing stage.

RNN Models	*Train*		*Test*		*LongStrings*	
	Mean	**Best**	**Mean**	**Best**	**n=120**	**n=160**
RNN	9.12	14.02	0.2	0.4	0.0	0.0
LSTM	54.00	62.80	1.40	4.00	0.2	0.0
GRU	48.00	50.00	1.00	1.02	0.2	0.0
Δ-*RNN*	52.00	60.02	1.50	4.00	0.0	0.0
Stack RNN	71	100	70	100	85	50
Stack-RNN+Softmax	98.99	100	99.18	100	85.00	75.10
Baby-NTM+Softmax	73.59	99.99	67.52	99.01	65.50	55.50
*DiffStk-RNN**(Ours)***	100	100	99.99	100	86.5	79.50
*DiffStk-LSTM**(Ours)***	94.20	100	90	100	83.20	71.00
*DiffStk-MRNN**(Ours)***	**100**	**100**	**100**	**100**	**92**	**90**
*DiffStk-MLSTM**(Ours)***	96	100	95.69	100	89.50	85.00
*DiffStk-MIRNN**(Ours)***	98	100	98.20	100	81.50	68.50

Table 1: Percentage of correctly classified strings of RNNs trained on the D_2 language (over 10 trials). We report the mean and best accuracy for each. The test set contained a mix of short & long samples of length up to $T = 102$ with both positive and negative samples. We report mean accuracy of models tested on longer strings ($n = 120$ & $n = 160$)

We report the mean and best performance accuracy for each model. All networks were optimized over the course of 30 epochs and we report the best training and testing score for each model. Percentage of correctly classified strings refer to accuracy of the model in successfully recognizing input strings as valid or invalid.

5.2. Classification on Longer Strings

If a model is operating in the same way as its equivalent pushdown automata, in theory, it should be able to recognize a string of any length (Chomsky and Schützenberger, 1959; Chomsky, 1962). This provides a way for analyzing the limitations of RNNs on longer strings. We created a separate test set containing 1500 string samples of length ($105 < T$ and $T > 160$). This sampling of lengths properly tests the limitations of RNNs, since interpretability is largely dependent on the fact that the RNN is acting like a PDA. This is crucial if we wish to extract a minimal PDA from the final trained model weights (Das et al., 1993; Sun et al., 1998; Weiss et al., 2018b; Omlin and Giles, 1992; Jacobsson, 2005).

6. Result and Discussion

In Table 1, all first order RNNs without memory appear to struggle to correctly recognize the D_2 grammar – even the LSTM reaches only a 4% accuracy. In contrast, the DiffStk-MRNN performed the best, which we hypothesize is due to its ability to approximate the higher order weights (which is similar to a tensor RNN). Similar performance was observed for grammars D_3, D_6, and the Palindrome in Tables 2, 3, and 4. However, for the grammar D_6 as well as for the palindrome grammar, we observe a constant drop in performance as the string length increases, which indicates that even stack-augmented RNNs have limitations.

	Train		*Test*		*LongStrings*	
RNN Models	**Mean**	**Best**	**Mean**	**Best**	**n=120**	**n=160**
RNN	9.60	15.02	0.0	0.0	0.0	0.0
LSTM	30.88	35.00	0.02	0.04	0.0	0.0
GRU	10.00	15.00	0.0	0.0	0.0	0.0
Δ-RNN	20.00	21.00	0.02	0.04	0.0	4.00
Stack RNN	80	100	69	100	67	60
Stack-RNN+Softmax	80.25	99.99	78.99	99.99	78.30	59.50
Baby-NTM+Softmax	59.50	99.99	43.50	99.21	40.50	51.80
*DiffStk-RNN**(Ours)***	82.60	100	82.00	100	78.20	60
*DiffStk-LSTM**(Ours)***	76.75	100	57.50	99.50	55	44.50
*DiffStk-MRNN**(Ours)***	**84.02**	**100**	**84.00**	**100**	**81**	**80**
*DiffStk-MLSTM**(Ours)***	77	100	59.00	100	59	49.99
*DiffStk-MIRNN**(Ours)***	78.60	99.50	83.00	97.50	73.00	58.50

Table 2: Percentage of correctly classified strings of various RNNs trained on the D_3 language (averaged over 10 trials). We report the mean and best accuracy for each model. Test set used in all experiments had a mix of short samples and long ones of length up to$T = 102$ containing both positive and negative samples. Further we report mean accuracy on models when tested on longer strings of length $n = 120$ and $n = 160$ respectively

	Train		*Test*		*LongStrings*	
RNN Models	**Mean**	**Best**	**Mean**	**Best**	**n=120**	**n=160**
RNN	22.60	25.00	0.0	0.0	0.0	0.0
LSTM	38.00	42.80	0.01	0.04	0.0	0.0
GRU	25.00	28.55	0.01	0.01	0.0	0.0
Δ-RNN	32.62	38.55	0.02	0.04	0.0	0.0
StackRNN	99.95	100	99.80	100	92	89
Stack-RNN+Softmax	98.99	100	99.08	99.58	94.50	89.50
Baby-NTM+Softmax	99.99	100	99.58	99.81	98.50	95.00
*DiffStk-RNN**(Ours)***	99.99	100	99.89	100	94	91
*DiffStk-LSTM**(Ours)***	99.80	100	98.00	100	90	89.20
*DiffStk-MRNN**(Ours)***	**100**	**100**	**100**	**100**	**100**	**99**
*DiffStk-MLSTM**(Ours)***	99.90	100	99.99	100	98	97.50
*DiffStk-MIRNN**(Ours)***	99.99	98	99.90	100	91	90.50

Table 3: Percentage of correctly classified strings of various RNNs trained on the D_6 language (averaged over 10 trials). We report the mean and best accuracy for each model. Test set used in all experiments had a mix of short samples and long ones of length up to$T = 102$ containing both positive and negative samples. Further we report mean accuracy on models when tested on longer strings of length $n = 120$ and $n = 160$ respectively

Based on our experiments and examination of the internal hidden state representations of the trained recognition models, we did find that whenever a test set contained more strings of longer length, the majority of RNNs struggle as does a similar model proposed in (Suzgun et al., 2019b). This seems to be largely due to the models not being exposed during training

	Train		*Test*		*LongStrings*	
RNN Models	**Mean**	**Best**	**Mean**	**Best**	**n=120**	**n=160**
RNN	0	0	0	0	0.0	0.0
LSTM	2.50	5.45	0	0	0.0	0.0
GRU	0.9	1.2	0.3	0.5	0.0	0.0
Δ-RNN	0.5	1.0	0.01	0.01	0.0	0.0
StackRNN	49.50	100	50	100	50	49
Stack-RNN+Softmax	61.35	100	59.50	99.99	55	53.50
Baby-NTM+Softmax	52.50	100	52.80	99.91	46.50	45.00
*DiffStk-RNN**(Ours)***	62.00	100	61.00	100	60.50	60.00
*DiffStk-LSTM**(Ours)***	63.50	100	62.20	100	62	60.50
*DiffStk-MRNN**(Ours)***	**64.00**	**100**	**63.00**	**100**	**62.50**	**62.00**
*DiffStk-MLSTM**(Ours)***	65.55	100	64.50	100	64.00	64.00
*DiffStk-MIRNN**(Ours)***	61.55	99.99	61.50	99.99	60.50	60

Table 4: Percentage of correctly classified strings of various RNNs trained on the Palindrome language (averaged over 10 trials). We report the mean and best accuracy for each model. Test set used in all experiments had a mix of short samples and long ones of length up to$T = 102$ containing both positive and negative samples. Further we report mean accuracy on models when tested on longer strings of length $n = 120$ and $n = 160$ respectively.

	Validation	*Test*
RNN Models	**PPL**	**PPL**
RNN	137	129
LSTM	120	**115**
SRCN	120	**115**
StackRNN	124	118
Stack-RNN+Softmax	128	123
Baby-NTM+Softmax	142	131
*DiffStk-RNN**(Ours)***	122	118
*DiffStk-MRNN**(Ours)***	**119**	**115**
*DiffStk-MIRNN**(Ours)***	121	117

Table 5: Word level perplexity of various models on Penn Treebank.

to strings of long enough lengths which is shown in tables 1, 2, 3 and 4. One way to deal with this and improve generalization would be to create a more complex validation set and optimize based on its performance on that sample. Furthermore, one should use negative sampling when working with complex grammars, since our results demonstrate that RNNs that train with "difficult" negative samples generalize better on longer strings. Finally, it is important to ensure that an RNN's weights do not change much when encountering a NO-OP (NoOP) operation. This avoids conflicts for whenever the weights of each stack operation are shifted by a small amount, since even a small degree of noise can alter the RNN's entire prediction.

We also compared our models by word level perplexity on the Penn Treebank data benchmark; results are shown in Table 5. It is important to note that this experiment is designed not to obtain a state of the art language model but rather to show how memory-augmented models, with similar parameters, can yield better performance and are also stable. We adopted the experiment protocol from (Joulin and Mikolov, 2015) and report validation and test word level perplexity (PPL). Our proposed DiffStk-MRNN model achieves best validation and test PPL and also matches the performance of LSTM on language modelling task. In Appendix B we show how individual components introduced in this work contributes towards enhanced model performance.

7. Conclusion

We introduce five improved stack-augmented recurrent neural network (RNN) models and evaluated their ability to recognize complex and long context free grammars (CFGs). We also develop various techniques to efficiently train differentiable, stack-augmented models. To our knowledge, this is the first work to analyze the performance of a continuous stack on long strings for complex grammars. We show that utilizing higher order weights improves model generalization. In addition, the value of memory structures is demonstrated and why important when learning non-regular languages. Since memory-augmented RNNs often suffer from stability issues while training, several efficient schemes are provided to facilitate more stable training. Interesting research would be to design a custom optimization process that couples a discrete stack with first order RNNs and achieve stable learning. Another direction would be to explore data structures other than stacks such as random access memory and to develop a scalable approach towards stably train these networks, especially on challenging real-world problems.

References

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages, 2020.

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

Rafael C Carrasco and Mikel L Forcada. Second-order recurrent neural networks can learn regular grammars from noisy strings. In *International Workshop on Artificial Neural Networks*, pages 605–610. Springer, 1995.

Rafael C Carrasco, Mikel L Forcada, M Angeles Valdés-Munoz, and Ramón P Ñeco. Stable encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units. *Neural Computation*, 12(9):2129–2174, 2000.

Noam Chomsky. Context-free grammars and pushdown storage. *MIT Res. Lab. Electron. Quart. Prog. Report.*, 65:187–194, 1962.

Noam Chomsky and David W Lightfoot. *Syntactic structures*. Walter de Gruyter, 2002.

Noam Chomsky and Marcel P Schützenberger. The algebraic theory of context-free languages. In *Studies in Logic and the Foundations of Mathematics*, volume 26, pages 118–161. Elsevier, 1959.

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

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.

Tristan Deleu and Joseph Dureau. Learning operations on a stack with neural turing machines. *CoRR*, abs/1612.00827, 2016. URL http://arxiv.org/abs/1612.00827.

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.

Mikel L Forcada and Rafael C Carrasco. Learning the initial state of a second-order recurrent neural network during regular-language inference. *Neural computation*, 7(5):923–930, 1995.

Paolo Frasconi and Marco Gori. Computational capabilities of local-feedback recurrent networks acting as finite-state machines. *IEEE Transactions on Neural Networks*, 7(6): 1521–1525, 1996.

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, Clifford B Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

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.

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

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649, 2013.

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, Dec 2020. ISSN 2307-387X. doi: 10.1162/tacl_a_00306. URL http://dx.doi.org/10.1162/tacl_a_00306.

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.

Bill G Horne and Don R Hush. Bounds on the complexity of recurrent neural network implementations of finite state machines. In *Advances in neural information processing systems*, pages 359–366, 1994.

Henrik Jacobsson. Rule extraction from recurrent neural networks: Ataxonomy and review. *Neural Computation*, 17(6):1223–1263, 2005.

Kam-Chuen Jim, C Lee Giles, and Bill G Horne. An analysis of noise in recurrent neural networks: convergence and generalization. *IEEE Transactions on neural networks*, 7(6): 1424–1438, 1996.

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

Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1700–1709, 2013.

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

Ryan Kiros, Ruslan Salakhutdinov, and Rich Zemel. Multimodal neural language models. In *International conference on machine learning*, pages 595–603, 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.

Stefan C Kremer. On the computational power of elman-style recurrent networks. *IEEE Transactions on neural networks*, 6(4):1000–1004, 1995.

Stefan C Kremer, Ramón P Ñeco, and Mikel L Forcada. Constrained second-order recurrent networks for finite-state automata induction. In *International Conference on Artificial Neural Networks*, pages 529–534. Springer, 1998.

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.

A. Mali, A. G. Ororbia, and C. L. Giles. The sibling neural estimator: Improving iterative image decoding with gradient communication. In *2020 Data Compression Conference (DCC)*, pages 23–32, 2020.

Ankur Mali, Alexander Ororbia, and C Lee Giles. The neural state pushdown automata. *arXiv preprint arXiv:1909.05233*, 2019.

William Merrill. Sequential neural networks as automata. *CoRR*, abs/1906.01615, 2019. URL http://arxiv.org/abs/1906.01615.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048, 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.

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.

Maurice Nivat. On some families of languages related to the dyck language. In *Proceedings of the second annual ACM symposium on Theory of computing*, pages 221–225, 1970.

Hyeonwoo Noh, Tackgeun You, Jonghwan Mun, and Bohyung Han. Regularizing deep neural networks by noise: Its interpretation and optimization. In *Advances in Neural Information Processing Systems*, pages 5109–5118, 2017.

Christian W Omlin and C Lee Giles. Training second-order recurrent neural networks using hints. In *Machine Learning Proceedings 1992*, pages 361–366. Elsevier, 1992.

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, 1996.

Alexander Ororbia, Ankur Mali, Matthew Kelly, and David Reitter. Like a baby: Visually situated neural language acquisition. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pages 5127–5136, 2019.

Alexander G Ororbia II, Tomas Mikolov, and David Reitter. Learning simpler language models with the differential state framework. *Neural computation*, 29(12):3327–3352, 2017.

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.

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

Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *J. Comput. Syst. Sci.*, 50(1):132–150, 1995.

G. Z. Sun, C. L. Giles, and H. H. Chen. *The neural network pushdown automaton: Architecture, dynamics and training*, pages 296–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-69752-7. doi: 10.1007/BFb0054003.

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.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, September 9-13, 2012*, pages 194–197, 2012.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.

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.

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

Raymond L Watrous and Gary M Kuhn. Induction of finite-state languages using second-order recurrent networks. *Neural Computation*, 4(3):406–414, 1992.

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.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers, 2021.

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.

Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Russ R Salakhutdinov. On multiplicative integration with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 2856–2864, 2016.

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.

Wojciech Zaremba, Tomas Mikolov, Armand Joulin, and Rob Fergus. Learning simple algorithms from examples. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 421–429, 2016.

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.

Tom Ziemke. Towards adaptive perception in autonomous robots using second-order recurrent networks. In *Proceedings of the First Euromicro Workshop on Advanced Mobile Robots (EUROBOT'96)*, pages 89–98. IEEE, 1996.

Appendix A. Variants of DiffStk RNNs

Here we describe different variants of the RNN and how they can be extended with the differentiable stacks introduced in this work.

A.1. DiffStk-LSTM Archiecture

The DiffStk-LSTM is similar to the DiffStk-RNN but it contains additional gating components to help it better preserve memory over long time spans. There are several variants of LSTM (Gers and Schmidhuber, 2001) and we describe the variant used in our experiments (which we found worked best from preliminary experimentation).The hidden state for our LSTM is computed as follows:

$$\hat{\mathbf{z}}_t = U \cdot \mathbf{x}_t + R \cdot \mathbf{z}_{t-1} \tag{11}$$

where U is a $d \times m$ matrix and R is a $m \times m$ matrix. LSTM's consist of 3 gating units – an input gate $\mathbf{i}$, an output gate $\mathbf{o}$, and a forget gate $\mathbf{f}$, which all have recurrent and feedforward connections themselves:

$$\mathbf{i}_t = \sigma(U_i \cdot \mathbf{x}_t + R_i \cdot \mathbf{z}_{t-1}) \tag{12}$$
$$\mathbf{o}_t = \sigma(U_o \cdot \mathbf{x}_t + R_o \cdot \mathbf{z}_{t-1}) \tag{13}$$
$$\mathbf{f}_t = \sigma(U_f \cdot \mathbf{x}_t + R_f \cdot \mathbf{z}_{t-1}). \tag{14}$$

Given the above gate definitions, the final internal state and output of the hidden state are calculated in the following manner:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f_1(\hat{\mathbf{z}}_t) \tag{15}$$
$$\mathbf{z}_t = f_1(\mathbf{c}_t) \odot \mathbf{o}_t \tag{16}$$

where $\odot$ indicates the Hadamard product. The ability of the LSTM to control how information is stored in each of its cells has proven to be useful in many applications.

A.2. DiffStk-MRNN

The multiplicative RNN (MRNN) (Krause et al., 2016) is similar to second order tensor RNNs (Giles et al., 1990; Watrous and Kuhn, 1992; Forcada and Carrasco, 1995; Ziemke, 1996; Kremer et al., 1998; Carrasco and Forcada, 1995). It uses a factorized hidden-to-hidden transition matrix in place of the normal RNN hidden-to-hidden matrix (R or W_{zz}). The MRNN can be formulated to compute an intermediate state $\mathbf{m}_t$ as:

$$\mathbf{m}_t = (W_{mx} \cdot \mathbf{x}_t) \cdot (W_{mz} \cdot \mathbf{z}_{t-1}) \tag{17}$$
$$\hat{\mathbf{z}}_t = W_{zm} \cdot \mathbf{m}_t + W_{zx} \cdot \mathbf{x}_t \tag{18}$$

with the final hidden state update calculated as follows:

$$\hat{\mathbf{z}}_t = f_1(U \cdot \mathbf{x}_t + \hat{R} \cdot \hat{\mathbf{z}}_{t-1}) \tag{19}$$

where $\hat{R}$ is a $m \times z$ matrix (z is the dimension of the intermediate state). The MRNN can be shown to approximate a NSPDA with second order weights (Mali et al., 2019; Das et al., 1993; Sun et al., 1998).

A.3. DiffStk-MLSTM

Motivated by the success of tensor RNNs on complex CFLs, we designed a multiplicative LSTM (MLSTM) as a hybrid model which combines a factorized hidden-to-hidden transition (which is an approximation of a second order RNN) with our differentiable stack. A MRNN's intermediate state ($\mathbf{m}_t$) can easily be connected to the LSTM gating units. This gives the model not only more expressive cells but also allows it to operate similarly to a pushdown automata. The resulting formulation of the model is as follows:

$$\mathbf{m}_t = (U_m \cdot \mathbf{x}_t) \odot (R_m \cdot \mathbf{z}_{t-1}) \tag{20}$$

$$\hat{\mathbf{z}}_t = U \cdot \mathbf{x}_t + R_z \cdot \mathbf{m}_t) \tag{21}$$

$$\mathbf{i}_t = \sigma(U_i \cdot \mathbf{x}_t + R_{im} \cdot \mathbf{m}_t) \tag{22}$$

$$\mathbf{o}_t = \sigma(U_o \cdot \mathbf{x}_t + R_{om} \cdot \mathbf{m}_t) \tag{23}$$

$$\mathbf{f}_t = \sigma(U_f \cdot \mathbf{x}_t + R_{fom} \cdot \mathbf{m}_t). \tag{24}$$

In all of our experiments, we set the dimensions of the state $\mathbf{z}$ and $\mathbf{m}$ to be similar. Integrating a stack with the MLSTM is similar to the formulation provided in the DiffStk-RNN section.

A.4. DiffStk-MIRNN

The multiplicative integration RNN (MIRNN) (Wu et al., 2016) is yet another approximation of higher order/tensor RNNs (Giles et al., 1990; Watrous and Kuhn, 1992; Forcada and Carrasco, 1995; Ziemke, 1996; Kremer et al., 1998; Carrasco and Forcada, 1995). This model uses a Hadamard product $\odot$ instead of a sum operation $+$ in the standard RNN update rule. Hence, this formulation allows it to update its hidden state without introducing any extra parameters. This change can be written in the following manner:

$$\mathbf{z}_t = f1(U \cdot \mathbf{x}_t \odot R \cdot \mathbf{z}_{t-1}). \tag{25}$$

These models have the capacity to retain memory over longer time spans and their approximate second order connections add extra expressiveness that proves useful when learning to recognize complex CFLs. Integration of a stack with the MI-RNN cell is also similar to the previous models discussed.

Appendix B. Ablation Study and Discussion

When testing on various CFGs, our experiments show the importance of external memory structures for RNNs. We observe that all stack-augmented RNNs at least once were able to achieve full accuracy on the smaller test set (n=102) and a few models achieved 90% accuracy on the larger test set(n $>$ 102). In Table 6, we show how carefully adding noise increases robustness of the model and helps in learning longer sequences. This form of noise is added to weights rather than to gradients and shows a consistent improvement for models that use our proposed form of noise. Beside noise, we also show the importance of skipping states whenever multiple no-ops stack operations are encountered for a given input symbol. This step is crucial since a small change in gradients can corrupt a model's prediction and

	WithNoise		*W/oNoise*	
RNN Models	**Mean**	**Best**	**Mean**	**Best**
RNN	0.2	0.4	0.0	0.0
LSTM	1.40	4.00	1.00	3.85
StackRNN	70	100	70	99.95
DiffStk-RNN (Ours)	99.99	100	97.58	99.00
DiffStk-MRNN (Ours)	**99.00**	**100**	97.20	99.99
DiffStk-MIRNN (Ours)	97.25	99	97.00	99

Table 6: Percentage of correctly classified strings of various RNNs, trained with and without noise, on D_2 language (averaged over 10 trials), with mean and best accuracy measurements reported for each model. For all experiments the test set used had short samples mixed with long ones up to length $T = 102$, containing both positive and negative sample strings.

	Wchanges		*W/ochanges*	
RNN Models	**Mean**	**Best**	**Mean**	**Best**
RNN	0.2	0.4	0.0	0.0
LSTM	1.40	4.00	1.00	3.85
StackRNN	70	100	70	99.95
DiffStk-RNN (Ours)	99.99	100	97.58	99.00
DiffStk-MRNN (Ours)	**100**	**100**	97.20	99.99
DiffStk-MIRNN (Ours)	98.20	100	97.00	99

Table 7: Percentage of correctly classified strings of various RNNs with and without noise as well as carry forwarding state on D_2 language (averaged over 10 trials). Mean and best accuracy are for each of the models. For all experiment The test set had short samples mixed with long ones up to length $T = 102$, with both positive and negative samples.

	Sequential		*Incremental*	
RNN Models	**Mean**	**Best**	**Mean**	**Best**
RNN	0.2	0.4	0.0	0.0
LSTM	1.40	4.00	1.00	3.85
StackRNN	70	100	70	99.95
DiffStk-RNN (Ours)	99.99	100	97.58	99.00
DiffStk-MRNN (Ours)	**100**	**100**	97.20	99.99
DiffStk-MIRNN (Ours)	98.20	100	97.00	99

Table 8: Percentage of correctly classified strings of various RNNs trained under incremental vs sequential schemes on D_2 language (measurements averaged over 10 trials). We report the mean and best accuracy for each model. For all experiment the test set used had short samples mixed with long ones up to length $T = 102$ containing both positive and negative sample strings.

significantly affect the mean squared error (MSE) loss value. It is well known that a small time shift in prediction can lead to a poor MSE. Therefore, we introduced a scheme which

carries forward states and preserves the current state whenever a stack encounters multiple NoOps for any input string. Table 7 demonstrates the effect of this extension.

In Table 8, we experimented with two popular training variants for our RNN models, sequential versus curriculum learning. We see in Table 8 that sequential learning provided a slight advantage over curriculum learning. Future work could focus on understanding the importance of these approaches under other constraints.

Continuous vs Discrete Stacks: Using a continuous or differentiable stack has been shown to benefit RNNs when recognizing patterns generated by CFGs (Giles et al., 1992; Joulin and Mikolov, 2015; Grefenstette et al., 2015). However, it is not entirely clear that continuous stack models are stable. Given that RNNs trained using BPTT are difficult to train and scale up, adding a differentiable memory introduces further instability during the optimization process, often resulting in sub-optimal performance. On the other hand, in theory, discrete stacks (Zeng et al., 1994; Mali et al., 2019), if trained or programmed correctly, are visually interpretable and are stable. Future work should focus on designing hybrid optimization approaches to build models that can utilize both discrete and differentiable units/stacks.