State-Regularized Recurrent Neural Networks

Cheng Wang [1] **Mathias Niepert** [1]

Abstract

Recurrent neural networks are a widely used class of neural architectures with two shortcomings. First, it is difficult to understand what exactly they learn. Second, they tend to work poorly on sequences requiring long-term memorization, despite having this capacity in principle. We aim to address both shortcomings with a class of recurrent networks that use a stochastic state transition mechanism between cell applications. This mechanism, which we term state-regularization, makes RNNs transition between a finite set of learnable states. We evaluate state-regularized RNNs on (1) regular languages for the purpose of automata extraction; (2) nonregular languages such as balanced parentheses, palindromes, and the copy task where external memory is required; and (3) real-word sequence learning tasks for sentiment analysis, visual object recognition, and language modeling. We show that state-regularization simplifies the extraction of finite state automata from the RNN's state transition dynamics; forces RNNs to operate more like automata with external memory and less like finite state machines; and makes RNNs more interpretable.

1. Introduction

Recurrent neural networks (RNNs) have found their way into numerous applications. Still, RNNs have two shortcomings. First, it is difficult to understand what concretely RNNs learn. Some applications require a close inspection of learned models before deployment and RNNs are more difficult to interpret than rule-based systems. There are a number of approaches for extracting finite state automata (DFAs) from trained RNNs (Giles et al., 1991; Wang et al., 2018b; Weiss et al., 2018b) as a means to analyze their behavior. These methods apply extraction algorithms after training and it remains challenging to determine whether the extracted DFA faithfully models the RNN's state transition behavior. Most extraction methods are rather complex, depend crucially on hyperparameter choices, and tend to be computationally costly. Second, RNNs tend to work poorly on input sequences requiring long-term memorization, despite having this ability in principle. Indeed, there is a growing body of work providing evidence, both empirically (Daniluk et al., 2017; Bai et al., 2018; Trinh et al., 2018) and theoretically (Arjovsky et al., 2016; Zilly et al., 2017; Miller & Hardt, 2018), that recurrent networks offer no benefit on longer sequences, at least under certain conditions. Intuitively, RNNs tend to operate more like DFAs with a large number of states, attempting to memorize all the information about the input sequence solely with their hidden states, and less like automata with external memory.

We propose state-regularized RNNs as a possible step towards addressing both of the aforementioned problems. State-regularized RNNs (SR-RNNs) are a class of recurrent networks that utilize a stochastic state transition mechanism between cell applications. The stochastic mechanism models a probabilistic state dynamics that lets the SR-RNNs transition between a finite number of learnable states. The parameters of the stochastic mechanism are trained jointly with the parameters of the base RNNs[1]. SR-RNNs have several advantages over standard RNNs. First, instead of having to apply post-training DFA extraction, SR-RNNs determine their (probabilistic and deterministic) state transition behavior more directly. We propose a method that extracts DFAs truly representing the state transition behavior of the underlying RNNs. Second, we hypothesize that the frequently-observed poor extrapolation behavior of RNNs is caused by memorization with hidden states. It is known that RNNs – even those with cell states or external memory – tend to memorize mainly with their hidden states and in an unstructured manner (Strobelt et al., 2016; Hao et al., 2018). We show that the state-regularization mechanism shifts representational power to memory components such as the cell state, resulting in improved extrapolation performance.

We support our hypotheses through experiments both on synthetic and real-world datasets. We explore the improvement

[1]NEC Laboratories Europe, Heidelberg, Germany. Correspondence to: Cheng Wang <cheng.wang@neclab.eu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

[1]An implementation of SR-RNNs is available at https://github.com/deepsemantic/sr-rnns.

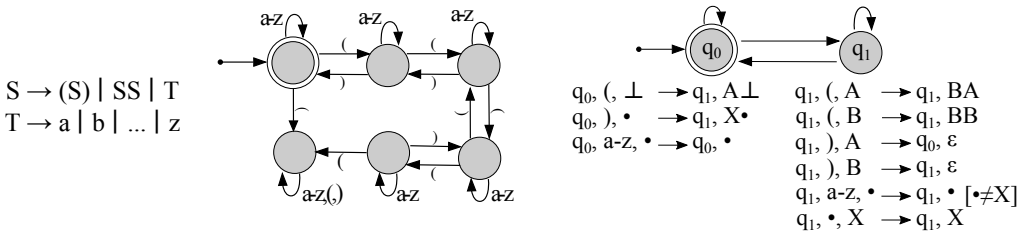


Figure 1. (Left) The context-free grammar for the language balanced parentheses (BP). (Center) A DFA that recognizes BP up to depth 4. (Right) A deterministic pushdown automaton (DPDA) that recognizes BP for all depths. The symbol • is a wildcard and stands for all possible tokens. The DPDA extrapolates to all sequences of BP, the DFA recognizes only those up to nesting depth 4.

of the extrapolation capabilities of SR-RNNs and closely investigate their memorization behavior. For state-regularized LSTMs, for instance, we observe that memorization can be shifted entirely from the hidden state to the cell state. For text and visual data, state-regularization provides more intuitive interpretations of the RNNs' behavior.

2. Background

We provide some background on deterministic finite automata (DFAs) and deterministic pushdown automata (DPDAs) for two reasons. First, one contribution of our work is a method for extracting DFAs from RNNs. Second, the state regularization we propose makes RNNs behave more like DPDAs and less like DFAs by limiting their ability to memorize with hidden states.

A DFA is a state machine that accepts or rejects sequences of tokens and produces one unique computation path for each input. Let Σ^* be the language over the alphabet Σ and let ϵ be the empty sequence. A DFA over an alphabet (set of tokens) Σ is a 5-tuple $(\mathcal{Q}, \Sigma, \delta, q_0, F)$ consisting of finite set of states $\mathcal{Q}$; a finite set of input tokens Σ called the input alphabet; a transition functions $\delta : \mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$; a start state q_0; and a set of accept states $F \subseteq \mathcal{Q}$. A sequence w is accepted by the DFA if the application of the transition function, starting with q_0, leads to an accepting state. Figure 1(center) depicts a DFA for the language of balanced parentheses (BP) up to depth 4. A language is regular if and only if it is accepted by a DFA.

A pushdown automata (PDA) is defined as a 7-tuple $(\mathcal{Q}, \Sigma, \Gamma, \delta, q_0, \perp, F)$ consisting of a finite set of states $\mathcal{Q}$; a finite set of input tokens Σ called the input alphabet; a finite set of tokens Γ called the stack alphabet, Γ^* is a set of strings over Γ; a transition function $\delta \subseteq \mathcal{Q} \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow \mathcal{Q} \times \Gamma^*$; a start state q_0; the initial stack symbol $\perp$; and a set of accepting states $F \subseteq \mathcal{Q}$. The computation starts in q_0 with the initial stack symbol $\perp$ on the stack and sequence w as input. The pushdown automaton accepts w if after reading w the automaton reaches an accepting state. Figure 1(right) depicts a deterministic PDA for the language BP.

3. State-Regularized Recurrent Networks

The standard recurrence of an RNN is $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$ where $\mathbf{h}_{t-1}$ is the hidden state vector at time $t-1$, and $\mathbf{h}_t$ and $\mathbf{x}_t$ are the hidden state and the input symbol at time t. We refer to RNNs whose unrolled cells are only connected through gated hidden states $\mathbf{h}$ as RNNs *without* ∞*-memory*. This is because values of gated hidden states $\mathbf{h}$ can only be in an interval such as $[-1, 1]$ for `tanh` and not $(-\infty, \infty)$. This limits, in this case, the information flow between cells to values between -1 and 1 and, therefore, memorization has to be performed with fractional changes. The family of GRUs (Chung et al., 2014) is without ∞-memory. LSTMs (Hochreiter & Schmidhuber, 1997), on the other hand, have ∞-memory due to their cell state.

A cell of a state-regularized RNN (SR-RNN) consist of two components. The first component, which we refer to as the *recurrent component*, applies the function of a standard RNN cell

$$\mathbf{u}_t = f(\mathbf{h}_{t-1}, \mathbf{c}_{t-1}, \mathbf{x}_t). \tag{1}$$

For the sake of completeness, we include the cell state $\mathbf{c}$ here, which is absent in RNNs without ∞-memory.

We propose a second component which we refer to as *stochastic component*. The stochastic component is responsible for modeling the probabilistic state transitions that let the RNN transition implicitly between a finite number of states. Let d be the size of the hidden state vectors of the recurrent cells. Moreover, let $\Delta^D := \{\boldsymbol{\lambda} \in \mathbb{R}_+^D : \| \boldsymbol{\lambda} \| = 1\}$ be the $(D-1)$ probability simplex. The stochastic component maintains k learnable centroids $\mathbf{s}_1$, ..., $\mathbf{s}_k$ of size d which we often write as the column vectors of a matrix $\mathbf{S} \in \mathbb{R}^{d \times k}$. The weights of these centroids are global parameters shared among all cells. The stochastic component computes, at each time step t, a discrete probability distribution from the output $\mathbf{u}_t$ of the recurrent component and the centroids of the stochastic component

$$\boldsymbol{\alpha} = g(\mathbf{S}, \mathbf{u}_t) \text{ with } \boldsymbol{\alpha} \in \Delta^k. \tag{2}$$

Crucially, instances of g should be differentiable to facilitate end-to-end training. Typical instances of the function g are

based on the dot-product, normalized into a probability distribution

$$\alpha_i = \frac{\exp\left((\mathbf{u}_t \cdot \mathbf{s}_i)/\tau\right)}{\sum_{i=1}^{k} \exp\left((\mathbf{u}_t \cdot \mathbf{s}_i)/\tau\right)} \tag{3}$$

Here, τ is a temperature parameter that can be used to anneal the probabilistic state transition behavior. The lower τ the more $\boldsymbol{\alpha}$ resembles the one-hot encoding of a centroid. The higher τ the more uniform is $\boldsymbol{\alpha}$. Equation 3 is reminiscent of the equations of attentive mechanisms (Bahdanau et al., 2014; Vaswani et al., 2017). Instead of attending to the hidden states, however, SR-RNNs attend to the k centroids to compute transition probabilities. Each α_i is the probability of the RNN to transition to centroid (state) i given the vector $\mathbf{u}_t$ for which we write $p_{\mathbf{u}_t}(i) = \alpha_i$.

The state transition dynamics of an SR-RNN is that of a probabilistic finite state machine. At each time step, when being in state $\mathbf{h}_{t-1}$ and reading input symbol $\mathbf{x}_t$, the probability for transitioning to state $\mathbf{s}_i$ is α_i. Hence, in its second phase the stochastic component computes the hidden state $\mathbf{h}_t$ at time step t from the distribution $\boldsymbol{\alpha}$ and the matrix $\mathbf{S}$ with a (possibly stochastic) mapping $h : \Delta^k \times \mathbb{R}^{d \times k} \rightarrow \mathbb{R}^d$. Hence, $\mathbf{h}_t = h(\boldsymbol{\alpha}, \mathbf{S})$. An instance of h is to

$$\text{sample } j \sim p_{\mathbf{u}_t} \text{ and set } \mathbf{h}_t = \mathbf{s}_j. \tag{4}$$

This renders the SR-RNN not end-to-end differentiable, however, and one has to use backpropagation approaches for discrete latent variable models such as REINFORCE (Williams, 1992) which are often less stable and less efficient. A possible alternative is to set the hidden state $\mathbf{h}_t$ to be the probabilistic mixture of the centroids

$$\mathbf{h}_t = \sum_{i=1}^{k} \alpha_i \mathbf{s}_i. \tag{5}$$

Every internal state $\mathbf{h}$ of the SR-RNN, therefore, is computed as a weighted sum $\mathbf{h} = \alpha_1 \mathbf{s}_1 + ... + \alpha_k \mathbf{s}_k$ of the centroids $\mathbf{s}_1, ..., \mathbf{s}_k$ with $\boldsymbol{\alpha} \in \Delta^k$. Here, h is a smoothed variant of the function that computes a hard assignment to one of the centroids. One can show that for $\tau \rightarrow 0$ the state dynamics based on equations (4) and (5) are identical and correspond to those of a DFA. Figure 2 depicts one of variants of the proposed SR-RNNs.

Additional instances of h are conceivable. For instance, one could, for every input sequence and the given current parameters of the SR-RNN, compute the most probable state sequence and then backpropagate based on a structured loss. Since finding these most probable sequences is possible with Viterbi type algorithms, one could apply a form of differentiable dynamic programming (Mensch & Blondel, 2018). The probabilistic state transitions of SR-RNNs open up new possibilities for applying more complex differentiable functions. We leave these considerations to future

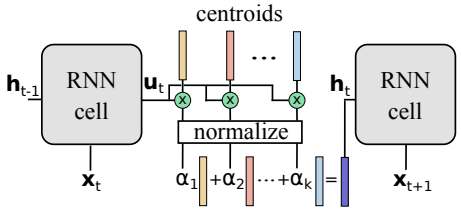


Figure 2. A possible instance of an SR-RNN corresponding to equations 3&5.

work. The probabilistic state transition mechanism is also applicable when RNNs have more than one hidden layer. In RNNs with $l > 1$ hidden layers, every such layer can maintain its own centroids and stochastic component. In this case, a global state of the SR-RNN is an l-tuple, with the lth argument of the tuple corresponding to the centroids of the lth layer.

Even though we have augmented the original RNN with additional learnable parameter vectors, we are actually constraining the SR-RNN to output hidden state vectors that are similar to the centroids. For lower temperatures and smaller values of k, the ability of the SR-RNN to memorize with its hidden states is increasingly impoverished. We argue that this behavior is beneficial for three reasons. First, it makes the extraction of interpretable DFAs from SR-RNNs without ∞-memory straight-forward. Instead of applying post-training DFA extraction as in previous work, we extract the true underlying DFA directly from the SR-RNN. Second, we hypothesize that overfitting in the context of RNNs is often caused by memorization via hidden states. Indeed, we show that regularizing the state space pushes representational power to memory components such as the cell state of an LSTM, resulting in improved extrapolation behavior. Third, the values of hidden states tend to increase in magnitude with the length of the input sequence, a behavior that has been termed *drifting* (Zeng et al., 1993). The proposed state regularization stabilizes the hidden states for longer sequences.

First, let us explore some of the theoretical properties of the proposed mechanism. We show that the addition of the stochastic component, when capturing the complete information flow between cells as, for instance, in the case of GRUs, makes the resulting RNN's state transition behavior identical to that of a probabilistic finite state machine.

Theorem 3.1. *The state transition behavior of a* SR-RNN *without ∞-memory using equation 4 is identical to that of a probabilistic finite automaton.*

Theorem 3.2. *For $\tau \rightarrow 0$ the state transition behavior of a* SR-RNN *without ∞-memory (using equations 4 or 5) is equivalent to that of a deterministic finite automaton.*

We can show that the lower the temperature the more SR-

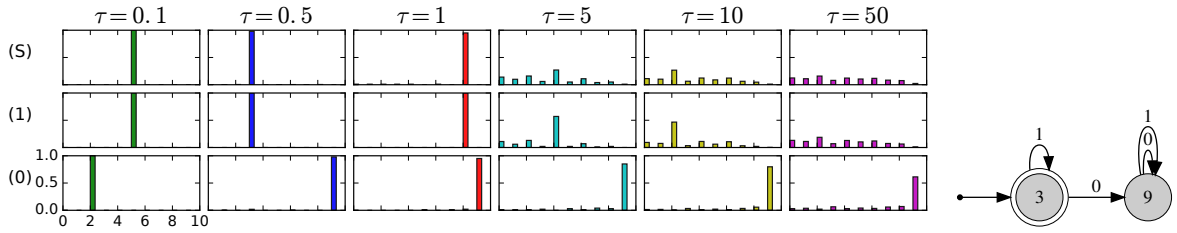


Figure 3. (Left) State transition probabilities for the SR-GRU learned from the data for the Tomita 1 grammar, for temperatures τ and input sequence $[10]$. S is the start token. Centroids are listed on x-axis, probabilities on y-axis. Up to temperature $\tau = 1$ the behavior of the trained SR-GRUs is almost identical to that of a DFA. Despite the availability of $k = 10$ centroids, the trained SR-GRUs use the minimal set of states for $\tau \leq 1$. (Right) The extracted DFA for Tomita grammar 1 and temperature $\tau = 0.5$.

RNN without ∞-memory operate like DFAs. The proofs of the theorems are in the Supplementary Material.

3.1. Learning DFAs with State-Regularized RNNs

Extracting DFAs from RNNs is motivated by applications where a thorough understanding of learned neural models is required before deployment. SR-RNNs maintain a set of learnable states and compute and explicitly follow state transition probabilities. It is possible, therefore, to extract finite-state transition functions that truly model the underlying state dynamics of the SR-RNN. The centroids do not have to be extracted from a clustering of a number of observed hidden states but can be read off of the trained model. This renders the extraction also more efficient. We adapt previous work (Schellhammer et al., 1998; Wang et al., 2018b) to construct the transition function of a SR-RNN. We begin with the start token of an input sequence, compute the transition probabilities $\boldsymbol{\alpha}$, and move the SR-RNN to the highest probability state. We continue this process until we have seen the last input token. By doing this, we get a count of transitions from every state $\mathbf{s}_i$ and input token $a \in \Sigma$ to the following states (including selfloops). After obtaining the transition counts, we keep only the most frequent transitions and discard all other transitions. Due to space constraints, the pseudo-code of the extraction algorithm is listed in the Supplementary Material. As a corollary of Theorem 3.2 we have that, for $\tau \to 0$, the extracted transition function is identical to the transition function of the DFA learned by the SR-RNN. Figure 3 shows that for a wide range of temperatures (including the standard softmax temperature $\tau = 1$) the transition behavior of a SR-GRU is identical to that of a DFA, a behavior we can show to be common when SR-RNNs are trained on regular languages.

3.2. Learning Nonregular Languages with State-Regularized LSTMs

For more complex languages such as context-free languages, RNNs that behave like DFAs generalize poorly to longer sequences. The DPDA shown in Figure 1, for instance, correctly recognizes the language of BP while the DFA only recognizes it up to nesting depth 4. We want to encourage RNNs with memory to behavor more like DPDAs and less like DFAs. The transition function δ of a DPDA takes (a) the current state, (b) the current top stack symbol, and (c) the current input symbol and maps these inputs to (1) a new state and (2) a replacement of the top stack symbol (see section 2). Hence, to allow an SR-RNN with memory such as the SR-LSTM to operate in a manner similar to a DPDA we need to give the RNNs access to these three inputs when deciding what to forget from and what to add to memory. Precisely this is accomplished for LSTMs with peephole connections (Gers & Schmidhuber, 2000). The following additions to the functions defining the forget, input, and output gates include the cell state into the LSTM's memory update decisions

$$\mathbf{f}_t = \sigma\big(\mathbf{W}^f\mathbf{x}_t + \mathbf{R}^f\mathbf{h}_{t-1} + \mathbf{p}^f \odot \mathbf{c}_{t-1} + \mathbf{b}^f\big) \tag{6}$$

$$\mathbf{i}_t = \sigma\big(\mathbf{W}^i\mathbf{x}_t + \mathbf{R}^i\mathbf{h}_{t-1} + \mathbf{p}^i \odot \mathbf{c}_{t-1} + \mathbf{b}^i\big) \tag{7}$$

$$\mathbf{o}_t = \sigma\big(\mathbf{W}^o\mathbf{x}_t + \mathbf{R}^o\mathbf{h}_{t-1} + \mathbf{p}^o \odot \mathbf{c}_t + \mathbf{b}^o\big) \tag{8}$$

Here, $\mathbf{h}_{t-1}$ is the output of the previous cell's stochastic component; $\mathbf{W}$s and $\mathbf{R}$s are the matrices of the original LSTM; the $\mathbf{p}$s are the parameters of the peephole connections; and $\odot$ is the elementwise multiplication. We show empirically that the resulting SR-LSTM-P operates like a DPDA, incorporating the current cell state when making decisions about changes to the next cell state.

3.3. Practical Considerations

Implementing SR-RNNs only requires extending existing RNN cells with a stochastic component. We have found the use of start and end tokens to be beneficial. The start token is used to transition the SR-RNN to a centroid representing the start state which then does not have to be fixed a priori. The end token is used to perform one more cell application but without applying the stochastic component before a classification layer. We find that a temperature of $\tau = 1$ (standard softmax) and an initialization of the centroids with values sampled uniformly from $[-0.5, 0.5]$ work well across different datasets.

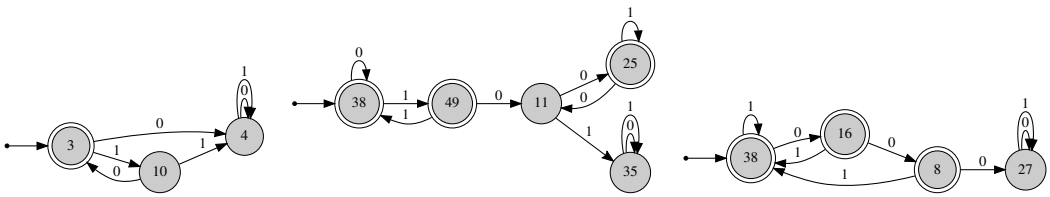


Figure 4. Extracted DFAs of the Tomita grammars 2-4. The state numbers correspond to the index of the learned SR-GRU centroids.

Dataset	Large Dataset				Small Dataset			
Models	LSTM	LSTM-P	SR-LSTM	SR-LSTM-P	LSTM	LSTM-P	SR-LSTM	SR-LSTM-P
$d \in [1, 10], l \leq 100$	0.005	0.022	0.038	**0.000**	0.068	0.031	0.037	**0.017**
$d \in [10, 20], l \leq 100$	0.334	0.308	0.255	**0.001**	0.472	0.407	0.347	**0.189**
$d \in [10, 20], l \leq 200$	0.341	0.313	0.351	**0.003**	0.479	0.440	0.352	**0.196**
$d = 5, l \leq 200$	0.002	0.044	0.013	**0.000**	0.042	0.021	0.028	**0.015**
$d = 10, l \leq 200$	0.207	0.227	0.254	**0.004**	0.409	0.362	0.279	**0.138**
$d = 20, l \leq 1000$	0.543	0.540	0.496	**0.020**	0.519	0.507	0.508	**0.380**

Table 1. Error rates for the balanced parentheses (BP) test sets (d=depth, l=length, k=5 centroids, the training depth ≤ 5).

4. Experiments

We conduct three types of experiments to investigate our hypotheses. First, we apply a simple algorithm for extracting DFAs and assess to what extent the true DFAs can be recovered from input data. Second, we compare the behavior of LSTMs and state-regularized LSTM on nonregular languages such as the languages of balanced parentheses and palindromes. Third, we investigate the performance of state-regularized LSTMs on non-synthetic datasets.

We implemented SR-RNNs with Theano (Theano Development Team, 2016). All experiments were conducted on a single Titan Xp with 12G memory. Unless otherwise indicated we always (a) use single-layer RNNs, (b) learn an embedding for input tokens before feeding it to the RNNs, (c) apply ADADELTA (Zeiler, 2012) for regular language and RMSPROP (Tieleman & Hinton, 2012) with a learning rate of 0.01 and momentum of 0.9 for the rest; (d) do not use dropout (Srivastava et al., 2014) or batch normalization (Cooijmans et al., 2017) of any kind; and (e) use state-regularized RNNs based on equations 3&5 with a temperature of $\tau = 1$ (standard softmax).

4.1. Regular Languages and DFA Extraction

We evaluate the DFA extraction algorithm for SR-RNNs on RNNs trained on the Tomita grammars (Tomita, 1982) which have been used as benchmarks in previous work (Wang et al., 2018b; Weiss et al., 2018b). We use available code (Weiss et al., 2018b) to generate training and test data for the regular languages. We first trained a single-layer GRU with 100 units on the data. We use GRUs since they do not have cells states and, hence, Theorem 3.2 applies. Whenever the GRU converged within 1 hour to a training accuracy of 100%, we also trained a SR-GRU based on equations 3&5 with $k = 50$ and $\tau = 1$. This was the case for the grammars 1-4 and 7. The difference in time to convergence between the vanilla GRU and the SR-GRU was negligible. We applied the transition function extraction outlined in section 3.1. In all cases, we could recover the minimal and correct DFA corresponding to the grammars. Figure 4 depicts the DFAs for grammars 2-4 extracted by our approach. Remarkably, even though we provide more centroids (possible states; here $k = 50$) the SR-GRU only utilizes the required minimal number of states for each of the grammars. All extracted DFAs are equivalent to the languages. Figure 3 visualizes the transition probabilities for different temperatures and $k = 10$ for grammar 1. The numbers on the states correspond directly to the centroid numbers of the learned SR-GRU. One can observe that the probabilities are spiky, causing the SR-GRU to behave like a DFA for $\tau \leq 1$.

4.2. Nonregular Languages

For nonregular languages external memorization is required. We investigate whether our hypothesis that SR-LSTM behave more like DPDAs and, therefore, extrapolate to longer sequences, is correct. To that end, we use the context-free language "balanced parentheses" (BP; see Figure 1(left)) over the alphabet $\Sigma = \{a, ..., z, (,)\}$, used in previous work (Weiss et al., 2018b). We created two datasets for BP. A large one with 22,286 training sequences (positive: 13,025; negative: 9,261) and 6,704 validation sequences (positive: 3,582; negative: 3,122). The small dataset consists of 1,008 training sequences (positive: 601; negative: 407), and 268 validation sequences (positive: 142; negative: 126). The training sequences have nesting depths

Number of centroids	$k=2$	$k=5$	$k=10$	$k=50$
$d \in [1,10], l \leq 100$	0.019	**0.017**	0.021	0.034
$d \in [10,20], l \leq 100$	**0.096**	0.189	0.205	0.192
$d \in [10,20], l \leq 200$	**0.097**	0.196	0.213	0.191
$d = 5, l \leq 200$	0.014	0.015	**0.012**	0.047
$d = 10, l \leq 200$	**0.038**	0.138	0.154	0.128
$d = 20, l \leq 1000$	0.399	**0.380**	0.432	0.410

Table 2. Error rates of the SR-LSTM-P on the small BP test data for various numbers of centroids k (d=depth, l=length).

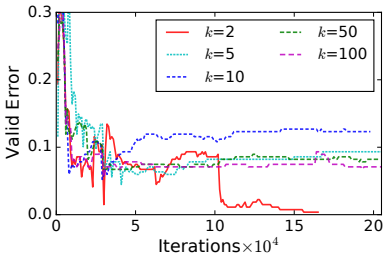
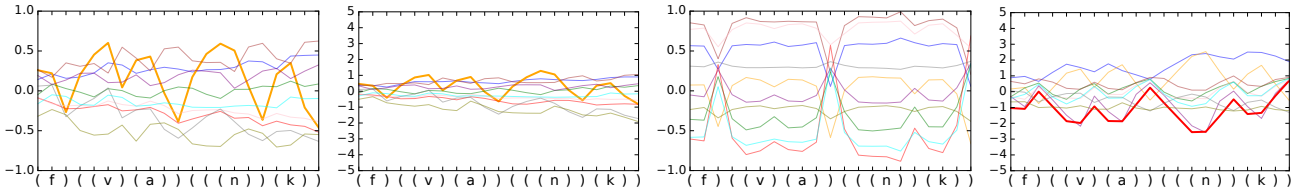


Figure 5. SR-LSTM-P error curves on the small BP validation data.



(a) $\mathbf{h}_t$ of the LSTM (b) $\mathbf{c}_t$ of the LSTM (c) $\mathbf{h}_t$ of the SR-LSTM-P (d) $\mathbf{c}_t$ of the SR-LSTM-P

Figure 6. Visualization of hidden state $\mathbf{h}_t$ and cell state $\mathbf{c}_t$ of the LSTM and the SR-LSTM-P for a specific input sequence from BP. Each color corresponds to one of 10 hidden units. The LSTM memorizes the number of open parentheses both in the hidden and to a lesser extent in the cell state (bold orange lines). The memorization is not accomplished with saturated gate outputs and a drift is observable for both vectors. The SR-LSTM-P maintains two distinct hidden states (accept and reject) and does not visibly memorize counts through its hidden states. The cell state is used to cleanly memorize the number of open parentheses (bold red line) with saturated gate outputs (± 1). For SR-LSTM-P, a state vector drift is not observable; solutions with less drift generalize better (Gers & Schmidhuber, 2001).

$d \in [1,5]$ and the validation sequences $d \in [6,10]$. We train the LSTM and the SR-RNNs using curriculum learning as in previous work (Zaremba & Sutskever, 2014; Weiss et al., 2018b) and using the validation error as stopping criterion. We then apply the trained models to unseen sequences. Table 1 lists the results on 1,000 test sequences each with the respective depths and lengths. The results show that both SR-LSTM and SR-LSTM-Ps extrapolate better to longer sequences and sequences with deeper nesting. Moreover, the SR-LSTM-P performs almost perfectly on the large data indicating that peephole connections are indeed beneficial.

To explore the effect of the hyperparameter k, that is, the number of centroids of the SR-RNNs, we run experiments on the small BP dataset varying k and keeping everything else the same. Table 2 lists the error rates and Figure 5 the error curves on the validation data for the SR-LSTM-P and different values of k. Two centroids ($k = 2$) result in the best error rates for most sequence types. However, while k has an impact on the validation performance, it is relatively small. A close inspection of the transition probabilities reveals that the SR-LSTM-P mostly utilizes two states, independent of the value of k. These two states are used as accept and reject states. This also explains the superior results for $k = 2$ for this task. The results show that SR-RNNs utilize a minimal set of states similar to DPDAs.

A major hypothesis of ours is that the state-regularization encourages RNNs to operate more like DPDAs. To explore this hypothesis, we trained an SR-LSTM-P with 10 units on the BP data and visualized both the hidden state $\mathbf{h}_t$ and the cell state $\mathbf{c}_t$ for various input sequences. Similar state visualizations have been used in previous work (Strobelt et al., 2016; Weiss et al., 2018a). Figure 6 plots the hidden and cell states for a specific input, where each color corresponds to a dimension in the respective state vectors. Note the different scale of the y-axes of plots a,c and b,d, respectively. As hypothesized, the LSTM relies primarily on its hidden states for memorization. The SR-LSTM-P, on the other hand, does not use its hidden states for memorization. Instead it utilizes two main states (accept and reject) and memorizes the nesting depth cleanly in the cell state. Memorization is accomplished with saturated gate-outputs, incrementing or decrementing the respective cell states by a value of 1. The visualization also shows a drifting behavior for the LSTM, in line with observations made for first-generation RNNs (Zeng et al., 1993). The values for the various hidden states increase and diverge relative to each other over time. Drifting is not observable for the SR-LSTM-P.

We also experiment with the nonregular language ww^{-1} (Palindromes) (Schmidhuber et al., 2002) over the alphabet $\Sigma = \{a, ..., z\}$. We follow the same setup as for BP and include the details in the Supplementary Material. The results of Table 3 add evidence for the improved generalization behavior of state-regularized LSTMs over vanilla LSTMs (with peepholes). Additional experimental results on the Copying Memory task are provided in the supplement.

Max Length	100	200	500
LSTM	31.2	42.0	47.7
LSTM-P	28.4	36.2	41.5
SR-LSTM	28.0	36.0	44.6
SR-LSTM-P	**10.5**	**16.7**	**29.8**

Table 3. Error rates in % on sequences of varying lengths from the Palindrome test set.

Model	LSTM		SR-LSTM		LSTM-P		SR-LSTM-P	
Length	100	200	100	200	100	200	100	200
train error	29.23	32.77	27.17	31.72	25.50	29.05	**23.30**	**24.67**
test error	29.96	33.19	28.82	32.88	28.02	30.12	**26.04**	**26.21**
time (epoch)	400.29s		429.15s		410.29s		466.10s	

Table 4. Error rates (on training and test splits of the IMDB data) in % and averaged training time in seconds when training only on truncated sequences of length 10.

Methods	Error
Full+unlabelled+BoW (Maas et al., 2011)†	11.1
LM-LSTM+unlabelled (Dai & Le, 2015)†	7.6
SA-LSTM+unlabelled (Dai & Le, 2015)†	7.2
seq2-bown-CNN (Johnson & Zhang, 2015)	7.67
WRRBM+BoW(bnc) (Dahl et al., 2012)	10.8
LSTM-Jump (Yu et al., 2017)	10.6
Skip RNN (Campos et al., 2018)	13.4
LSTM	10.1
LSTM-P	10.3
SR-LSTM ($k = 10$)	9.4
SR-LSTM-P ($k = 10$)	**9.2**
SR-LSTM-P ($k = 50$)	9.8

Table 5. Test error rates (%) on IMDB. † uses unlabeled data.

Methods	Error
IRNN (Le et al., 2015)	3.0
URNN (Arjovsky et al., 2016)	4.9
Full URNN (Wisdom et al., 2016)	2.5
sTANH-RNN (Zhang et al., 2016)	1.9
Skip LSTM (Campos et al., 2018)	2.7
r-LSTM Full BP (Trinh et al., 2018)	1.6
BN-LSTM (Cooijmans et al., 2017)	1.0
Dilated GRU (Chang et al., 2017)	**0.8**
LSTM	2.3
LSTM-P	1.5
SR-LSTM ($k = 100$)	1.4
SR-LSTM-P ($k = 100$)	**0.8**
SR-LSTM-P ($k = 50$)	1.4

Table 7. Test error rates (%) on sequential MNIST.

4.3. Sentiment Analysis and Pixel-by-Pixel MNIST

We evaluated state-regularized LSTMs on the IMDB review dataset (Maas et al., 2011). It consists of 100k movie reviews (25k training, 25k test, and 50k unlabeled). We used only the labeled training and test reviews. Each review is labeled as *positive* or *negative*. To investigate the models' extrapolation capabilities, we train all models on truncated sequences of up to length 10 but test on longer sequences (length 100 and 200). The results listed in Table 4 show the improvement in extrapolation performance of the SR-LSTM-P. The table also lists the training time per epoch, indicating that the overhead compared to the standard LSTM (or with peepholes LSTM-P) is modest. Table 5 lists the results when training without truncating sequences. The SR-LSTM-P is competitive with state of the art methods that also do not use the unlabeled data.

We also explored the impact of state-regularization on pixel-by-pixel MNIST (LeCun et al., 1998; Le et al., 2015). Here, the 784 pixels of MNIST images are fed to RNNs one by one for classification. This requires the ability to memorize long-term dependencies. Table 7 shows the results. The classification function has the final hidden and cell state as input. Our (state-regularized) LSTMs do not use dropout, batch normalization, sophisticated weight-initialization, and are based on a simple single-layer LSTM. We can observe that SR-LSTM-Ps achieve competitive results, outperforming the vanilla LSTM and LSTM-P. We also conducted additional experiments with state-regularization on vanilla RNNs and GRUs (with the same number of hidden units as SR-LSTM). We achieve 31.9 and 13.6 test error, respectively, for SR-RNNs and SR-GRUs, which is worse than the results for the SR-LSTM-Ps. On MINST both networks failed to converge with same number of training epochs. This suggests the importance of a cell state and ∞-memory which we regularized to be used in a more structured manner. The language modeling experiments are provided in the Supplementary Material.

4.4. Interpretability and Explainability

State-regularization provides new ways to interpret the working of RNNs. Since SR-RNNs have a finite set of states, we can use the observed transition probabilities to visualize their behavior. For instance, to generate prototypes for the SR-RNNs we can select, for each state i, the input tokens that have the highest average probability leading to state i. For the IMDB reviews, these are the top probability words leading to each of the states. For pixel-by-pixel MNIST, these are the top probabilities of each pixel leading to the states. Table 8 lists, for each state (centroid), the word with the top transition probabilities leading to this state. Figure 7 shows the prototypes associated with digits "3" of MNIST and "T-shirt" of Fashion-MNIST. In contrast to previous methods (Berkes & Wiskott, 2006; Nguyen et al., 2016), the prototypes are directly generated with the centroids of the trained SR-RNNs and the input token embeddings and hidden states do not have to be similar. This separates the input representations and the mechanism of the network dynamics when generating prototypes. We refer the interested reader to the supplement for additional visualizations.

cent.	words with top-4 highest transition probabilities
1	but (0.97) hadn (0.905) college (0.87) even (0.853)
2	not (0.997) or (0.997) italian (0.995) never (0.993)
3	loved (0.998) definitely (0.996) 8 (0.993) realistic (0.992)
4	no (1.0) worst (1.0) terrible (1.0) poorly (1.0)

Table 8. The learned centroids and their prototypical words with the top-4 highest transition probabilities. This interprets the SR-LSTM-P model with centroids. The 3^{rd} (4^{th}) centroid is "positive" ("negative").

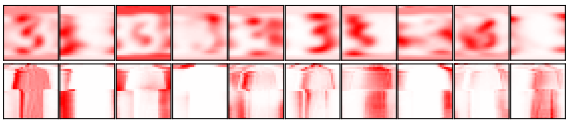


Figure 7. The prototypes of generated by learned SR-LSTM-P (k=10) centroids for "3" in MNIST (top) and "T-shirt" in Fashion-MNIST (bottom) (Xiao et al., 2017).

5. Related Work

RNNs are powerful learning machines. Siegelmann and Sontag (Siegelmann & Sontag, 1992; 1994; Siegelmann, 2012) proved that a variant of Elman-RNNs (Elman, 1990) can simulate a Turing machine. Recent work considers the more practical situation where RNNs have finite precision and linear computation time (Weiss et al., 2018a).

Extracting DFAs from RNNs goes back to work on first-generation RNNs in the 1990s (Giles et al., 1991; Zeng et al., 1993). These methods perform a clustering of hidden states after the RNNs are trained (Wang et al., 2018a; Frasconi & Bengio, 1994; Giles et al., 1991). Recent work introduced more sophisticated learning approaches to extract DFAs from LSTMs and GRUs (Weiss et al., 2018b). The latter methods tend to be more successful in finding DFAs behaving similar to the RNN. Differently, SR-RNNs learn an explicit set of states which facilitates the extraction of DFAs exactly modeling their state transition dynamics. A different line of work is motivated by more interpretable models (Koh & Liang, 2017; Doshi-Velez & Kim, 2017), attempting to learn more interpretable RNNs (Foerster et al., 2017), or extracting rule-based classifiers from RNNs (Murdoch & Szlam, 2017).

There is a large body of work on regularization techniques for RNNs. Most of these adapt regularization approaches developed for feed-forward networks to the recurrent setting. Representative instances are dropout regularization (Zaremba et al., 2014), variational dropout (Gal & Ghahramani, 2016), weight-dropped LSTMs (Merity et al., 2018), noise injection (Dieng et al., 2018) and independently RNNs (Li et al., 2018). Two approaches that can improve convergence and generalization capabilities are batch normalization (Cooijmans et al., 2017) and weight initialization strategies (Le et al., 2015) for RNNs. The work similar to SR-RNNs are self-clustering RNNs (Zeng et al., 1993) and clustering hidden states with Gaussian mixture models (Das & Mozer, 1994). Both approaches learn *discretized* states and show that these networks generalize better to longer input sequences. Contrary to those work, SR-RNNs incorporates self-attention to facilitate an end-to-end differentiable probabilistic state transition mechanism between cell applications.

Stochastic RNNs are a class of generative recurrent models for sequence data (Bayer & Osendorfer, 2014; Fraccaro et al., 2016; Goyal et al., 2017). Contrary to SR-RNNs, stochastic RNNs do not model probabilistic state transition dynamics. Hence, they do not address the problem of overfitting through hidden state memorization and improvements to DFA extraction.

There are proposals for extending RNNs with various types of external memory. Representative examples are the neural Turing machine (Graves et al., 2014), improvements thereof (Graves et al., 2016), memory network (Weston et al., 2015), associative LSTM (Danihelka et al., 2016), and RNNs augmented with neural stacks, queues, and deques (Grefenstette et al., 2015). Contrary to these proposals, we do not augment RNNs with differentiable data structures but regularize RNNs to make better use of existing memory components such as the cell state.

6. Discussion and Conclusion

State-regularization provides new mechanisms for understanding the workings of RNNs. Inspired by recent DFA extraction work (Weiss et al., 2018b), our work simplifies the extraction approach by directly learning a finite set of states and an interpretable state transition dynamics. Even on realistic tasks such as sentiment analysis, exploring the learned centroids and the transition behavior of SR-RNNs makes for more interpretable RNN models without sacrificing accuracy: a single-layer SR-RNNs is competitive with state-of-the-art methods. The purpose of our work is not to surpass all existing state of the art methods but to gain a deeper understanding of the dynamics of RNNs.

State-regularized RNNs operate more like automata with external memory and less like DFAs. This results in a markedly improved extrapolation behavior on several datasets. We do not claim, however, that SR-RNNs are a panacea for all problems associated with RNNs. For instance, we could not observe an improved convergence of SR-RNNs. Sometimes SR-RNNs converged faster, sometimes vanilla RNNs. While we have mentioned that the computational overhead of SR-RNNs is modest, it still exists, and this might exacerbate the problem that RNNs often take a long to be trained and tuned. We plan to investigate variants of state regularization and the ways in which it could improve differentiable computers with RNN controllers.

Acknowledgments

The authors wish to thank all the anonymous reviewers for their insightful comments and suggestions.

References

Arjovsky, M., Shah, A., and Bengio, Y. Unitary evolution recurrent neural networks. In *ICML*, pp. 1120–1128, 2016.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Bai, S., Kolter, J. Z., and Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Bayer, J. and Osendorfer, C. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

Berkes, P. and Wiskott, L. On the analysis and interpretation of inhomogeneous quadratic forms as receptive fields. *Neural computation*, 18(8):1868–1895, 2006.

Campos, V., Jou, B., Giró-i Nieto, X., Torres, J., and Chang, S.-F. Skip rnn: Learning to skip state updates in recurrent neural networks. *ICLR*, 2018.

Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., Cui, X., Witbrock, M., Hasegawa-Johnson, M. A., and Huang, T. S. Dilated recurrent neural networks. In *NIPS*, pp. 77–87, 2017.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv e-prints*, abs/1412.3555, 2014.

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., and Courville, A. Recurrent batch normalization. *ICLR*, 2017.

Dahl, G. E., Adams, R. P., and Larochelle, H. Training restricted boltzmann machines on word observations. *ICML*, 2012.

Dai, A. M. and Le, Q. V. Semi-supervised sequence learning. In *NIPS*, pp. 3079–3087, 2015.

Danihelka, I., Wayne, G., Uria, B., Kalchbrenner, N., and Graves, A. Associative long short-term memory. In *ICML*, pp. 1986–1994, 2016.

Daniluk, M., Rocktäschel, T., Welbl, J., and Riedel, S. Frustratingly short attention spans in neural language modeling. 2017.

Das, S. and Mozer, M. C. A unified gradient-descent/clustering architecture for finite state machine induction. In *NIPS*, pp. 19–26, 1994.

Dieng, A. B., Ranganath, R., Altosaar, J., and Blei, D. M. Noisin: Unbiased regularization for recurrent neural networks. *ICML*, 2018.

Doshi-Velez, F. and Kim, B. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

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

Foerster, J. N., Gilmer, J., Sohl-Dickstein, J., Chorowski, J., and Sussillo, D. Input switched affine networks: An rnn architecture designed for interpretability. In *ICML*, pp. 1136–1145, 2017.

Fraccaro, M., Sø nderby, S. r. K., Paquet, U., and Winther, O. Sequential neural models with stochastic layers. In *NIPS*, pp. 2199–2207. 2016.

Frasconi, P. and Bengio, Y. An em approach to grammatical inference: input/output hmms. In *ICPR*, pp. 289–294. IEEE, 1994.

Gal, Y. and Ghahramani, Z. A theoretically grounded application of dropout in recurrent neural networks. In *NIPS*, pp. 1019–1027, 2016.

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

Gers, F. A. and Schmidhuber, J. Recurrent nets that time and count. In *IJCNN (3)*, pp. 189–194, 2000.

Giles, C. L., Chen, D., Miller, C., Chen, H., Sun, G., and Lee, Y. Second-order recurrent neural networks for grammatical inference. In *IJCNN*, volume 2, pp. 273–281. IEEE, 1991.

Goyal, A., Sordoni, A., Côté, M.-A., Ke, N., and Bengio, Y. Z-forcing: Training stochastic recurrent networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *NIPS*, pp. 6713–6723. 2017.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwinska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu,

K., and Hassabis, D. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538 (7626):471–476, 2016.

Grefenstette, E., Hermann, K. M., Suleyman, M., and Blunsom, P. Learning to transduce with unbounded memory. In *NIPS*, pp. 1828–1836, 2015.

Hao, Y., Merrill, W., Angluin, D., Frank, R., Amsel, N., Benz, A., and Mendelsohn, S. Context-free transductions with neural stacks. In *Proceedings of the Analyzing and Interpreting Neural Networks for NLP workshop at EMNLP*, 2018.

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

Johnson, R. and Zhang, T. Effective use of word order for text categorization with convolutional neural networks. *NAACL HLT*, 2015.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *ICML*, pp. 1885–1894, 2017.

Le, Q. V., Jaitly, N., and Hinton, G. E. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, S., Li, W., Cook, C., Zhu, C., and Gao, Y. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *CVPR*, pp. 5457–5466, 2018.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *ACL-HLT*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

Mensch, A. and Blondel, M. Differentiable dynamic programming for structured prediction and attention. In *ICML*, volume 80, pp. 3462–3471, 2018.

Merity, S., Keskar, N. S., and Socher, R. Regularizing and optimizing lstm language models. *ICLR*, 2018.

Miller, J. and Hardt, M. When recurrent models don't need to be recurrent. *CoRR*, abs/1805.10369, 2018.

Murdoch, W. J. and Szlam, A. Automatic rule extraction from long short term memory networks. In *ICLR*, 2017.

Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T., and Clune, J. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *NIPS*, pp. 3387–3395, 2016.

Schellhammer, I., Diederich, J., Towsey, M., and Brugman, C. Knowledge extraction and recurrent neural networks: An analysis of an elman network trained on a natural language learning task. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, pp. 73–78, 1998.

Schmidhuber, J., Gers, F., and Eck, D. Learning nonregular languages: A comparison of simple recurrent networks and lstm. *Neural Computation*, 14(9):2039–2041, 2002.

Siegelmann, H. T. *Neural networks and analog computation: beyond the Turing limit*. Springer Science & Business Media, 2012.

Siegelmann, H. T. and Sontag, E. D. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 440–449. ACM, 1992.

Siegelmann, H. T. and Sontag, E. D. Analog computation via neural networks. *Theoretical Computer Science*, 131 (2):331–360, 1994.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Strobelt, H., Gehrmann, S., Huber, B., Pfister, H., Rush, A. M., et al. Visual analysis of hidden state dynamics in recurrent neural networks. *CoRR, abs/1606.07461*, 2016.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, 2016.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2):26–31, 2012.

Tomita, M. Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, pp. 105–108, 1982.

Trinh, T. H., Dai, A. M., Luong, T., and Le, Q. V. Learning longer-term dependencies in rnns with auxiliary losses. *ICML*, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *NIPS*, pp. 5998–6008. 2017.

Wang, Q., Zhang, K., Ororbia, I., Alexander, G., Xing, X., Liu, X., and Giles, C. L. A comparison of rule extraction for different recurrent neural network models and grammatical complexity. *arXiv preprint arXiv:1801.05420*, 2018a.

Wang, Q., Zhang, K., Ororbia II, A. G., Xing, X., Liu, X., and Giles, C. L. An empirical evaluation of rule extraction from recurrent neural networks. *Neural Computation*, 30 (9):2568–2591, 2018b.

Weiss, G., Goldberg, Y., and Yahav, E. On the practical computational power of finite precision rnns for language recognition. In *ACL (Volume 2: Short Papers)*, pp. 740–745, 2018a.

Weiss, G., Goldberg, Y., and Yahav, E. Extracting automata from recurrent neural networks using queries and counterexamples. In *ICML*, volume 80, pp. 5247–5256, 2018b.

Weston, J., Chopra, S., and Bordes, A. Memory networks. In *ICLR*, 2015.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Wisdom, S., Powers, T., Hershey, J., Le Roux, J., and Atlas, L. Full-capacity unitary recurrent neural networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *NIPS*, pp. 4880–4888. 2016.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yu, A. W., Lee, H., and Le, Q. V. Learning to skim text. *ACL*, 2017.

Zaremba, W. and Sutskever, I. Learning to execute. *CoRR*, abs/1410.4615, 2014.

Zaremba, W., Sutskever, I., and Vinyals, O. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Zeng, Z., Goodman, R. M., and Smyth, P. Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6):976–990, 1993.

Zhang, S., Wu, Y., Che, T., Lin, Z., Memisevic, R., Salakhutdinov, R. R., and Bengio, Y. Architectural complexity measures of recurrent neural networks. In *NIPS*, pp. 1822–1830, 2016.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. Recurrent highway networks. In *ICML*, pp. 4189–4198, 2017.