Abstract—Recurrent neural networks are a widely used class of neural architectures. They have, however, two shortcomings. First, they are often treated as black-box models and as such it is difficult to understand what exactly they learn as well as how they arrive at a particular prediction. 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) non-regular languages such as balanced parentheses and palindromes where external memory is required; and (3) real-word sequence learning tasks for sentiment analysis, visual object recognition and text categorisation. We show that state-regularization (a) simplifies the extraction of finite state automata that display an RNN’s state transition dynamic; (b) forces RNNs to operate more like automata with external memory and less like finite state machines, which potentiality leads to a more structural memory; (c) leads to better interpretability and explainability of RNNs by leveraging the probabilistic finite state transition mechanism over time steps.

Index Terms—Automata extraction, explainability, interpretability, memorization, recurrent neural networks, state machine

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. However, 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 deterministic finite automata (DFAs) from trained RNNs [22], [60], [61] 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 [3], [13], [56] and theoretically [1], [42], [73], 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 dynamic 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 RNN.

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 [28], [52]. 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 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. A preliminary version of this work appeared in [74].
2 BACKGROUND

sr-RNNs add state-regularization to RNNs and therefore we first define RNNs (Section 2.1). Furthermore, given an sr-RNN, we describe how a DFA can be extracted faithfully and thus we describe deterministic finite & pushdown automata (Section 2.2).

2.1 Recurrent Neural Networks (RNNs)

RNNs are powerful learning machines. Siegelmann and Sontag [48], [49], [50], for instance, proved that a variant of Elman-RNNs [15] can simulate a Turing machine. Given an input $x_t$ at $t$th time step
\[ h_t = \sigma(W_{x}x_t + U_{h}h_{t-1} + b_h), \]  
\[ y_t = \sigma(W_{y}x_t + b_y), \]
where $W, U, b_h, b_y$ are parameters, $\sigma$ is the sigmoid activation function, $h$ is the hidden representation and $y$ the output. The key issue for an RNN is to learn to preserve information over long time steps due to vanishing gradients. To alleviate this, several RNN variations have been proposed. Two popular ones are Long Short-Term Memory (LSTMs) [29] and Gated Recurrent Units (GRUs) [8]. An LSTM consists of an input gate $i$, a forget gate $f$, a memory cell $c$ and an output gate $o$

\[ f_t = \sigma(W_{f}x_t + U_{f}h_{t-1} + b_f), \]  
\[ i_t = \sigma(W_{i}x_t + U_{i}h_{t-1} + b_i), \]  
\[ o_t = \sigma(W_{o}x_t + U_{o}h_{t-1} + b_o), \]  
\[ c_t = f_t \odot c_{t-1} + i_t \odot c_t, \]  
\[ h_t = o_t \odot \phi(c_t); \]

a GRU has an update gate $z$ and a reset gate $r$

\[ z_t = \sigma(W_{z}x_t + U_{z}h_{t-1} + b_z), \]  
\[ r_t = \sigma(W_{r}x_t + U_{r}h_{t-1} + b_r), \]  
\[ h_t = \phi(W_{h}x_t + r_t \odot h_{t-1} + b_h), \]  
\[ y_t = z_t \odot h_t + (1 - z) \odot h_{t-1}, \]

where $\phi$ is hyperbolic tangent function and $\odot$ is the element-wise multiplication operation. Recent work considers the more practical situation where RNNs have finite precision and linear computation time in their input length [62].

2.2 Deterministic Finite & Pushdown Automata

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 is intended to make 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 $\Sigma^{*}$ be the language over the alphabet $\Sigma$ and let $\epsilon$ be the empty sequence. A DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ consisting of a finite set of states $Q$, a finite set of input tokens $\Sigma$ called the input alphabet, a transition function $\delta : Q \times \Sigma \rightarrow Q$, a start state $q_0$ and a set of accept states $F \subseteq Q$. A sequence $w$ is accepted by the DFA if the application of the transition function, starting with $q_0$, leads to an accepting state. Fig. 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 can be described by a DFA.

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

3 RELATED WORK

Our sr-RNNs and applications relate to four different lines of work. First, we discuss existing works that examine how to extract DFAs from RNNs (Section 3.1). Second, we look at alternative options for regularizing RNNs (Section 3.2). Third, we describe other RNN extensions that modify the state or add an external memory (Section 3.3). Fourth, we list approaches to better understand RNNs and discuss how sr-RNNs can support with this effort (Section 3.4).

3.1 Extracting DFAs From RNNs

Extracting DFAs from RNNs goes back to work on first-generation RNNs in the 1990s [22], [70]. These methods perform a clustering of hidden states after the RNNs are trained [18], [22], [59]. Recent work introduced more sophisticated learning approaches to extract DFAs from LSTMs and GRUs [61]. The latter methods tend to be more successful in finding DFAs behaving similar to the underlying RNN. In contrast to all existing methods, sr-RNNs learn an explicit set of states which facilitates the extraction of DFAs from memory-less sr-RNNs by modelling exactly their state transition dynamics. A different line of work attempt to learn more interpretable RNNs [16], or rule-based classifiers from RNNs [44].

3.2 Regularizations of RNNs

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 [67], variational dropout [19], weight-dropped LSTMs [41], Zoneout [35]
and noise injection [14]. Two approaches that can improve convergence and generalization capabilities are batch normalization [9] and weight initialization strategies [36] for RNNs. In contrast, the proposed sr-RNNs regularize the number of hidden states to a finite set of states. As a result, LSTMs with state regularization can learn in a more structural manner, which leads to improved generalization.

3.3 State and External Memory Extensions to RNNs

The work most similar to sr-RNNs are self-clustering RNNs [70]. These RNNs learn discretized states, that is, binary valued hidden state vectors, and it can be shown that these networks generalize better to longer input sequences. Contrary to self-clustering RNNs, we propose an end-to-end differentiable probabilistic state transition mechanism between cell applications.

Stochastic RNNs are a class of generative recurrent models for sequence data [4], [17], [23]. They model uncertainty in the hidden states of an RNN by introducing latent variables. In contrast 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 nor can they improve DFA extraction.

There are proposals for extending RNNs with various types of external memory. Representative examples are the neural Turing machine [24], improvements thereof [25], memory network [63], associative LSTM [12], and RNNs augmented with neural stacks, queues, and deques [26]. 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. We hope, however, that differentiable neural computers could benefit from state-regularization.

3.4 Understanding RNNs

Approaches for understanding CNNs [51], [69], [71] have been explored extensively. Studies for interpreting and explaining RNNs are less common. [32] revealed the existence of interpretable LSTM cells with character-level language models. [38] visualized neural language models. [52] presented a visual analysis tool (namely, LSTMVIS) for visualizing the raw gate activations of LSTMs on understanding these hidden state dynamics over sequences. While the methods can identify the semantic correlations between hidden cells and abstract attributes or concepts, it is still not obvious how to explain the prediction for given inputs. One of the most recent methods from [44] describe a method to extract simple phrase patterns for determining LSTM predictions. With state-regularization we are able to increase the interpretability of RNNs by inspecting the probabilistic state transition over time steps and by directly extracting automata from trained RNN models.

4 STATE-REGULARIZED RECURRENT NETWORKS

The standard recurrence of an RNN is \( h_t = f(h_{t-1}, x_t) \) where \( h_{t-1} \) is the hidden state vector at time \( t-1 \) and \( h_t \) and \( x_t \) are the hidden state and the input symbol at time \( t \), respectively. We refer to RNNs whose unrolled cells are only connected through gated hidden states \( h \) as RNNs without \( \infty \)-memory. This is because values of gated hidden states \( h \) can only be in a particular 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 is without \( \infty \)-memory, while LSTMs have \( \infty \)-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

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

For the sake of completeness, we include the cell state \( \mathbf{c} \) here, which is absent in RNNs without \( \infty \)-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 \( \Delta^d := \{ \lambda \in \mathbb{R}_{d}^d \mid \| \lambda \| = 1 \} \) be the \((D-1)\) probability simplex. The stochastic component maintains \( k \) learnable centroids \( s_1, \ldots, s_k \) of size \( d \) which we often write as the column vectors of a matrix \( S \in \mathbb{R}^{d \times k} \). 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

\[
\mathbf{a} = \omega(\mathbf{S}, \mathbf{u}_t) \text{ with } \mathbf{a} \in \Delta^k.
\]

Crucially, instances of \( \omega \) should be differentiable to facilitate end-to-end training. Typical instances of the function \( \omega \) are based on the dot-product, normalized into a probability distribution

\[
\alpha_i = \frac{\exp(\langle \mathbf{u}_t, \mathbf{s}_i \rangle / \tau)}{\sum_{i=1}^k \exp(\langle \mathbf{u}_t, \mathbf{s}_i \rangle / \tau)}.
\]

Here, \( \cdot \) is the inner product between two vectors and \( \tau \) is a temperature parameter that can be used to anneal the probabilistic state transition behavior. The lower \( \tau \) the more \( \mathbf{a} \) resembles the one-hot encoding of a centroid. The higher \( \tau \) the more uniform \( \mathbf{a} \) becomes. Equation (15) is reminiscent of the equations of attentive mechanisms [2], [57]. However, instead of attending to the hidden states, sr-RNNs attend to the \( k \) centroids to compute transition probabilities. Each \( \alpha_i \) is the probability of the RNN to transition to centroid (state) \( i \) given the vector \( \mathbf{u}_t \) for which we write \( p_{u_t}(i) = \alpha_i \). The method has been recently introduced to estimate the uncertainty of RNN [58] and transformer models [45].

4.1 State Transition Mechanisms

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

\[
\text{sample } j \sim p_{u_t} \text{ and set } \mathbf{h}_t = \mathbf{s}_j.
\]
Assigning a particular centroid to be a hidden state, i.e., $h_t = s_j$, is equivalent to an one-hot encoding of centroids. However, the direct application of arg max renders the sr-RNN not end-to-end differentiable. To ensure end-to-end differentiability, there are three possible alternative to represent states $s_i$: (1) soft argmax, (2) gumbel-softmax [27, 30, 33] and (3) mixture of centroids.

First, a soft and differentiable version of arg max can be achieved using Equation (15) with a low temperature parameter $\tau$ and as $\tau$ approaches 0, $\alpha$ approximates a one-hot distribution and is differentiable.

$$\alpha = \text{one-hot} \left( \arg \max_i \left( \frac{\exp((u_i \cdot s_i)/\tau)}{\sum_{j=1}^{k} \exp((u_j \cdot s_j)/\tau)} \right) \right).$$  \hspace{1cm} (17)

when $\tau$ approaches 0, the $\alpha$ approximates one-hot distribution, but it offers differentiability to sr-RNN.

Second, with the gumbel-trick it is possible to draw samples $z$ from a categorical distribution given by parameters $\theta$, that is,

$$z = \text{one-hot} (\arg \max_i [g_i + (u_i \cdot s_i)]), i \in [1 \ldots k],$$  \hspace{1cm} (18)

where $g_i$ are i.i.d. samples from the Gumbel$(0, 1)$, that is, $g = -\log (-\log (u)), u \sim \text{uniform}(0, 1)$. Because the arg max operator breaks end-to-end differentiability, the categorical distribution $z$ can be approximated using the differentiable softmax function [30, 33]. This enables us to draw a $k$-dimensional sample vector $\alpha \in \Delta^{k-1}$, where $\Delta^{k-1}$ is the $(k-1)$-dimensional probability simplex. Each instance $\alpha_i$ in $\alpha$ is assigned a probability, that is,

$$\alpha_i = \frac{\exp((u_i \cdot s_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((u_j \cdot s_j) + g_j)/\tau)}, i \in [1 \ldots k],$$  \hspace{1cm} (19)

where $\tau$ is a temperature and $\alpha$ approaches $z$ as $\tau \rightarrow 0$. Recently [58] showed this approach is able to learn better-calibrated models.

Third, it is possible to set the hidden state $h_t$ to be the probabilistic mixture of the centroids

$$h_t = \sum_{i=1}^{k} \alpha_i s_i.$$  \hspace{1cm} (20)

Fig. 2. Three possible instances of an sr-RNN corresponding to Equations (17), (19) and (20) respectively.

Every internal state $h$ of the sr-RNN, therefore, is computed as a weighted sum $h = \alpha_1 s_1 + \ldots + \alpha_k s_k$ of the centroids $s_1, \ldots, s_k$ with $\alpha \in \Delta^k$. Here, $h$ is a smoothed variant in contrast to a hard assignment to one of the centroids. Fig. 2 depicts the three variants of the proposed sr-RNNs.

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 $l$th argument of the tuple corresponding to the centroids of the $l$th 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 for $\tau$, 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 memory-less sr-RNNs straight-forward. Instead of applying post-training DFA extraction as in previous work [60, 61], we extract the true underlying DFA directly from the sr-RNN. Specifically, sr-RNNs don’t need an intermediate step that extracts representations from pre-trained RNN models and performs clustering e.g., k-means over the representations. This is automatically done by Equations (14), (15), (16) and Algorithm 1.1

- 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 [70]. The proposed state regularization stabilizes the hidden states for longer sequences.

Next, 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 4.1.** The state transition behavior of an sr-RNN without $\infty$-memory using Equation (16) is identical to that of a probabilistic finite state automaton.

**Theorem 4.2.** For $\tau \rightarrow 0$ the state transition behavior of an sr-RNN without $\infty$-memory (using Equations (16) or (20)) is equivalent to that of a deterministic finite automaton.

The proofs of the theorems are part of the appendix, available online.

1. It can be integrated to the training loop and output extract DFA at every epoch. Importantly, we don’t need to explicitly tune the number of clusters $k$. 

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
4.2 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.

Algorithm 1. Learning Transition Function (Counts-Based)

| Input: | model M, dataset D, alphabet Σ, start token s |
|--------|-------------------------------------------------|
| Output:| transition function δ |

1: /* the transition prob. of start token */
2: \(\Phi[ (s_i, x_i, s_j)] = 0\), \(i, j \in \{1, \ldots, k\}\), \(x_i \in \Sigma\)
3: \(\alpha = M(s), \alpha = (a_i), i \in \{1, \ldots, k\}\)
4: /* select the next state */
5: \(i = \arg \max_{a \in \{1, \ldots, k\}} (M(s_i)), \ s_{\text{start}} = s_i\)
6: /* compute and update for each input symbol */
7: for \(x = (x_1, x_2, \ldots, x_T) \in D\) do
8: for \(i \in \{1, \ldots, T\}\) do
9: \(j = \arg \max_{a \in \{1, \ldots, k\}} M(x_i), s_{\text{end}} = s_j\)
10: \(\Phi[ (s_{\text{start}}, x_i, s_{\text{end}})] = \Phi[ (s_{\text{start}}, x_i, s_{\text{end}})] + 1\)
11: \(s_{\text{start}} = s_{\text{end}}\)
12: end for
13: end for
14: /* compute transition function based on transition counts */
15: for \(i, j \in \{1, \ldots, k\}\) and \(x_i \in \Sigma\) do
16: \(\delta(s_i, x_i) = \arg \max_{a \in \{1, \ldots, k\}} \Phi[ (s_i, x_i, s_j)]\)
17: end for

We developed two possible approaches, transition counts-based and mean transition probability-based, to extract DFAs. First, we adapt previous work [46, 60] to construct the transition function of an sr-RNN. We begin with the start token of an input sequence, compute the transition probabilities \(\alpha\), 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 \(s_i\) and input token \(a \in \Sigma\) to the following states (including self-loops). After obtaining the transition counts, we keep only the most frequent transitions and discard all other transitions. Concretely, Algorithm 1 presents the pseudo-code of DFA extraction, where \(\Phi[ (i, x_i, s_j)]\) is a dictionary of transitions and counts, where the tuple \((s_i, x_i, s_j)\) denotes a transition from centroid \(s_i\) to \(s_j\), given an input symbol \(x_i\) and \(\delta\) is the returned transition function.

As a corollary of Theorem 4.2, we have that, for \(\tau \to 0\), the extracted transition function becomes increasingly identical to the transition function of the DFA learned by the sr-RNN. Fig. 3 shows that for a wide range of temperatures (including the standard softmax temperature \(\tau = 1\)) the transition behavior of an 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.

4.3 Learning Non-Regular 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 Fig. 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 behave more like PDAs and less like DFAs. The transition function \(\delta\) 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, 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 the memory. Precisely this is accomplished for LSTMs with peephole connections [21]. Concretely, to update the memory, the cell state of a LSTM incorporates forget, input and output gates

\[
f_i = \sigma(W_i^x + R_i h_{t-1} + p_i^f \odot c_{t-1} + b_i^f),
\]
\[
i_t = \sigma(W_i x_t + R_i h_{t-1} + p_i^o \odot c_{t-1} + b_i^o),
\]
\[
o_t = \sigma(W_o x_t + R_o h_{t-1} + p_o^o \odot c_t + b_o^o),
\]
\[
f_i = \sigma(W_i^x + R_i h_{t-1} + p_i^f \odot c_{t-1} + b_i^f),
\]
where \( h_{t-1} \) is the output of the previous cell’s stochastic component; \( W_s \) and \( R_s \) are the matrices of the original LSTM; the \( p_s \) are the parameters of the peephole connections; and \( \odot \) is the elementwise multiplication. We show empirically that the resulting sr-LSTM-r operates like a DPDA, incorporating the current cell state when making decisions about changes to the next cell state.

### 4.4 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. The end token lets the sr-RNN consider both the cell state and the hidden state to make the accept/reject decision. 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.

### 4.5 Understanding RNN Models and Predictions

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. We argue that the proposed probabilistic state transition mechanism helps to understand RNN in: (1) model interpretation, what the RNN models learned from training data and (2) prediction explanation, the explanation for a specific prediction. We use the concept and definition of interpretation and explanation from Montavona et al. [43].

#### Algorithm 2. Generating Model Prototype

**Input:** trained model \( M \), training dataset \( D_{\text{train}} \)

**Output:** the top \( N \) prototypical words for each centroid \( V_i = \{x_i\}^N, i \in \{1, \ldots, k\} \)

1. \^ initialise an empty set for each centroid \^ 
2. \( s_i = \{\}, i \in \{1, \ldots, k\} \)
3. for \( z = (x_1, x_2, \ldots, x_T) \in D_{\text{train}} \) do
   4. for \( t \in \{1, \ldots, T\} \) do
      5. \^ select the centroid that word \( x_t \) has highest prob. \^ 
      6. \( j = \arg \max_{c \in \{1, \ldots, k\}} |M(x_t)| \)
      7. \^ update the top \( N \) words for \( j \)th centroid \^ 
5. \( V_i \), \( s_i \)
6. end for
7. end for

#### 4.5.1 Extracting RNN Model Prototypes

To understand what RNN models learn on training data, we are interested in what each learned centroid can represent, because the learning of a centroid is essentially a “prototype-based clustering” which is similar to learning vector quantization (LVQ) [34]. When RNN models are trained in a supervised manner, the centroids can be learned to represent the semantic meaning of each categorical class. To represent centroids with most representative inputs, we keep track of the input with highest probability for each centroid. The set of words or pixels with highest transition probability are used to represent a specific centroid. We summarize the procedure in Algorithm 2.

#### 4.5.2 Explaining RNN Predictions

When applying a trained model to predict on test samples, it is often difficult to understand how the model arrived at the prediction. Probabilistic state transition offers a way to highlight the input symbols which trigger a high transition probability to each centroid. This allows us to highlight the inputs which are highly relevant for the final predictions. We summarize this in Algorithm 3.

#### Algorithm 3. Explaining Model Prediction

**Input:** model \( M \), a test sample \( U = \{u_t\}_{t=1}^T \), vocabulary \( \Sigma \) (in language task)

**Output:** model prediction \( y \) and its explanation.

1. \^ initialise an empty word-centroid transition matrix \( P_i, i \in \{1, \ldots, k\} \^\)
2. \^ compute transition prob. for each word \( u \) \^ 
3. for \( u = (u_1, u_2, \ldots, u_T) \in U \) do
   4. \^ output \( y, j, \) last centroid index \( j \), and \( P_i \) \^ 
5. \( y, j, P_i \leftarrow M(u) \)
6. generate explanations (e.g., heatmap or highlight words) with \( \{u_t\}_{t=1}^T, \{P_i\}_{i=1}^k, \Sigma \) by mapping \( P_i^j \) to \( u_t \).
7. end for

### 5 EXPERIMENTS

We conduct four 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 non-regular languages, such as the languages of balanced parentheses and palindromes. Third, we investigate the performance of state-regularized LSTMs on non-synthetic datasets. Last, we visualize the probabilistic state transitions to understand RNN models and explain their predictions.

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 [68] for regular language and RMSPROP [54] with a learning rate of 0.01 and momentum of 0.9 for the rest; (d) do not use dropout or batch normalization of any kind; and (e) use state-regularized RNNs based on Equations (15) and (20) with a temperature of \( \tau = 1 \) (standard softmax). We implemented sr-RNNs with Theano [53]. All experiments were performed on a single Titan Xp with 12G memory. The hyper-parameter were tuned to make sure the vanilla RNNs achieve the best performance.

#### 5.1 Regular Languages and DFA Extraction

We evaluate the DFA extraction algorithm for sr-RNNs on RNNs trained on the Tomita grammars [55], which have been used as benchmarks in previous work [60], [61]. We...
use available code [61] 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 are memory-less and, hence, Theorem 4.2 applies. Whenever the GRU converged within 1 hour to a training accuracy of 100%, we also trained an SR-GRU based on Equations (15) and (20) 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 4.2. In all cases, we could recover the minimal and correct DFA corresponding to the grammars. Fig. 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. Fig. 3 visualizes the transition probabilities for different temperatures and $k = 50$ 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 $\tau \leq 1$.

### 5.2 Non-Regular Languages

We conducted experiments on non-regular languages where external memorization is required. This allows us to investigate whether SR-LSTM behave more like PDAs and, therefore, extrapolate to longer sequences. To this end, we used the context-free language “balanced parentheses” (BP; see Fig. 1 (left)) over the alphabet $\Sigma = \{a, \ldots, z, (, )\}$, used in previous work [61]. 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). Both datasets have 1,000 test samples. The training sequences have nesting depths $d \in [1, 5]$, the validation sequences $d \in [6, 10]$ and the test sequences $d \in [0, 20]$. We trained the LSTM and the SR-LSTMs using curriculum learning as in previous work [61], [66] and using the validation error as stopping criterion. We then applied the trained models to unseen sequences. Table 1 lists the results on 1,000 test sequences with the respective depths and lengths. The results show that both SR-LSTM and SR-LSTM-rs extrapolate better on longer sequences and sequences with deeper nesting. Moreover, the SR-LSTM-r 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 ran experiments on the small BP dataset varying $k$ and keeping everything else the same. Table 2 lists the error rates and Fig. 6 the error curves on the validation data for the SR-LSTM-r and different values of $k$. While two centroids ($k = 2$) result in the best error rates for most sequence types, the differences are not very pronounced. This indicates that the SR-LSTM-r is robust to changes in the hyperparameter $k$. A close inspection of the transition probabilities reveals that the SR-LSTM-r mostly utilizes two states, independent of the value of $k$. These two states are used as accept and reject states. These results show that SR-RNNs generalize and tend to utilize a minimal set states similar to PDAs.

A major hypothesis of ours is that the state-regularization encourages RNNs to operate more like PDAs. To explore

---

**TABLE 1**

| Dataset Models | Large Dataset | Small Dataset |
|----------------|---------------|---------------|
|                | LSTM | SR-LSTM | SR-LSTM-\(\tau\) | LSTM | SR-LSTM | SR-LSTM-\(\tau\) |
| $d \in [1, 10], l \leq 100$ | 0.005 | 0.038 | 0.000 | 0.068 | 0.037 | 0.017 |
| $d \in [10, 20], l \leq 100$ | 0.334 | 0.255 | 0.001 | 0.472 | 0.347 | 0.189 |
| $d \in [10, 20], l \leq 200$ | 0.341 | 0.313 | 0.003 | 0.479 | 0.352 | 0.196 |
| $d = 5, l \leq 200$ | 0.002 | 0.044 | 0.000 | 0.042 | 0.028 | 0.015 |
| $d = 10, l \leq 200$ | 0.207 | 0.227 | 0.004 | 0.409 | 0.279 | 0.138 |
| $d = 20, l \leq 1000$ | 0.543 | 0.540 | 0.020 | 0.519 | 0.508 | 0.380 |

---

**TABLE 2**

| 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 |
and cell state $c$ for a specific input sequence from BP (maintains two 11.1 10.56 11.1 (13.50 7.6 7.2 $¼$ $¼$ 28.4 36.2 41.5 $¼$ $¼$ 10.5 16.7 29.8 14.7 of the LSTM and the (1). The number is taken from 1 10.3 10.8 $¼$ 11.0 9.2 (10.6 curves on the small BP validation data. [36] shows the results. The classification function has the ...on the tasks for sentiment analysis (Section 5.3.1) and digit recognition (Section 5.3.2).

5.3 Performance on Real-World Tasks

Next we test our state regularization on real world datasets, namely on the tasks for sentiment analysis (Section 5.3.1) and test reviews. Each review is labeled as positive or negative. Table 4 lists the results. The sr-LSTM-P is competitive with state of the art methods that also do not use the unlabeled data.

5.3.1 Sentiment Analysis

We evaluated state-regularized LSTMs on the IMDB review dataset [39]. 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. Table 4 lists the results. The sr-LSTM-P is competitive with state of the art methods that also do not use the unlabeled data.

5.3.2 Pixel-by-Pixel MNIST

We also explored the impact of state-regularization on pixel-by-pixel MNIST [36], [37]. 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 5 shows the results. The classification function has the final hidden and cell state as input. Our sr-LSTM-Ps 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-Ps). We achieve 31.9 and 13.6 test error, ...

3. The number is taken from [58]
respectively, for sr-RNNs and sr-GRUs, which is worse than the results for the sr-LSTM-rs. On MNIST 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.

### 6 UNDERSTANDING RNNs WITH PROBABILISTIC STATE TRANSITIONS

In this section, we visualize the learned centroids and the corresponding transition probabilities to understand the working of RNNs. We use the models trained on IMDB and sequential MNIST from the previous section. Additionally, we train a new sr-RNN model on 20NewsGroup dataset. The dataset consists of 18,846 samples, among which 15,076 samples are used for training, and 3,770 samples are used for testing. The task is to classify text document into 20 categories. We built a vocabulary with a size of 10,003 and each word is encoded as one-hot representation. sr-LSTM is able to achieve classification accuracy of 85.6%. We also trained an sr-RNN \((k = 20)\) on a more clean version of the dataset by removing header and footnotes, which achieves an accuracy of 65.7%.

#### 6.1 Understanding RNN Models

Table 6 lists, for each state (centroid), the word with the top transition probabilities leading to this state. Here the sr-RNN is trained with \(k = 5\) centroids. As we can see, only a limited number of centroids are meaningful, for example, the 3rd and 4th represent positive and negative sentiment.

Fig. 7 presents the prototypes for digits 0, 3, 7. Interestingly, we also find the transition functions for each learned centroid are more like the “kernels” in CNNs. Each centroid intends to capture the different types of features. For example, the 7th centroid pays most of its attention to the left part of digits, and the 2nd centroid intends to capture the bold property of digits.

Fig. 8 gives the visual explanations for RNNs predictions on 10 randomly selected MNIST test samples. Table 8 presents the explanations of two sample texts from 20NewsGroup which are categorized to “misc.forsale” and “sci.space” categories. The highlighted words “sale, asking, offer delivery”

---

**TABLE 6**

The Learned Centroids and Their Prototypical Words With the Top-4 Highest Transition Probabilities on the IMDB Dataset

| cent. | words with top-4 highest transition probabilities |
|-------|--------------------------------------------------|
| 1     | but (0.97) hadn’t (0.91) college (0.87) even (0.85) |
| 2     | not (1.0) or (1.0) italian (1.0) never (0.99) |
| 3     | loved (1.0) definitely (1.0) 8 (0.99) realistic (0.99) |
| 4     | no (1.0) worst (1.0) terrible (1.0) poorly (1.0) |

This interprets the sr-LSTM-r model with centroids. The 3rd (4th) centroid is “positive” (“negative”).

---

4. [http://qwone.com/~jason/20Newsgroups/](http://qwone.com/~jason/20Newsgroups/)
and “planetary, orbit, spacecraft, planet” are highly associative to the RNN predictions.

Table 9 presents the predictions of two examples (a positive sample and a negative sample). At each time step, the input words are transitioned between the learned “positive” and “negative” centroids. In the negative example, the representative negative words “no, stupid, worse, skip” are given high transition probabilities to the negative centroid. Similarly, in the bottom example, the words “great” and “laughed” are associated with the positive centroid.

7 DISCUSSION
We believe that the probabilistic finite state transition has some additional nice properties, which we discuss in the following.

**Table 9**
For a Negative (top) and Positive (bottom) IMDB Prediction, The sr-RNN Highlights the Words According to the Probability of Transitioning to the negative(top)/positive(bottom) Centroid (Stronger Highlight Indicates Higher Transition Probability)

**Table 10**
The n-gram phrases extracted from 1K random IMDB samples for positive (green) and negative (red) predictions.

-N-Gram Phase Extraction. With probabilistic finite state transition, we can extend word-level to phrase-level interpretability and explainability. This can be achieved by maintaining an attention window with a size of n on the input sequence. The phrases with the highest mean transition probability can be extracted. Table 10 demonstrates the extracted n-gram phrases for explaining predictions on the IMDB dataset.

-Transitions as Features. The pattern of probabilistic finite state transition that the sr-RNN learnt can be used as a representation. Fig. 9 shows the t-SNE [40] of finite state transition probabilities for test samples. We find that the transition probabilities of categories are discriminative.

-Dimensionality Reduction. Note that, Fig. 9 is not the visualization of d-dimension intermediate representation X ∈ R^d,
but the state transition probabilities $P \in \mathbb{R}^k$ of pixel sequences over $k$ centroids. In most cases $k \ll d$ (in this case, $k = 10, d = 256$), which also suggests a possibility of using probabilistic finite state transition as dimensionality reduction method.

8 Conclusion

State-regularization provides new mechanisms for understanding the workings of RNNs. Inspired by recent DFA extraction work [61], our work simplifies the extraction approach by directly learning a finite set of states and an interpretable state transition dynamic. 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-RNN 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 did. 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 in the future.

References

[1] M. Arjovsky, A. Shah, and Y. Bengio, “Unitary evolution recurrent neural networks,” in Proc. 33rd Int. Conf. Mach. Learn., 2016, pp. 1120–1128.
[2] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” in Proc. Int. Conf. Learn. Representations, 2015.
[3] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” 2018, arXiv:1803.01271.
[4] J. Bayer and C. Osendorfer, “Learning stochastic recurrent networks,” 2014, arXiv:1411.7610.
[5] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” J. Mach. Learn. Res., vol. 3, pp. 993–1022, 2003.
[6] V. Campos, B. Jou, X. G. Nieto, J. Torres, and S.-F. Chang, “Skip RNNG. Learning to skip state updates in recurrent neural networks,” in Proc. Int. Conf. Learn. Representations, 2018.
[7] S. Chang et al., “Dilated recurrent neural networks,” in Proc. 31st Int. Conf. Neural Inf. Process. Syst., 2017, pp. 77–87.
[8] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” 2014, arXiv:1412.3555.
[9] T. Coojmans, N. Ballas, C. Laurent, Ç. Gülcehre, and A. Courville, “Recurrent batch normalization,” in Proc. Int. Conf. Learn. Representations, 2017.
[10] G. E. Dahl, R. P. Adams, and H. Larochelle, “Training restricted boltzmann machines on word observations,” in Proc. 29th Int. Conf. Mach. Learn., 2012, pp. 1163–1170.
[11] A. M. Dai and Q. V. Le, “Semi-supervised sequence learning,” in Proc. 28th Int. Conf. Neural Inf. Process. Syst., 2015, pp. 3079–3087.
[12] I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves, “Associative long short-term memory,” in Proc. 33rd Int. Conf. Mach. Learn., 2016, pp. 1986–1994.
[13] M. Daniluk, T. Rocktäschel, J. Welbl, and S. Riedel, “Frustratingly short attention spans in neural language modeling,” 2017, arXiv:1702.04521.
[14] A. B. Dieng, R. Ranganath, J. Altschuler, and D. M. Blei, “Noising unbiased regularization for recurrent neural networks,” in Proc. 35th Int. Conf. Mach. Learn., 2018, pp. 1252–1261.
[15] J. L. Elman, “Finding structure in time,” Cogn. Sci., vol. 14, no. 2, pp. 179–211, 1990.
[16] J. N. Foerster, J. Gilmer, J. Sohl-Dickstein, J. Chorowski, and D. S. Fallout, “Input-aware affine networks,” in Proc. Int. Conf. Learn. Representations, 2017, pp. 1136–1145.
[17] M. Fraccaro, S. K. Senderby, U. Paquet, and O. Winther, “Sequential neural models with stochastic layers,” in Proc. 30th Int. Conf. Neural Inf. Process. Syst., 2016, pp. 2199–2207.
[18] P. Frasconi and Y. Bengio, “An EM approach to grammatical inference: Input/output training,” in Proc. 12th IAPR Int. Conf. Pat. Recognit., 1994, pp. 289–294.
[19] Y. Gal and Z. Ghahramani, “A theoretically grounded application of dropout in recurrent neural networks,” in Proc. 30th Int. Conf. Neural Inf. Process. Syst., 2016, pp. 1019–1027.
[20] F. A. Gers and E. Schmidhuber, “LSTM recurrent networks learn simple context-free and context-sensitive languages,” IEEE Trans. Neural Netw., vol. 12, no. 6, pp. 1333–1340, Nov. 2001.
[21] F. A. Gers and J. Schmidhuber, “Recurrent nets that time and count,” in Proc. IEEE-NNNS-ENNS Int. Joint Conf. Neural Netw., 2000, pp. 189–194.
[22] C. L. Giles, D. Chen, C. B. Miller, H. H. Chen, G. Z. Sun, and Y. C. Lee, “Second-order recurrent neural networks for grammatical inference,” in Proc. Int. Joint Conf. Neural Netw., 1991, pp. 273–281.
[23] A. Goyal, A. Sordoni, M.-A. Côé, N. Ke, and Y. Bengio, “Z-forcing: Training stochastic recurrent networks,” in Proc. 31st Int. Conf. Neural Inf. Process. Syst., 2017, pp. 6713–6723.
[24] A. Graves, G. Wayne, and I. Danihelka, “Neural turing machines,” 2014, arXiv:1410.5401.
[25] A. Graves et al., “Hybrid computing using a neural network with dynamic external memory,” Nature, vol. 538, no. 7666, pp. 471–476, 2016.
[26] É. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom, “Learning to transduce with unbounded memory,” in Proc. 28th Int. Conf. Neural Inf. Process. Syst., 2015, pp. 1828–1836.
[27] E. J. Gumbel, “Statistical theory of extreme values and some practical applications. A series of lectures,” Number 33. US Govt. Print. Office, 1954.
[28] Y. Hao et al., “Context-free transductions with neural stacks,” in Proc. Conf. Empir. Methods Natural Lang. Process. Workshops, 2018, pp. 306–315.
[29] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997.
[30] E. Jang, S. Gu, and B. Poole, “Categorical reparameterization with gumbel-softmax,” in Proc. Int. Conf. Learn. Representations, 2017. [Online]. Available: https://openreview.net/forum?id=rkEy85ee6
[31] R. Johnson and T. Zhang, “Effective use of word order for text categorization with convolutional neural networks,” in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol., 2015, pp. 103–112.
[32] A. Karpathy, J. Johnson, and L. Fei-Fei, “Visualizing and understanding recurrent networks,” in Proc. Int. Conf. Learn. Representations Workshop, 2016.
[33] A. Kendall and Y. Gal, “What uncertainties do we need in Bayesian deep learning for computer vision?,” in Proc. 31st Int. Conf. Neural Inf. Process. Syst., 2017, pp. 5580–5590.
[34] T. Kohonen, “Learning vector quantization,” in Self-Organizing Maps. Berlin, Germany: Springer, 1995, pp. 175–189.
[35] D. Krueger et al., “Zonestat: Regularizing RNNs by randomly preserving hidden activations,” in Proc. Int. Conf. Learn. Representations, 2017.
