STATE-REGULARIZED RECURRENT NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent networks are a widely used class of neural architectures. They have, however, 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 show that state-regularization (a) simplifies the extraction of finite state automata modeling an RNN’s state transition dynamics, and (b) forces RNNs to operate more like automata with external memory and less like finite state machines.

1 INTRODUCTION

Recurrent neural networks (RNNs) have found their way into numerous applications. Still, RNNs have two major 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) and theoretically (Miller & Hardt, 2018), that recurrent networks offer no benefit on longer non-truncated sequences, at least under certain conditions. Intuitively, RNNs tend to operate more like DFAs with a large number of states and less like automata with external memory.

We introduce state-regularized RNNs which we propose 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-RNN 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. Instead of having to apply a post-hoc DFA extraction, it is possible to determine their (probabilistic and deterministic) state transition behavior more directly. We propose a methods that extracts DFAs truly representing the state transition dynamics of the RNNs. 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. State-regularization forces RNNs to operate less like DFAs with large state spaces and more like automata with external memory.

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.
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 more like DPDAs and less like DFAs by limiting their ability to memorize with hidden states.

A pushdown automata (PDA) is defined as a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ consisting of a finite set of states $Q$; a finite set of input tokens $\Sigma$ called the input alphabet; a transition function $\delta : Q \times (\Sigma \cup \{\bot\}) \times \Gamma \rightarrow Q \times \Gamma^*$; a start state $q_0$; the initial stack symbol $\bot$; 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 $\bot$ 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 of balanced parentheses (BP).

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

$$u_t = f(h_{t-1}, c_{t-1}, x_t).$$

For the sake of completeness, we include the cell state $c$ here, which is absent in memory-less RNNs.

We propose a second component which we refer to as stochastic component. The stochastic component is responsible for modeling the probabilistic state transitions mechanism that lets 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 $\Delta^D := \{ \lambda \in \mathbb{R}^D : \| \lambda \| = 1 \}$ 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.

---

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 $\bullet$ 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.

2 BACKGROUND

The standard recurrence of an RNN is $h_t = f(h_{t-1}, c_{t-1}, x_t)$ where $h_t$ is the hidden state vector at time $t$, $c_t$ is the cell state at time $t$, and $x_t$ is the input symbol at time $t$. We refer to RNNs whose unrolled cells are only connected through the hidden output states $h$ and no additional vectors such as the cell state, as memory-less RNNs. For instance, the family of GRUs (Chung et al., 2014) does not have cell states and, therefore, is memory-less. LSTMs (Hochreiter & Schmidhuber, 1997), on the other hand, are not memory-less 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

$$u_t = f(h_{t-1}, c_{t-1}, x_t).$$

For the sake of completeness, we include the cell state $c$ here, which is absent in memory-less RNNs.

We propose a second component which we refer to as stochastic component. The stochastic component is responsible for modeling the probabilistic state transitions mechanism that lets 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 $\Delta^D := \{ \lambda \in \mathbb{R}^D : \| \lambda \| = 1 \}$ 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.

3 STATE-REGULARIZED RECURRENT NEURAL NETWORKS
shared among all cells. The stochastic component computes, at each time step $t$, a discrete probability distribution from the output $u_t$ of the recurrent component and the centroids of the stochastic component

$$\alpha = g(S, u_t) \text{ with } \alpha \in \Delta^k. \quad (1)$$

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 or the Euclidean distance, normalized into a probability distribution

$$\alpha_i = \frac{\exp \left( \frac{(u_i \cdot s_i)}{\tau} \right)}{\sum_{i=1}^{k} \exp \left( \frac{(u_i \cdot s_i)}{\tau} \right)} \quad (2)$$

$$\alpha_i = \frac{\exp \left( -\frac{\|u_i - s_i\|}{\tau} \right)}{\sum_{i=1}^{k} \exp \left( -\frac{\|u_i - s_i\|}{\tau} \right)} \quad (3)$$

Here, $\tau$ is a temperature parameter that can be used to anneal the probabilistic state transition behavior. The lower $\tau$ the more $\alpha$ resembles the one-hot encoding of a centroid. The higher $\tau$ the more uniform is $\alpha$. Equation (2) is reminiscent of the equations of attentive mechanisms [Bahdanau et al. (2015), Vaswani et al. (2017)]. Instead of attending to the hidden states, however, 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 $u_t$, for which we write $p_{u_t}(i) = \alpha_i$.

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

$$j \sim p_{u_t}, \text{ and set } h_t = s_j. \quad (4)$$

This renders the sr-RNN not end-to-end differentiable, however, and one has to use EM or reinforcement learning strategies which are often less stable and less efficient. A possible alternative is to set the hidden state $h_t$ to be the probabilistic mixture of the centroids

$$h_t = \sum_{i=1}^{k} \alpha_i s_i. \quad (5)$$

Every internal state $h$ of the sr-RNN, therefore, is computed as a weighted sum $h = \alpha_1 s_1 + ... + \alpha_k s_k$ of the centroids $s_1,...,s_k$ with $\alpha \in \Delta^k$. 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 two 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, decode the most probable state sequence and then backpropagate based on a structured loss. Since decoding is possible with Viterbi type algorithms, one could apply recent approaches to differentiable dynamic programming [Mensch & Blondel, 2018]. The probabilistic state transitions of sr-RNNs open up new possibilities for applying more complex differentiable decoding functions. We leave these considerations to future 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 $l$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 \( k \), 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 memory-less SR-RNNs straightforward. 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, it has been shown that the hidden states of RNNs tend to drift off for long input sequences \cite{Zeng1993}. 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 lower the temperature the more memory-less RNNs operate like DFAs.

**Theorem 3.1** The state dynamics of a memory-less SR-RNN using equation 4 is that of a probabilistic deterministic finite automaton.

We can show that the lower the temperature the more memory-less RNNs operate like DFAs.

**Theorem 3.2** For \( \tau \to 0 \) the state dynamics of a memory-less SR-RNN (using either equations 4 or 5) is equivalent to that of a deterministic finite automaton.

### 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 the trained model. This renders the extraction also more efficient. We adapt previous work \cite{Schellhammer1998, Wang2018b} to construct the transition function of a 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
DFA, a behavior we can show to be common when SR standard softmax temperature \( \tau \) is kept.

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 behave more like DPDA 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 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.

#### 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. 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 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. Under review as a conference paper at ICLR 2019.
languages such as the languages of balanced parentheses and palindromes. Third, we investigate the performance of state-regularized LSTMs on non-synthetic datasets.

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 RMSprop with a learning rate of 0.01 and momentum of 0.9; (d) do not use dropout or batch normalization of any kind; and (e) use state-regularized RNNs based on equations 2&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 (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 are memory-less 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 2&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 $k$), the SR-GRU only utilizes the required minimal number of states for each of the grammars. 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

We conducted experiments on nonregular languages where external memorization is required. By reducing the number of centroids we can encourage the SR-LSTM to solely utilize the cell state for memorization. We wanted to investigate whether our hypothesis that SR-LSTM behave more like DFAs and, therefore, extrapolate to longer sequences, is correct. To that end, we used 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 $d \in [1, 5]$ and the validation sequences $d \in [6, 10]$. We trained the vanilla LSTM and the SR-RNNs using the error rate on the validation data as stopping criterion. We then applied the trained models to unseen sequences. Table I 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 ran experiments on the small BP dataset varying $k$ and keeping everything else the same. Table II lists the error rates and Figure 5 shows the error curves on the validation data for the SR-LSTM-P and different values of $k$. While two centroids ($k = 2$) result in the best error rates for most sequence...
Table 2: Error rates of the SR-LSTM-P on the small BP test data for various numbers of centroids ($d$=depth, $l$=length).

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

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

Figure 6: Visualization of hidden state $h_t$ and cell state $c_t$ of the LSTM and the SR-LSTM-P for a specific input sequence from BP. The LSTM memorizes the number of open parentheses both in the hidden and to a lesser extent in the cell state (bold yellow 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 ($\pm 1$). A state vector drift is not observable.

Figure 5: Visualization of hidden state $h_t$ and cell state $c_t$ of the LSTM and the SR-LSTM-P for a specific input sequence from BP. The LSTM memorizes the number of open parentheses both in the hidden and to a lesser extent in the cell state (bold yellow 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 ($\pm 1$). A state vector drift is not observable.

types, the differences are not very pronounced. This indicates that the SR-LSTM-P is robust to changes in the hyperparameter $k$. 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. These results show that SR-RNNs generalize tend to utilize a minimal set states similar to DPDAs.

A major hypothesis of ours is that the state-regularization encourages RNNs to operate more like DPDAs and less like DFAs. To explore this hypothesis, we trained an SR-LSTM with 10 units on the BP data and visualized both the hidden state $h_t$ and the cell state $c_t$ for various input sequences. Similar state visualizations have been used in previous work (Stroebelt et al., 2016; Weiss et al., 2018). Figure [13] plots the hidden and cell states for a specific input. As hypothesized, the LSTM relies primarily on its hidden states for memorization. This is detrimental to its extrapolation performance. 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. The visualization also shows a drifting behavior for the LSTM, in line with observations made for first-generation RNNs (Zeng et al., 1993). Drifting is not observable for the SR-LSTM-P.

We also performed experiments for the nonregular language $ww^{-1}$ (Palindromes) over the alphabet $\Sigma = \{a, \ldots, z\}$. We follow the same experiment setup as for BP and include the details in the appendix. The results of Table [3] add evidence for the improved generalization behavior of state-regularized LSTMs. We will make available all datasets used in the experiments.
State regularization provides new ways to interpret the working of recurrent networks. 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 of the pixels leading to the states. 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 explored the impact of state-regularization on the pixel-by-pixel MNIST digits problem. Here, the 784 pixels of MNIST image are fed to RNNs one by one for classification. This requires the modeling of long-term dependencies. Table 7 shows the results on the normal and permuted versions of pixel-by-pixel MNIST. 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. While the SR-LSTM-P cannot reach state of the art accuracy, we observe a significant improvement compared to the LSTM.

State regularization provides new ways to interpret the working of recurrent networks. 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 of the pixels leading to the states. Table 5 lists, for each state (centroid), the word with the top transition probabilities leading to this state. Figure 7 shows the prototypes associated with 4 states for the MNIST digits “3” and “7”. In contrast to previous methods such as activation maximization [Berkes & Wiskott (2006); Nguyen et al. (2016)], the prototypes are directly generated with the centroids of the trained SR-RNNs, we do not make the assumption that input token embeddings and hidden states are similar.

---

We use the definitions of explainability and interpretability of previous work [Montavon et al. (2017)].

---

<table>
<thead>
<tr>
<th>Methods</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>use additional unlabeled data</td>
<td></td>
</tr>
<tr>
<td>Full+unlabelled+BoW (Maas et al.)</td>
<td>11.1</td>
</tr>
<tr>
<td>LM-LSTM+unlabelled (Dai &amp; Le)</td>
<td>7.6</td>
</tr>
<tr>
<td>SA-LSTM+unlabelled (Dai &amp; Le)</td>
<td>7.2</td>
</tr>
<tr>
<td>do not use additional unlabeled data</td>
<td></td>
</tr>
<tr>
<td>seq2-bown-CNN (Johnson &amp; Zhang)</td>
<td>14.7</td>
</tr>
<tr>
<td>WRRB-M+BoWinc (Dahl et al.)</td>
<td>10.8</td>
</tr>
<tr>
<td>JumpLSTM (Yu et al.)</td>
<td>10.6</td>
</tr>
<tr>
<td>sr-LSTM-P (max pool, $k=50$)</td>
<td>9.8</td>
</tr>
<tr>
<td>sr-LSTM-P (max pool, $k=10$)</td>
<td>9.2</td>
</tr>
<tr>
<td>sr-LSTM-P (max pool, $k=5$)</td>
<td>9.6</td>
</tr>
<tr>
<td>sr-LSTM-P (2 layer, last, $c^1$, $c^2$, $k=10$)</td>
<td>9.4</td>
</tr>
</tbody>
</table>

Table 5: Error rates in % of some SR-LSTM-Ps and state of the art methods for IMDB.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM-LSTM+unlabelled (Dai &amp; Le)</td>
<td>7.6</td>
</tr>
<tr>
<td>SA-LSTM+unlabelled (Dai &amp; Le)</td>
<td>7.2</td>
</tr>
<tr>
<td>Full+unlabelled+BoW (Maas et al.)</td>
<td>11.1</td>
</tr>
<tr>
<td>IRNN (Le et al.)</td>
<td>97.7</td>
</tr>
<tr>
<td>URRN (Arjovsky et al.)</td>
<td>98.8</td>
</tr>
<tr>
<td>Skip LSTM (Campos et al.)</td>
<td>99.0</td>
</tr>
<tr>
<td>WN-LSTM (Coomans et al.)</td>
<td>99.2</td>
</tr>
<tr>
<td>Zoneout (Krueger et al.)</td>
<td>99.4</td>
</tr>
<tr>
<td>Dilated GRU (Chang et al.)</td>
<td>99.3</td>
</tr>
<tr>
<td>LSTM (our implementation)</td>
<td>97.7</td>
</tr>
<tr>
<td>sr-LSTM-P ($k=100$)</td>
<td>98.8</td>
</tr>
<tr>
<td>sr-LSTM-P ($k=50$)</td>
<td>98.6</td>
</tr>
<tr>
<td>sr-LSTM-P ($k=10$)</td>
<td>98.1</td>
</tr>
</tbody>
</table>

Table 7: Accuracy in % of some SR-LSTM-Ps and state of the art methods for 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 trained all models on truncated sequences of up to length 10 but tested on longer sequences (length 100 and 200). The results listed in Table 4 show the improvement in extrapolation performance of the SR-LSTM-P and, to a lesser extent, the SR-LSTM. The table also lists the training time per epoch, indicating that the overhead compared to the standard LSTM 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 explored the impact of state-regularization on the pixel-by-pixel MNIST digits problem [Le et al. (2015)]. Here, the 784 pixels of MNIST image are fed to RNNs one by one for classification. This requires the modeling of long-term dependencies. Table 7 shows the results on the normal and permuted versions of pixel-by-pixel MNIST. 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. While the SR-LSTM-P cannot reach state of the art accuracy, we observe a significant improvement compared to the LSTM.

---

We use the definitions of explainability and interpretability of previous work [Montavon et al. (2017)].
Table 8: The words with the top-4 highest transition probabilities for four centroids. This visualizes the SR-LSTM-P’s behavior and associates centroids with prototypical words.

5 RELATED WORK

In principle, RNNs are powerful learning machines. Siegelmann and Sontag [Siegelmann & Sontag, 1992, 1994; Siegelmann, 2012], for instance, 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 in their input length (Weiss et al., 2018a).

Extracting DFAs from RNNs is an old idea that goes back to work on first-generation RNNs in the 1990s (Giles et al., 1991; Zeng et al., 1993). These and most other extraction 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. In contrast to all existing methods, SR-RNN learn an explicit set of states which facilitates the extraction of DFAs from memory-less SR-RNNs exactly modeling their state transition dynamics.

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), and noise injection (Dieng 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 most similar to SR-RNNs are self-clustering RNNs (Zeng et al., 1993). These RNNs learn discretized states, that is, binary valued hidden state vectors, and show 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 (Bayer & Osendorfer, 2014; Fraccaro et al., 2016; Goyal et al., 2017). They model uncertainty in the hidden states of an RNN by introducing latent variables. A core assumption of this line of work is that RNNs are good at modeling long-term dependencies. 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), 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. We hope, however, that differentiable neural computers could benefit from state-regularization.

6 CONCLUSION

State-regularized RNNs with cell states 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 behavior 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.
REFERENCES


APPENDIX A  IMPLEMENTATION DETAILS

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 RMSProp with a learning rate of 0.01 and momentum of 0.9; (d) do not use dropout or batch normalization of any kind; and (e) use state-regularized RNNs based on equations 2&5 with a temperature of \( \tau = 1 \) (standard softmax). We implemented SR-RNNs with Theano\(^2\). The hyper-parameter were tuned to make sure the vanilla RNNs achieves the best performance. For SR-RNNs we tuned the weight initialization values for the centroids and found that sampling uniformly from the interval \([-0.5, 0.5]\) works well across different datasets.

Table 9 lists some statistics about the datasets and the experimental set-ups. Table 10 shows the length and nesting depth (if applicable) for the sequences in the train, validation, and test datasets.

Table 9: A summary of dataset and experiment characteristics. The values in parentheses are the number of positive sequences.

<table>
<thead>
<tr>
<th>Task</th>
<th>Architecture</th>
<th>Units</th>
<th>( k )</th>
<th>Train</th>
<th>Valid</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomita 1</td>
<td>sr-GRU</td>
<td>100</td>
<td>5, 10, 50</td>
<td>265 (12)</td>
<td>182 (4)</td>
<td>–</td>
</tr>
<tr>
<td>Tomita 2</td>
<td>sr-GRU</td>
<td>100</td>
<td>10, 50</td>
<td>257 (6)</td>
<td>180 (2)</td>
<td>–</td>
</tr>
<tr>
<td>Tomita 3</td>
<td>sr-GRU</td>
<td>100</td>
<td>50</td>
<td>2141 (1028)</td>
<td>1344 (615)</td>
<td>–</td>
</tr>
<tr>
<td>Tomita 4</td>
<td>sr-GRU</td>
<td>100</td>
<td>50</td>
<td>2571 (1335)</td>
<td>2182 (1087)</td>
<td>–</td>
</tr>
<tr>
<td>Tomita 5</td>
<td>sr-GRU</td>
<td>100</td>
<td>50</td>
<td>1651 (771)</td>
<td>1298 (608)</td>
<td>–</td>
</tr>
<tr>
<td>Tomita 6</td>
<td>sr-GRU</td>
<td>100</td>
<td>50</td>
<td>2523 (1221)</td>
<td>2222 (1098)</td>
<td>–</td>
</tr>
<tr>
<td>Tomita 7</td>
<td>sr-GRU</td>
<td>100</td>
<td>50</td>
<td>1561 (745)</td>
<td>680 (288)</td>
<td>–</td>
</tr>
<tr>
<td>BP (large)</td>
<td>sr-LSTM (-P)</td>
<td>100</td>
<td>5</td>
<td>22286 (13025)</td>
<td>6704 (3582)</td>
<td>1k</td>
</tr>
<tr>
<td>BP (small)</td>
<td>sr-LSTM (-P)</td>
<td>100</td>
<td>2,5,10,50,100</td>
<td>1008 (601)</td>
<td>268 (142)</td>
<td>1k</td>
</tr>
<tr>
<td>Palindrome</td>
<td>sr-LSTM (-P)</td>
<td>100</td>
<td>5</td>
<td>229984 (115040)</td>
<td>50k (25k)</td>
<td>1k</td>
</tr>
<tr>
<td>IMDB (full)</td>
<td>sr-LSTM (-P)</td>
<td>256</td>
<td>2,5,10</td>
<td>25k</td>
<td>–</td>
<td>25k</td>
</tr>
<tr>
<td>IMDB (small)</td>
<td>sr-LSTM (-P)</td>
<td>256</td>
<td>2,5,10</td>
<td>25k</td>
<td>–</td>
<td>25k</td>
</tr>
<tr>
<td>MNIST (normal)</td>
<td>sr-LSTM-P</td>
<td>256</td>
<td>10,50,100</td>
<td>60k</td>
<td>–</td>
<td>10k</td>
</tr>
<tr>
<td>MNIST (perm.)</td>
<td>sr-LSTM-P</td>
<td>256</td>
<td>10,50,100</td>
<td>60k</td>
<td>–</td>
<td>10k</td>
</tr>
<tr>
<td>Fashion-MNIST</td>
<td>sr-LSTM-P</td>
<td>256</td>
<td>10</td>
<td>55k</td>
<td>5k</td>
<td>10k</td>
</tr>
</tbody>
</table>

Table 10: The lengths (\( l \)) and depths (\( d \)) of the sequences in the training, validation, and test sets of the various tasks.

APPENDIX B  TOMITA GRAMMARS AND DFA EXTRACTION

The Tomita grammars are a collection of 7 regular languages over the alphabet \{0,1\}\(^3\) Tomita (1982). Table 11 lists the regular grammars defining the Tomita grammars.

We follow previous work Wang et al. (2018b); Schellhammer et al. (1998) to construct the transition function of the DFA (deterministic finite automata).

We follow earlier work Weiss et al. (2018b) and attempt to train a GRU to reach 100\% accuracy for both training and validation data. We first trained a single-layer GRU with 100 units on the data. We use GRUs since they are memory-less. Whenever the GRU converged within 1 hour to

\[http://www.deeplearning.net/software/theano/\]

---


Table 11: The seven Tomita grammars (Tomita 1982).

<table>
<thead>
<tr>
<th>Grammars</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1^*$</td>
</tr>
<tr>
<td>2</td>
<td>$(10)^*$</td>
</tr>
<tr>
<td>3</td>
<td>An odd number of consecutive 1s is followed by an even number of consecutive 0s</td>
</tr>
<tr>
<td>4</td>
<td>Strings not contain a substring “000”</td>
</tr>
<tr>
<td>5</td>
<td>The numbers of 1s and 0s are even</td>
</tr>
<tr>
<td>6</td>
<td>The difference of the numbers of 1s and 0s is a multiple of 3</td>
</tr>
<tr>
<td>7</td>
<td>$0^*1^*0^<em>1^</em>$</td>
</tr>
</tbody>
</table>

Algorithm 1: Computes DFA transition function

Input: pre-trained SR-RNN, dataset D, alphabet $\Sigma$, start token $a$
Output: transition function $\delta$ of the DFA

\begin{align*}
\text{for } i, j \in \{1, \ldots, k\} \text{ and all } x \in \Sigma \text{ do} \\
& \quad T[(c_i, x, c_j)] = 0 \quad \# \text{initialize transition counts to zero} \\
& \{p_{i, t}\}_{t=1}^{k} \leftarrow \text{SR-RNN}(a) \\
& \quad j = \arg \max_{i \in \{1, \ldots, k\}} (p_i) \quad \# \text{compute the transition probabilities for the start token} \\
& \quad c_0 = j \quad \# \text{determine } j \text{ the centroid with max transition probability} \\
\text{for } x = (x_1, x_2, \ldots, x_T) \in D \text{ do} \\
& \quad \text{for } t \in [1, \ldots, T] \text{ do} \\
& \quad \quad \{p_{i, t}\}_{j=1}^{k} \leftarrow \text{SR-RNN}(x_t) \quad \# \text{compute the transition probabilities for the } t\text{-th token} \\
& \quad \quad j = \arg \max_{i \in \{1, \ldots, k\}} (p_i) \quad \# \text{determine } j \text{ the centroid with max transition probability} \\
& \quad \quad c_t = j \quad \# \text{set } c_t \text{, the centroid in } t \text{th step, to } j \\
& \quad \quad T[(c_{t-1}, x_t, c_t)] \leftarrow T[(c_{t-1}, x_t, c_t)] + 1 \quad \# \text{increment transition count} \\
\text{for } i \in \{1, \ldots, k\} \text{ and all } x \in \Sigma \text{ do} \\
& \quad \delta(i, x) = \arg \max_{j \in \{1, \ldots, k\}} T[(i, x, j)] \quad \# \text{compute the transition function of the DFA} \\
\text{return } \delta
\end{align*}

Figure 8: The DFA extracted from the SR-GRU for Tomita grammar 7. The numbers inside the circles correspond to the centroid indices of the SR-GRU. Double circles indicate accept states.

A training accuracy of 100%, we also trained a SR-GRU based on equations 2&5 with $k = 50$ and $\tau = 1$. This was the case for the grammars 1-4 and 7. For grammar 5 and 6, our experiments show that both vanilla GRU and SR-GRU were not able to achieve 100% accuracy. In this case, SR-GRU (97.2% train and 96.8% valid accuracy) could not extract the correct DFA for grammar 5 and 6. The exploration of deeper GRUs and their corresponding SR-GRUs (2 layers as in Weiss et al. 2018b) for DFA extraction could be interesting future work.

Algorithm 1 lists the pseudo-code of the algorithm that constructs the transition function of the DFA. Figure 8 shows the extracted DFA for grammar 7. All DFA visualization in the paper are created with GraphViz.

APPENDIX C  TRAINING CURVES

Figure 9 plots the validation error during training of the LSTM, SR-LSTM, and SR-LSTM-P on the BP (balanced parentheses) datasets. Here, the SR-LSTM and SR-LSTM both have $k = 5$ centroids. The state-regularized LSTMs tend to reach better error rates in a shorter amount of iteration.

https://www.graphviz.org/
Figure 10(left) plots the test error of the LSTM, SR-LSTM, and SR-LSTM-P on the IMDB dataset for sentiment analysis. Here, the SR-LSTM and SR-LSTM both have $k = 10$ centroids. In contrast to the LSTM, both the SR-LSTM and the SR-LSTM do not overfit.

Figure 10(right) plots the test error of the SR-LSTM-P when using either (a) the last hidden state and (b) the cell state as input to the classification function. As expected, the cell state contains also valuable information for the classification decision. In fact, for the SR-LSTM-P it contains more information about whether an input sequence should be classified as positive or negative.

Figure 9: The error curves on the validation data for the LSTM, SR-LSTM, and SR-LSTM-P ($k = 5$) on the large BP dataset (left) and the small BP dataset (right).

Figure 10: The error curves on the test data for the LSTM, SR-LSTM, SR-LSTM-P ($k = 10$) on the IMDB sentiment analysis dataset. (Left) It shows state-regularized RNNs show better generalization ability. (Right) A 2-layer SR-LSTM-P achieves better error rates when the classification function only looks at the last cell state compared to it only looking at the last hidden state.

APPENDIX D VIsUALIZATION, INTERPRETATION, AND EXPLANATION

The stochastic component and its modeling of transition probabilities and the availability of the centroids facilitates novel ways of visualizing and understanding the working of SR-RNNs.

D.1 BALANCED PARENTHESES

Figure 11(left) shows the $k = 5$ learned centroids of a SR-LSTM-P with hidden state dimension 100.

Figure 11(center) depicts the average of the ranked transition probabilities for a large number of input sequences. This shows that, on average, the transition probabilities are spiky, with the highest transition probability being on average 0.83, the second highest 0.16 and so on.

Figure 11(right) plots the transition probabilities for a SR-LSTM-P with $k = 5$ states and hidden state dimension 100 for a specific input sequence of BP.

Figure 12 visualizes the hidden states $h$ of a LSTM, SR-LSTM, and SR-LSTM-P trained on the large BP dataset. The SR-LSTM and SR-LSTM-P have $k = 5$ centroids and a hidden state dimen-
Figure 11: Visualization of a SR-LSTM-p with \(k = 5\) centroids, a hidden state dimension of 100, trained on the large BP data. (Left) visualization of the learned centroids. (Center) mean transition probabilities when ranked highest to lowest. This shows that the transition probabilities are quite spiky. (Right) transition probabilities for a specific input sequence.

Figure 12: Visualizations of the hidden states \(h\) of a vanilla LSTM, an SR-LSTM, and a SR-LSTM-p \((k = 10)\) for a specific input sequence. All models were trained on BP and have hidden state dimension of 100. The LSTM memorizes with its hidden state. The evolution of its hidden states is highly irregular. The SR-LSTM and SR-LSTM-P utilize few states and have a more stable behavior over time.

D.2 Sentiment Analysis

Since we can compute transition probabilities in each time step of an input sequence, we can use these probabilities to generate and visualize prototypical input tokens and the way that they are associated with certain centroids. We show in the following that it is possible to associate input tokens (here: words of reviews) to centroids by using their transition probabilities.
Figure 13: Visualization of hidden state \( h_t \) and cell state \( c_t \) of the LSTM and the \( \text{SR-LSTM-P} \) for a specific input sequence from BP. The LSTM memorizes the number of open parentheses both in the hidden and to a lesser extent in the cell state (bold yellow lines). The memorization is not accomplished with saturated gate outputs and a drift is observable for both vectors. The \( \text{SR-LSTM-P} \) 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 (\( \pm 1 \)). A state vector drift is not observable.

![Figure 13](image)

Table 12: List of prototypical words for the \( k = 10 \) centroids of an \( \text{SR-LSTM-P} \) trained on the IMDB dataset. The top-5 highest transition probability words are listed for each centroid. We colored the positive centroid words in green and the negative centroid words in red.

<table>
<thead>
<tr>
<th>Centroids</th>
<th>Top-5 words with probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>centroid 0</td>
<td>piece (0.465) instead (0.453) slow (0.453) surface (0.443) artificial (0.37)</td>
</tr>
<tr>
<td>centroid 1</td>
<td>told (0.752) mr. (0.647) their (0.616) she (0.584) though (0.561)</td>
</tr>
<tr>
<td>centroid 2</td>
<td>just (0.943) absolutely (0.781) extremely (0.708) general (0.663) sitting (0.587)</td>
</tr>
<tr>
<td>centroid 3</td>
<td>worst (1.0) bad (1.0) pointless (1.0) boring (1.0) poorly (1.0)</td>
</tr>
<tr>
<td>centroid 4</td>
<td>jean (0.449) bug (0.406) mind (0.399) start (0.398) league (0.386)</td>
</tr>
<tr>
<td>centroid 5</td>
<td>not (0.997) never (0.995) might (0.982) at (0.965) had (0.962)</td>
</tr>
<tr>
<td>centroid 6</td>
<td>against (0.402) david (0.376) to (0.376) saying (0.357) wave (0.349)</td>
</tr>
<tr>
<td>centroid 7</td>
<td>simply (0.961) totally (0.805) c (0.703) once (0.656) simon (0.634)</td>
</tr>
<tr>
<td>centroid 8</td>
<td>10 (0.994) best (0.992) loved (0.999) 8 (0.987) highly (0.987)</td>
</tr>
<tr>
<td>centroid 9</td>
<td>you (0.799) strong (0.735) magnificent (0.726) 30 (0.714) honest (0.69)</td>
</tr>
</tbody>
</table>

For the sentiment analysis data (IMDB) we can associate words to centroids for which they have high transition probabilities. To test this, we fed all test samples to a trained \( \text{SR-LSTM-P} \). We determine the average transition probabilities for each word and centroid and select those words with the highest average transition probability to a centroid as the prototypical words of said centroid. Table 12 lists the top 5 words according to the transition probabilities to each of the 5 centroids for the \( \text{SR-LSTM-P} \) with \( k = 10 \). It is now possible to inspect the words of each of the centroids to understand more about the working of the \( \text{SR-RNN} \).

Figure 14 and 15 demonstrate that it is possible to visualize the transition probabilities for each input sequence. Here, we can see the transition probabilities for one positive and one negative sentence for a \( \text{SR-LSTM-P} \) with \( k = 5 \) centroids.
this is quite possibly the worst sequel ever made. the script is unfunny and the acting stinks.

it probably helped that I watched it at midnight. if you want a real scare rent this one!

D.3 MNIST AND FASHION-MNIST

For pixel-by-pixel sequences, we can use the SR-RNNs to directly generate prototypes that might assist in understanding the way SR-RNNs work. We can compute and visualize the average transition probabilities for all examples of a given class. Note that this is different to previous post-hoc methods (e.g., activation maximization Berkes & Wiskott (2006), Nguyen et al. (2016)), in which a network is trained first and in a second step a second neural network is trained to generate the prototypes. Figure 16 visualizes the prototypes (average transition probabilities for all examples from a digit class) of SR-LSTM-P for $k = 10$ centroids. One can see that each centroid is paying attention to a different part of the image.

In addition, SR-RNN can be used to visualize the transition probabilities for specific inputs. To explore this, we trained an SR-LSTM-P ($k = 10$) on the MNIST (accuracy 98%) and Fashion MNIST (86%) data Xiao et al. (2017), having the models process the images pixel-by-pixel as a large sequence. Figure 17 visualizes the transition probabilities with a heatmap for specific input images.
Figure 17: We can visualize the working of an SR-LSTM-P on a specific input image by visualizing the transition probabilities of each of the centroids (here: $k = 10$). (Top) The visualization for some MNIST images. (Bottom) The visualization for some Fashion-MNIST images. The first column depicts the input image and the second to to 11th the state transition probability heatmaps corresponding to the 10 centroids.