State-Regularized Recurrent Neural Networks Supplementary Material

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

A. Proofs of Theorems 3.1 and 3.2

A.1. Theorem 3.1

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

Proof. The state transition function δ of a probabilistic finite state machine is identical to that of a finite deterministic automaton (see section 2) with the exception that it returns a probability distribution over states. For every state q and every input token a the transition mapping δ returns a probability distribution $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_k)$ that assigns a fixed probability to each possible state $q \in \mathcal{Q}$ with $|\mathcal{Q}| = k$. The automaton transitions to the next state according to this distribution. Since by assumption the SR-RNN is using equation 4, we only have to show that the probability distribution over states computed by the stochastic component of a SR-RNN without ∞-memory is identical for every state q and every input token a irrespective of the previous input sequence and corresponding state transition history.

More formally, for every pair of input token sequences $\mathbf{a}_1$ and $\mathbf{a}_2$ with corresponding pair of resulting state sequences $\mathbf{q}_1 = (q_{i_1}, ..., q_{i_n}, q)$ and $\mathbf{q}_2 = (q_{j_1}, ..., q_{j_m}, q)$ in SR-RNN without ∞-memory, we have to prove, for every token $a \in \Sigma$, that $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$, the probability distributions over the states returned by the stochastic component for state q and input token a, are identical. Now, since the RNN is, by assumption, without ∞-memory, we have for both $\mathbf{a}_1, \mathbf{q}_1$ and $\mathbf{a}_2, \mathbf{q}_2$ that the only inputs to the RNN cell are exactly the centroid $\mathbf{s}_q$ corresponding to state q and the vector representation of token a. Hence, under the assumption that the parameter weights of the RNN are the same for both state sequences $\mathbf{q}_1$ and $\mathbf{q}_2$, we have that the output $\mathbf{u}$ of the recurrent component (the base RNN cell) is identical for $\mathbf{q}_1$ and $\mathbf{q}_2$. Finally, since by assumption the centroids $\mathbf{s}_1, ..., \mathbf{s}_k$ are fixed, we have that the returned probability distributions $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ are identical. Hence, the transition behavior of SR-RNN without ∞-memory is identical to that of a probabilistic finite automaton. □

A.2. Theorem 3.2

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

Proof. Let us consider the softmax function with temperature parameter τ

$$\alpha_i = \frac{\exp(b_i/\tau)}{\sum_{i=1}^{k} \exp(b_i/\tau)}$$

for $1 \leq i \leq k$. SR-RNNs use this softmax function to normalize the scores (from a dot product) into a probability distribution. First, we show that for $\tau \to 0^+$, that there is exactly one $M \in \{1, ..., k\}$ such that $\alpha_M = 1$ and $\alpha_i = 0$ for all $i \in \{1, ..., k\}$ with $i \neq M$. Without loss of generality, we assume that there is a $M \in \{1, ..., k\}$ such that $b_M > b_i$ for all $i \in \{1, ..., k\}, i \neq M$. Hence, we can write for $\epsilon_1, ..., \epsilon_k > 0$ as shown in equations (1-3).

Now, for $\tau \to 0$ we have that $\alpha_M \to 1$ and for all other $i \neq M$ we have that $\alpha_i \to 0$. Hence, the probability distribution $\boldsymbol{\alpha}$ of the SR-RNN is always the one-hot encoding of a particular centroid.

By an argument analog to the one we have made for Theorem 3.1, we can prove that for every state $q \in \mathcal{Q}$ and every input token $a \in \Sigma$, the probability distribution $\boldsymbol{\alpha}$ of the SR-RNN is the same irrespective of the previous input sequences and visited states. Finally, by plugging in the one-hot encoding $\boldsymbol{\alpha}$ in both equations 4 and 5, we can conclude that the transition function of a SR-RNN without ∞-memory is identical to that of a DFA, because we always chose exactly one new state. □

B. 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 ADADELTA (Zeiler, 2012)

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

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

$$\alpha_i = \frac{\exp\left(b_i/\tau\right)}{\exp((b_M-\epsilon_1)/\tau)+...+\exp\left(b_M/\tau\right)+...+\exp((b_M-\epsilon_k)/\tau)} \tag{1}$$

$$= \frac{\exp\left(b_i/\tau\right)}{\exp(b_M/\tau)\exp(\epsilon_1/\tau)^{-1}+...+\exp(b_M/\tau)+...+\exp(b_M/\tau)\exp(\epsilon_k/\tau)^{-1}} \tag{2}$$

$$= \frac{\exp\left(b_i/\tau\right)}{\exp(b_M/\tau)\left[\exp(\epsilon_1/\tau)^{-1}+...+1+...+\exp(\epsilon_k/\tau)^{-1}\right]}. \tag{3}$$

Task	Architecture	Units	Centroids (k)	Train	Valid	Test
Tomita 1	SR-GRU	100	5, 10, 50	265 (12)	182 (4)	–
Tomita 2	SR-GRU	100	10, 50	257 (6)	180 (2)	–
Tomita 3	SR-GRU	100	50	2141 (1028)	1344 (615)	–
Tomita 4	SR-GRU	100	50	2571 (1335)	2182(1087)	–
Tomita 5	SR-GRU	100	50	1651 (771)	1298(608)	–
Tomita 6	SR-GRU	100	50	2523 (1221)	2222(1098)	–
Tomita 7	SR-GRU	100	50	1561 (745)	680(288)	–
BP (large)	SR-LSTM (-P)	100	5	22286 (13025)	6704 (3582)	1K
BP (small)	SR-LSTM (-P)	100	2,5,10,50,100	1008 (601)	268 (142)	1K
Palindrome	SR-LSTM (-P)	100	5	229984 (115040)	50K(25K)	1K
IMDB (full)	SR-LSTM (-P)	256	2,5,10	25K	–	25K
IMDB (small)	SR-LSTM (-P)	256	2,5,10	25K	–	25K
MNIST	SR-LSTM (-P)	256	10,50,100	60K	–	10K
Fashion-MNIST	SR-LSTM (-P)	256	10	55K	5K	10K
Copying Memory	SR-LSTM (-P)	128, 256	5,10,20	100K	–	10K
Wikipedia	SR-LSTM (-P)	300	1000	22.5M	1.2M	1.2M

Table 1. A summary of dataset and experiment characteristics. The values in parentheses are the number of positive sequences.

Task	Train l & d	Valid l & d	Test l & d
Tomita 1 -7	l = 0∼13, 16, 19, 22	l = 1, 4,..., 28	–
BP (large)	$d \in [1,5]$	$d \in [6,10]$	$d \in [1,20]$
BP (small)	$d \in [1,5]$	$d \in [6,10]$	$d \in [1,20]$
Palindrome	$l \in [1,25]$	$l \in [26,50]$	$l \in [50,500]$
IMDB (full)	$l \in [11,2820], l_{aver} = 285$	–	$l \in [8,2956], l_{aver} = 278$
IMDB (small)	$l = 10$	–	$l \in [100,200]$
MNIST	$l = 784$	$l = 784$	$l = 784$
Fashion-MNIST	$l = 784$	$l = 784$	$l = 784$
Copying Memory	$l = 100, 500$	–	$l = 100, 500$
Wikipedia	$l = 22.5M$	$l = 1.2M$	$l = 1.2M$

Table 2. The lengths (l) and depths (d) of the sequences in the training, validation, and test sets of the various tasks.

for regular language and RMSPROP (Tieleman & Hinton, 2012) with a learning rate of 0.01 and momentum of 0.9 for the rest; (d) do not use dropout or batch normalization of any kind; and (e) use state-regularized RNNs based on equations 3 and 5 with a temperature of $\tau = 1$ (standard softmax). We implemented SR-RNNs with Theano (Theano Development Team, 2016) [1]. All experiments were performed on a single Titan Xp with 12G memory. 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 1 lists some statistics about the datasets and the experimental set-ups. Table 2 shows the length and nesting depth (if applicable) for the sequences in the train, validation, and test datasets.

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

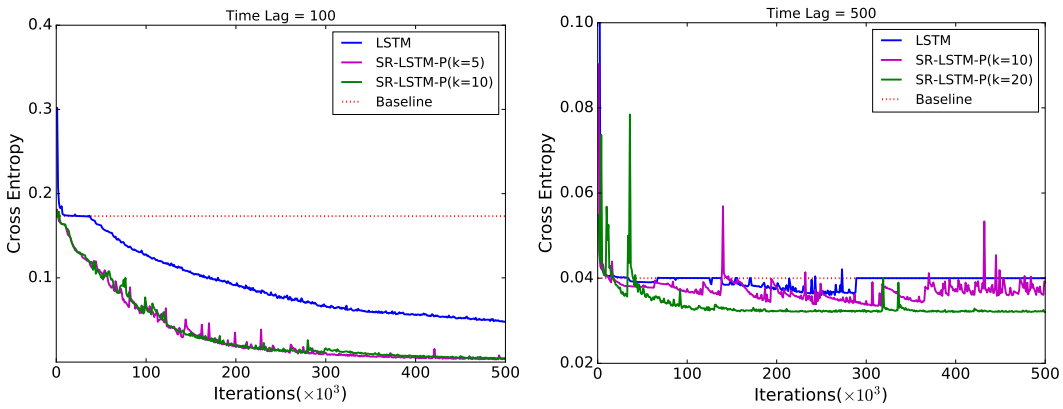


Figure 1. The results (test cross entropy loss) on copying memory problem for time lags $T = 100$ (left) and $T = 500$ (right). We used one recurrent layer with 128 hidden units for $T = 100$, and 256 hidden units for $T = 500$.

C. Experiments on Copying Memory Problem

Besides the introduced balanced paretness, palindrome task, we also conducted experiment on the copying memory problem (Hochreiter & Schmidhuber, 1997). We follow the similar setup as described in (Hochreiter & Schmidhuber, 1997; Arjovsky et al., 2016; Jing et al., 2017). The alphabet Σ has 10 characters $\Sigma = \{c_i\}_{i=0}^{9}$. The input and output sequences have lengths of $T + 2n$, where T is the time lag and n is the length of sequence that need to be memorized and copied. The exemplary input and output are listed as follows.

Input: $c_1c_4c_2c_1c_8c_4c_7c_3c_5c_6$_____$c_9$_____

Output: _____$c_1c_4c_2c_1c_8c_4c_7c_3c_5c_6$

We use $n = 10$, the first n symbols of input sequence are uniformly sampled from $[c_1, c_8]$, the c_0 is used as "blank" symbol (as shown as "_" in above examples. The length of "blank" sequence is T). The c_9 is used as an indicator to require algorithms or models to reproduce the first n symbols with exactly same sequential order. The task is to minimize the average categorical cross entropy at each time step. Similar to (Jing et al., 2017), we used 100000 samples for training and 10000 for test. Differently, all RNN models have only one recurrent layer. We used 128 hidden units for $T = 100$ and 256 hidden units for $T = 500$. The batch size is set to 128. We use RMSPROP with a learning rate of 0.001 and decay of 0.9. A simple baseline is a *memoryless* strategy, which has categorical cross entropy $\frac{10\log(8)}{T+20}$.

Figure 1 presents the performance of the proposed method on copying memory problem for $T = 100$ and $T = 500$. For time log $T = 100$, both standard LSTM and SR-LSTM-Ps are able to beat baseline. Clearly we can see the faster convergence of SR-LSTM-Ps. For $T = 500$, LSTM is hard to beat the baseline, while SR-LSTM-Ps outperforms baseline in a certain margin. Both figures demonstrate the memorization capability of SR-LSTM-Ps over standard LSTM.

D. Experiments on Language Modeling

We evaluated the SR-LSTMs on language modeling task with the Wikipedia dataset (Daniluk et al., 2017)[2]. It consists of 7500 English Wikipedia articles. We used the same experimental setup as in previous work (Daniluk et al., 2017). We used the provided training, validation, and test dataset: 22.5M words in the training set, 1.2M in the validation, and 1.2M words in the test set. We used the 77k most frequent words from the training set as vocabulary. We report the results in Table 3. We used the model with the best perplexity on the on the validation set. Note that we only tuned the number of centroids for SR-LSTM and SR-LSTM-P and used the same hyperparameters that were used for the vanilla LSTMs.

The results show that the perplexity results for the SR-LSTM and SR-LSTM-P outperform those of the vanilla LSTM and LSTM with peephole connection. The difference, however, is modest and we conjecture that the ability to model long-range dependencies is not that important for this type of language modeling tasks. This is an observation that has also been made by previous work (Daniluk et al., 2017). The perplexity of the SR-LSTMs cannot reach that of state of the art methods. The methods, however, all utilize a mechanism (such as attention) that allows the next-word decision to be based on a number of past hidden states. The SR-LSTMs, in contrast, makes the next-word decision only based on the current hidden state.

[2]The wikipedia corpus is available at https://goo.gl/s8cyYa

Model	a	θ_{W+M}	θ_M	Dev	Test
RNN	-	47.0M	23.9M	121.7	125.7
LSTM	-	47.0M	23.9M	83.2	85.2
FOFE HORNN(3-rd order)(Soltani & Jiang, 2016)	-	47.0M	23.9M	116.7	120.5
Gated HORNN(3-rd order)(Soltani & Jiang, 2016)	-	47.0M	23.9M	93.9	97.1
RM(+tM-g) (Tran et al., 2016)	15	93.7M	70.6M	78.2	80.1
Attention (Daniluk et al., 2017)	10	47.0M	23.9M	80.6	82.0
Key-Value (Daniluk et al., 2017)	10	47.0M	23.9M	77.1	78.2
Key-Value Predict (Daniluk et al., 2017)	5	47.0M	23.9M	74.2	75.8
4-gram RNN (Daniluk et al., 2017)	-	47.0M	23.9M	74.8	75.9
LSTM-P	-	47.0M	23.9M	85.8	86.9
SR-LSTM ($k = 1000$)	-	47.3M	24.2M	80.9	82.7
SR-LSTM-P ($k = 1000$)	-	47.3M	24.2M	80.2	81.3

Table 3. The perplexity results for the SR-LSTMs and the state of the art methods. Here, θ_{W+M} are the number of model parameters and θ_M the number of model parameters without word representations. a is the attention window size. The numbers of baseline methods are taken from (Daniluk et al., 2017)

Grammars	Descriptions
1	1^*
2	$(10)^*$
3	An odd number of consecutive 1s is followed by an even number of consecutive 0s
4	Strings not contain a substring "000"
5	The numbers of 1s and 0s are even
6	The difference of the numbers of 1s and 0s is a multiple of 3
7	$0^*1^*0^*1^*$

Table 4. The seven Tomita grammars (Tomita, 1982).

E. Tomita Grammars and DFA Extraction

The Tomita grammars are a collection of 7 regular languages over the alphabet $\{0,1\}$ (Tomita, 1982). Table 4 lists the regular grammars defining the Tomita grammars.

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

We follow earlier work (Weiss et al., 2018) 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 RNNs without ∞-memory. Whenever the GRU converged within 1 hour to a training accuracy of 100%, we also trained a SR-GRU based on equations 3 and 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., 2018)) 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 2 shows the extracted DFA for grammar 7. All DFA visualization in the paper are created with GraphViz [3].

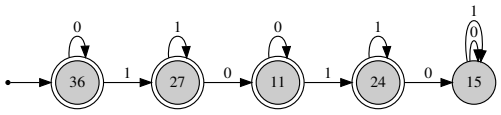


Figure 2. The DFA extracted from the SR-GRU for Tomita grammar 7. The numbers inside the circles correspond to the centroid indices of the SR-GRU. Double circles indicate accept states.

F. Training Curves

Figure 3 plots the validation error during training of the LSTM, SR-LSTM, and SR-LSTM-P on the BP (balanced

[3] https://www.graphviz.org/

Input: pre-trained SR-RNN, dataset **D**, alphabet Σ, start token $\mathtt{s}$
Output: transition function δ of the DFA
for $i, j \in \{1, ..., k\}$ *and all* $x \in \Sigma$ **do**
  $\boldsymbol{\mathcal{T}}[(c_i, x_t, c_j)] = 0$ # initialize transition counts to zero
end
$\{p_i\}_{i=1}^k \leftarrow$ SR-RNN($\mathtt{s}$) # compute the transition probabilities for the start token
$j = \arg\max_{i \in \{1,...,k\}}(p_i)$ # determine j the centroid with max transition probability
$c_0 = j$ # set the start centroid c_0 to j
for $\mathbf{x} = (x_1, x_2, ..., x_T) \in \boldsymbol{D}$ **do**
  for $t \in [1, ..., T]$ **do**
    $\{p_j\}_{j=1}^k \leftarrow$ SR-RNN(x_t) # compute the transition probabilities for the t-th token
    $j = \arg\max_{i \in \{1,...,k\}}(p_i)$ # determine j the centroid with max transition probability
    $c_t = j$ # set c_t, the centroid in time step t, to j
    $\boldsymbol{\mathcal{T}}[(c_{t-1}, x_t, c_t)] \leftarrow \boldsymbol{\mathcal{T}}[(c_{t-1}, x_t, c_t)] + 1$ # increment transition count
  end
end
for $i \in \{1, ..., k\}$ *and* $x \in \Sigma$ **do**
  $\delta(i, x) = \arg\max_{j \in \{1,...,k\}} \boldsymbol{\mathcal{T}}[(i, x, j)]$ # compute the transition function of the DFA
end
return δ

Algorithm 1: Computes DFA transition function

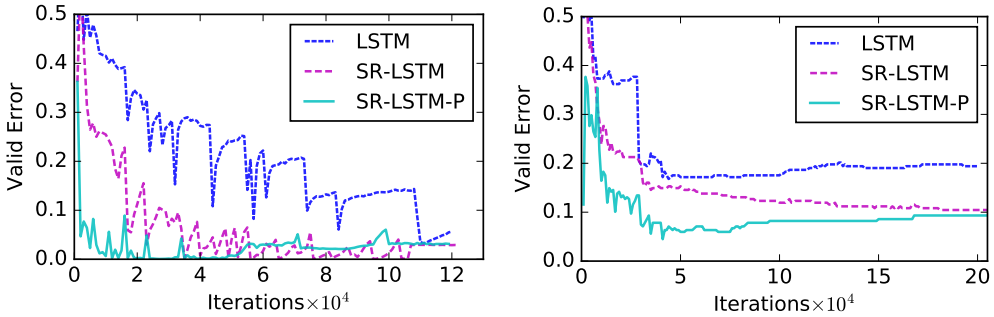


Figure 3. The error curves on the validation data for the LSTM, SR-LSTM, and SR-LSTM-P ($k = 5$) on the large BP dataset (left) and the small BP dataset (right).

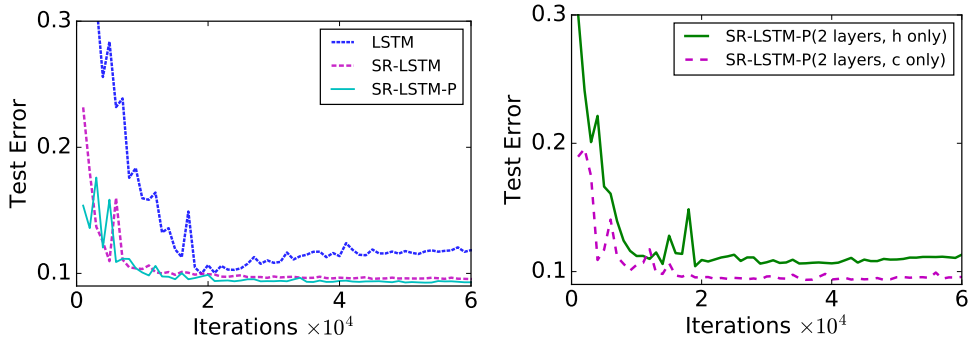
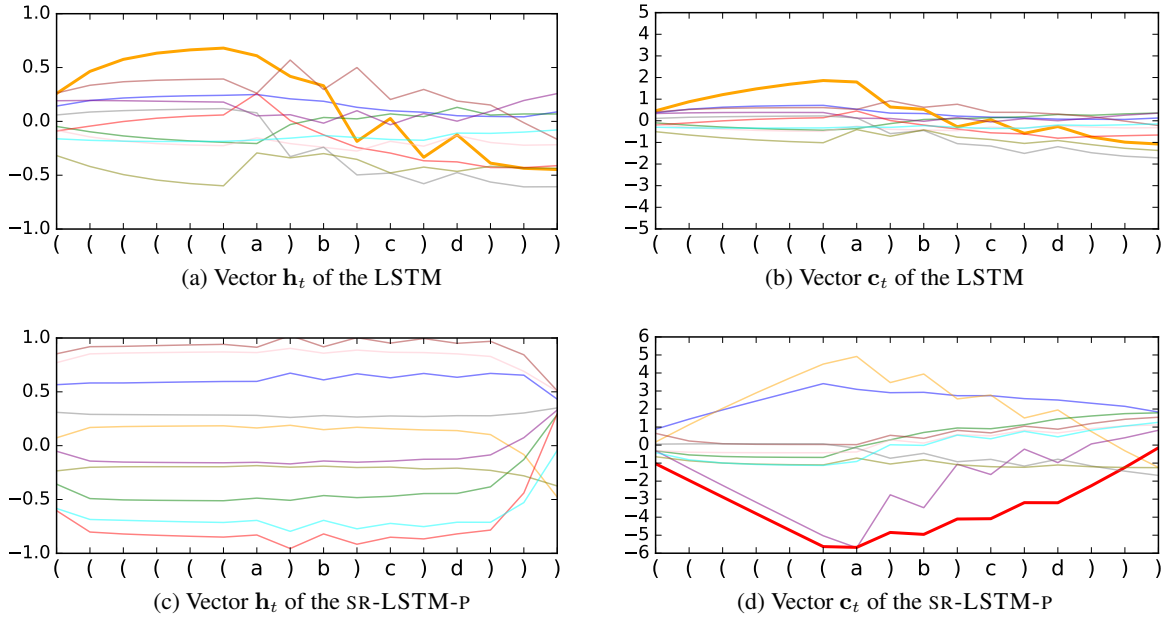


Figure 4. The error curves on the test data for the LSTM, SR-LSTM, SR-LSTM-P ($k = 10$) on the IMDB sentiment analysis dataset. (Left) It shows state-regularized RNNs show better generalization ability. (Right) A 2-layer SR-LSTM-P achieves better error rates when the classification function only looks at the last cell state compared to it only looking at the last hidden state.

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.

Figure 4 (left) plots the test error of the LSTM, SR-LSTM,



(a) Vector $\mathbf{h}_t$ of the LSTM

(b) Vector $\mathbf{c}_t$ of the LSTM

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

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

Figure 5. Visualization of hidden state $\mathbf{h}_t$ and cell state $\mathbf{c}_t$ of the LSTM and the SR-LSTM-P for a specific input sequence from BP. Each color corresponds to one of 10 hidden units. The LSTM memorizes the number of open parentheses both in the hidden and to a lesser extent in the cell state (bold orange lines). The memorization is not accomplished with saturated gate outputs and a drift is observable for both vectors. The SR-LSTM-P maintains two distinct hidden states (accept and reject) and does not visibly memorize counts through its hidden states. The cell state is used to cleanly memorize the number of open parentheses (bold red line) with saturated gate outputs (± 1). A state vector drift is not observable (the solutions with less drift to generalize better (Gers & Schmidhuber, 2001)).

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 4 (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.

G. 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.

G.1. Balanced Parentheses

Figure 5 presents the visualization of hidden state $\mathbf{h}_t$ and cell state $\mathbf{c}_t$ of the LSTM and the SR-LSTM-P for a specific input sequence from BP.

Figure 6 (left) shows the $k = 5$ learned centroids of a SR-LSTM-P with hidden state dimension 100.

Figure 6 (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 6 (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 7 visualizes the hidden states $\mathbf{h}$ 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 dimension of 100. One can see that the LSTM memorizes with its hidden states. The evolution of its hidden states is highly irregular. The SR-LSTM and SR-LSTM-P, on the other hand, have a much more regular behavior. The SR-LSTM-P utilizes mainly two states to accept and reject an input sequence.

G.2. Sentiment Analysis

Since we can compute transition probabilities in each time step of an input sequence, we can use these probabilities

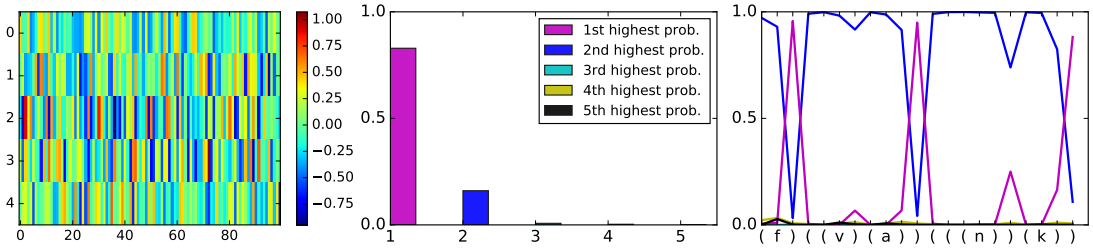


Figure 6. Visualization of a SR-LSTM-P with $k = 5$ centroids, a hidden state dimension of 100, trained on the large BP data. (Left) visualization of the learned centroids. (Center) mean transition probabilities when ranked highest to lowest. This shows that the transition probabilities are quite spiky. (Right) transition probabilities for a specific input sequence.

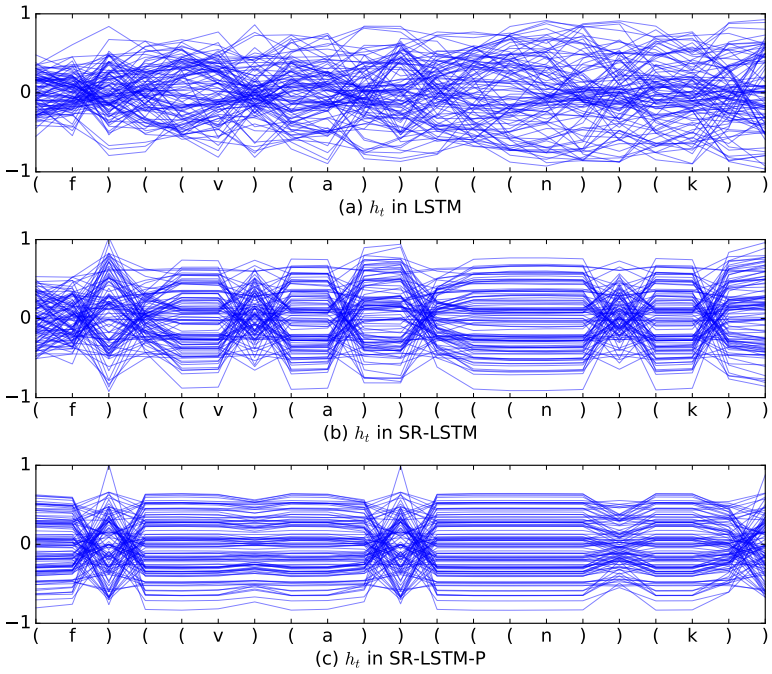


Figure 7. Visualizations of the hidden states $\mathbf{h}$ of a vanilla LSTM, an SR-LSTM, and an SR-LSTM-P ($k = 10$) for a specific input sequence. All models were trained on BP and have hidden state dimension of 100. The LSTM memorizes with its hidden state. The SR-LSTM and SR-LSTM-P utilize few states and have a more stable behavior over time.

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.

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 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 5 lists the top 5 words according to the transition probabilities to each of the 5 centroids for the 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 SR-RNN.

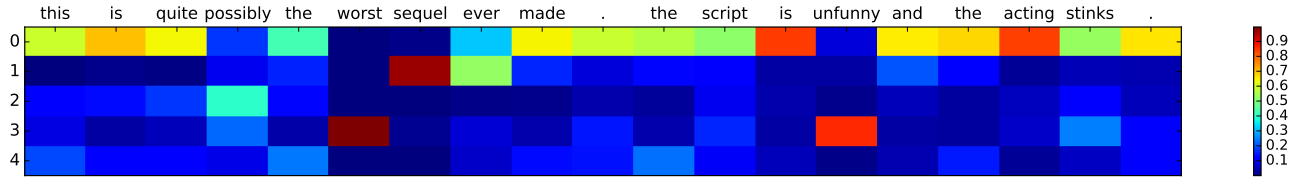
Figure 8 (top) and Figure 8 (bottom) demonstrate that it is possible to visualize the transition probabilities for each input sequence. Here, we can see the transition probabilities for one positive and one negative sentence for a SR-LSTM-P with $k = 5$ centroids. The heatmaps can be viewed as as explanations for SR-RNNs predictions.

G.3. MNIST and Fashion-MNIST

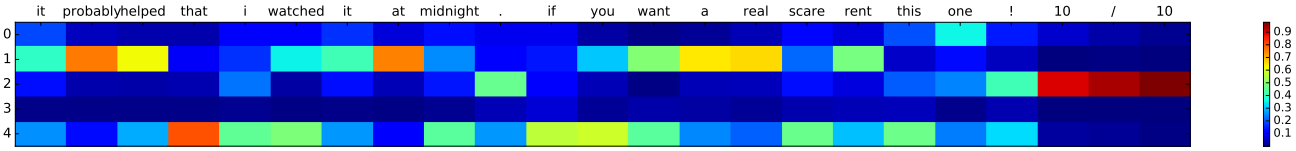
For pixel-by-pixel sequences, we can use the SR-RNNs to directly generate prototypes that might assist in understanding the way SR-RNNs work. We can compute and visualize

Centroids	Top-5 words with probabilities
centroid 0	piece (0.465) instead (0.453) slow (0.453) surface (0.443) artificial (0.37)
centroid 1	told (0.752) mr. (0.647) their (0.616) she (0.584) though (0.561)
centroid 2	just (0.943) absolutely (0.781) extremely (0.708) general (0.663) sitting (0.587)
centroid 3	worst (1.0) bad (1.0) pointless (1.0) boring (1.0) poorly (1.0)
centroid 4	jean (0.449) bug (0.406) mind (0.399) start (0.398) league (0.386)
centroid 5	not (0.997) never (0.995) might (0.982) at (0.965) had (0.962)
centroid 6	against (0.402) david (0.376) to (0.376) saying (0.357) wave (0.349)
centroid 7	simply (0.961) totally (0.805) c (0.703) once (0.656) simon (0.634)
centroid 8	10 (0.994) best (0.992) loved (0.99) 8 (0.987) highly (0.987)
centroid 9	you (0.799) strong (0.735) magnificent (0.726) 30 (0.714) honest (0.69)

Table 5. List of prototypical words for the $k = 10$ centroids of an 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.



(a) Negative input sample



(b) Positive input sample

Figure 8. Visualization of the transition probabilities for an SR-LSTM-P with $k = 5$ centroids trained on the IMDB dataset for a negative (top) and a positive (bottom) input sample.

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 9 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 10 visualizes the transition probabilities with a heatmap for specific input images.

References

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

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

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

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

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

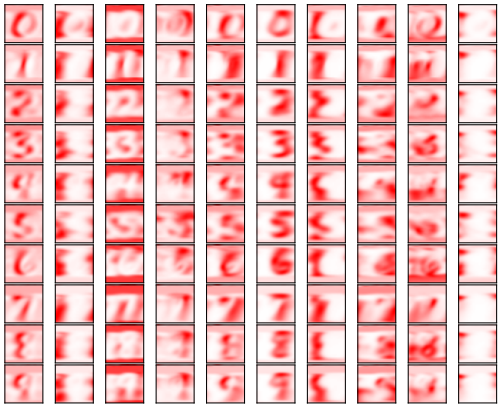


Figure 9. SR-RNNs for interpreting RNN models. Visualization of average transition probabilities of the SR-LSTM-P with $k = 10$ centroids, over all test images. Each row represents a digit class (a concept) and each column depicts the prototype (average transition probability) for each of the centroids.

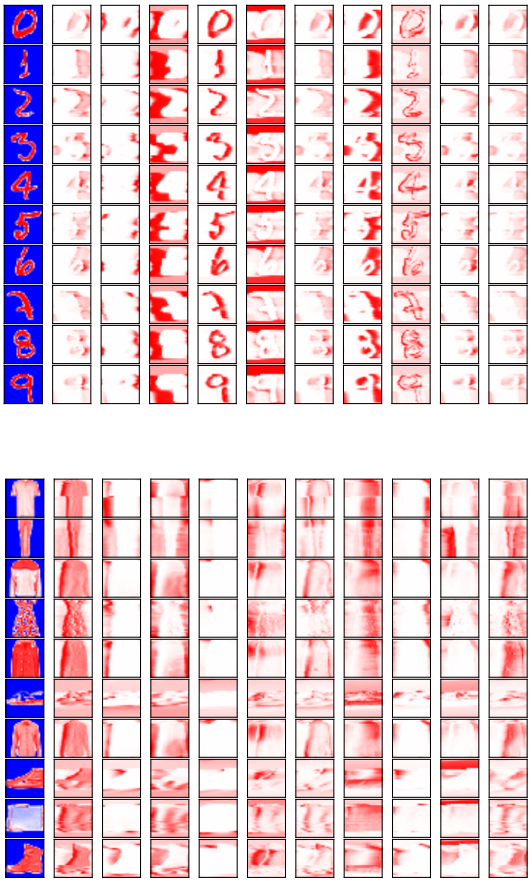


Figure 10. SR-RNNs for explaining RNN predictions. We can visualize the working of an SR-LSTM-P on a specific input image by visualizing the transition probabilities of each of the centroids (here: $k = 10$). (Top) The visualization for some MNIST images. (Bottom) The visualization for some Fashion-MNIST images. The first column depicts the input image and the 2^{nd} to 11^{th} the state transition probability heatmaps corresponding to the 10 centroids.

Jing, L., Shen, Y., Dubcek, T., Peurifoy, J., Skirlo, S., LeCun, Y., Tegmark, M., and Soljačić, M. Tunable efficient unitary neural networks (EUNN) and their application to RNNs. In Precup, D. and Teh, Y. W. (eds.), *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1733–1741, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

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

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

Soltani, R. and Jiang, H. Higher order recurrent neural networks. *arXiv preprint arXiv:1605.00064*, 2016.

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

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

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

Tran, K., Bisazza, A., and Monz, C. Recurrent memory networks for language modeling. *NAACL-HLT*, 2016.

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

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

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

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