Relating RNN layers with the spectral WFA ranks in sequence modelling

Farhana Ferdousi Liza
School of Computing
University of Kent
Canterbury, CT2 7NF, UK
fl207@kent.ac.uk

Marek Grzes
School of Computing
University of Kent
Canterbury, CT2 7NF, UK
m.grzes@kent.ac.uk

Abstract

We analyse Recurrent Neural Networks (RNNs) to understand the significance of multiple LSTM layers. We argue that the Weighted Finite-state Automata (WFA) trained using a spectral learning algorithm are helpful to analyse RNNs. Our results suggest that multiple LSTM layers in RNNs help learning distributed hidden states, but have a smaller impact on the ability to learn long-term dependencies. The analysis is based on the empirical results, however relevant theory (whenever possible) was discussed to justify and support our conclusions.

1 Introduction

Sequence prediction is a problem that involves using historical sequence data (i.e. context) to predict the next symbol or symbols in the sequence. Weighted Finite-state Automata (WFA) and Recurrent Neural Networks (RNNs) provide a general framework for the representation of functions that map strings (i.e. sequential data) to real numbers. Nondeterministic Weighted Finite-state Automata (WFA) map input words to real numbers and are not guaranteed to be tractable (Avni and Kupferman, 2015; Sharan et al., 2017). In general, WFA use hidden states and learning is usually done by the Expectation-Maximisation (EM) algorithm, which is computationally expensive and does not come with a guarantee of global optimality. Spectral learning algorithms for WFA (Balle et al., 2014) provide an alternative to EM that is both computationally efficient and statistically consistent. On the other hand, RNNs are remarkably expressive models. Even a single-layer RNN network has powerful sequence modelling capacity. RNNs are also Turing complete and can represent any computable function (Siegelmann and Sontag, 1991), but the theoretical analysis of even a single-layer RNN is difficult.

Existing research shows that multilayer RNNs are advantageous for efficient sequence modelling (Zaremba et al., 2014; Jozefowicz et al., 2015). However, it is hard to analyse such models theoretically. As a result, in spite of competitive empirical results, it is not clear what kind of additional modelling power is gained by a deep architecture (i.e. more than one hidden layer in RNNs). Stacking RNN layers (in space) is inspired by the multilayer perceptron (MLP) and the hypothesis Bengio et al. (2009) that multiple layers allow the model to have greater complexity by incorporating complex feature representations of each time step. This allows each recurrent level to operate at a different time-scale. For the non-recurrent networks, Bengio et al. (2009) hypothesise that a deep, hierarchical model can be exponentially more efficient at representing some functions than a shallow one. Theoretical (Le Roux and Bengio, 2010; Delalleau and Bengio, 2011; Pascanu et al., 2013) and empirical (Goodfellow et al., 2013; Hinton et al., 2012) work on non-recurrent networks agrees with the above hypothesis. Based on these results, Pascanu et al. (2014) assumed that the MLP-based hypothesis proposed by Bengio et al. (2009) is also true for the recurrent neural networks. The earlier work attempted at capturing large context and reducing the training time by using multilayer RNNs. For example, El Hihi and Bengio (1996) assumed that the layers increase the capacity of learning the context by capturing the improved long-term history, whereas Schmidhuber (2008) argues that the stacked RNN requires less computation per time-step and far fewer training sequences than a single-layer RNN.

Elman (1990) introduced the notion of 'memory' to capture non-fixed long-term contexts through the recurrent layer. When stacking the RNNs, the transition between the consecutive recurrent layers is still shallow (Pascanu et al.,

2014). Thus, stacking the RNNs does not extend the hypothesis of (Bengio et al., 2009) to the recurrent layer that is dedicated for long-term context capture. The empirical results of Zaremba et al. (2014); Jozefowicz et al. (2015) suggested that multilayer RNNs improve sequence modelling. We show empirical evidence that indicates that a multilayer RNN does capture better context as shown by El Hihi and Bengio (1996), but that is achieved across stacked layers instead of the time scale (i.e. instead of recurrent layer). Better learning depends on capturing the improved input representation at each time step and capturing improved long-term dependency from the previous time-steps in a sequence. In this paper, we investigate RNN learning from the formal language perspective using the WFA models, and we show that adding more layers may not be sufficient if the model has to deal with long-term dependencies.

WFA-based models are used for both theoretical studies and sequence prediction tasks including language modelling (Buchsbaum et al., 1998). Evaluating their performance on real and synthetic data can help us to understand the model's hidden state relationship with the RNN layers. In the existing literature, stacking multiple RNN layers (in space) is used to obtain improved accuracy on sequence prediction tasks, but this is done without deeply-justified reasons of such choices. Our experiments and analysis show that the hidden states of a process can be modelled efficiently using multiple layers, but multiple layers may not be sufficient to model long-term dependencies in sequential observations.

In this paper, we use two types of RNN models (one is a single-layer and another is a two-layer stacked RNN network) and a WFA. All methods were evaluated on fifteen datasets to answer the following question "what is the impact of multiple RNN layers in sequence modelling?". To answer this question, we contrasted the impact of the LSTM layers in RNNs with the rank (i.e. the number of hidden states) in the corresponding WFA models.

2 Data

In this section we introduce the 15 datasets used in the Sequence PredIction ChallengE (SPiCe) in 2016. The datasets consist of 8 synthetic (fully or partially) and 7 real-world datasets. Among the synthetic datasets, four are generated artificially and four are partially synthetic based on real data. Datasets are publicly available[1] and descriptions can be found in (Balle et al., 2017). Our numbering of datasets is consistent with SPiCe'16. The synthetic datasets 1, 2, and 3 were artificially generated based on a Hidden Markov Model (HMM) (Balle et al., 2017). HMM sequences were generated with n states and non-stationary transition probabilities were obtained by partitioning the unit interval [0,1) into n equal sub-intervals and letting the states evolve as $h_{t+1} = h_t + \Phi \bmod 1$, for some irrational number Φ. The emission probabilities were sampled from a Dirichlet distribution. Another synthetic dataset, 12, consists of synthetic data generated using the PAutomaC data generator (Verwer et al., 2014b). Partially synthetic datasets 6 and 9 are based on software engineering and come from the challenge RERS 2013 (Howar et al., 2014). Partially synthetic datasets 14 and 15 contain synthetic data generated from two Deterministic Finite State Automata learned using the ALERGIA algorithm (Carrasco and Oncina, 1994) based on the NLP datasets 4 and 5, respectively.

Real datasets 4 (English Verbs from Penn Treebank), 5 (Character Language Modelling benchmark from Penn Treebank), and 8 (POS from Ancora) all correspond to NLP problems from Penn Treebank (Marcus et al., 1993a) and the Spanish Ancora corpus (Taulé et al., 2008). Dataset 11 (lemmalisation) was created from a lemmatised version of the Fickr-8k dataset (Hodosh et al., 2013). Real dataset 13 (spelling correction) was derived from a Twitter spelling correction corpus (twi, 2010). Real datasets 7 and 10 are protein families sequences taken from the Pfam database (Finn et al., 2015).

3 Sequence Modelling and Evaluation

The Sequence PredictIction ChallengE (SPiCe) (Balle et al., 2017) was an on-line competition to predict the next element of a sequence. The competition scored methods on their performance on both real and synthetic data (see Sec. 2). Training datasets consist of whole sequences and the aim is to learn a model that allows the ranking of potential next symbols for a given test sequence (prefix or context), that is, the most likely options for a single next symbol. Once rankings for all prefixes were submitted by the participants, the score

[1] http://spice.lif.univ-mrs.fr/data.php

($NDCG_5$ explained below) of the submission was computed. The score is a ranking metric based on normalised discounted cumulative gain computed from the ranking of 5 potential next symbols starting from the most probable one. Suppose the test set is made of prefixes $y_1, \ldots, y_M$ and the distinct next symbols ranking submitted for y_i is $(\hat{a}_1^i, \ldots, \hat{a}_5^i)$ sorted from more likely to least likely. The target probability distribution of possible next symbols given the prefix y_i, $p(.|y_i)$, was known to the organisers. Thus, the exact measure for prefix y_i could be computed using the following equation:

$$NDCG_5(\hat{a}_1^i, \ldots, \hat{a}_5^i) = \frac{\sum_{k=1}^{5} \frac{p(\hat{a}_k^i|y_i)}{log_2(k+1)}}{\sum_{k=1}^{5} \frac{p_k}{log_2(k+1)}}$$

where $p_1 \geq p_2 \geq \ldots \geq p_5$ are the top 5 values in the distribution $p(.|y_i)$. More details on this evaluation can be found in (Balle et al., 2017).

4 WFA Models

WFA represent functions for mapping strings to real numbers. WFA include as special instances Deterministic Finite-state Automata (DFAs), hidden Markov models (HMMs), and predictive state representations (PSRs).

Let Σ^* denote the set of strings over a finite alphabet Σ and let λ be the empty word. A WFA with k states is a tuple $A = \langle a_0, a_\infty, A_\sigma \rangle$ where $a_0, a_\infty \in \mathbf{R}^k$ are the initial and final weight vectors respectively, and $A_\sigma \in \mathbf{k} \times \mathbf{k}$ is the transition matrix for each symbol $\sigma \in \Sigma$. A WFA computes a function $f_A : \Sigma^* \rightarrow \mathbf{R}$ defined for each word $x = x_1 x_2 \ldots x_n \in \Sigma^*$ by $f_{A(x)} = a_o{}^\top A_{x_1} A_{x_2} \ldots A_{x_n} a_\infty$.

A WFA A with k states is minimal if its number of states is minimal, i.e., any WFA B such that $f_A = f_B$ has at least k states. A function $f : \Sigma^* \rightarrow R$ is recognisable if it can be computed by a WFA. In this case the rank of f is the number of states of a minimal WFA computing f. Note that this is the key reason why rank (i.e. the number of hidden states) is an important parameter that we exploit in this paper. If f is not recognisable, we let rank(f) $= \infty$.

Approximating distributions over strings is a hard learning problem. Learning WFA has exponential computational complexity (Mohri, 2004). The recent advancement in learning WFA is based on spectral learning, which reduces the computation complexity of learning WFA (Balle et al., 2014).

In this paper we use a Hankel matrix based spectral learning algorithm for WFA. The basic steps of the algorithm are as follows:

S1. Basis Selection: Choose a set of prefixes P and suffixes S

S2. Build a Hankel matrix: The Hankel matrix ($H_f \in \mathbf{R}^{\Sigma^* \times \Sigma^*}$) associated with a function $f : \Sigma^* \rightarrow \mathbf{R}$ is a bi-infinite matrix . In practice, one deals with finite sub-blocks of the Hankel matrix based on the chosen basis in S1, thus $B = (P, S) \subset \Sigma^* \times \Sigma^*$. The corresponding sub-block of the Hankel matrix is denoted by $H \in R^{|P| \times |S|}$. The entry $H(p, s)$ is the value of the target function on the sequence obtained by concatenating prefix p with suffix s. Among all possible basis, we are particularly interested in the ones with the same rank as f. We say that a basis is complete if $rank(H) = rank(f) = rank(H_f)$.

S3. Perform SVD on $H = u\sigma v^\top$.

S4. Use the factorization $F = u\sigma$, $B = v^\top$ and H to recover the parameters of the minimal WFA, following (Hsu et al., 2012, see Sec. 2.3).

The hyperparameters of the learning algorithm (Balle et al., 2014) for retrieving the parameters of the minimum WFA are the number of states n of the target WFA and the basis (i.e. sets of P and S). This n is also a rank of the n-dimensional reconstruction of the Hankel matrix when the best n dimensions of its SVD are used.

We choose a basis that contains most frequent elements (substrings) observed in the sample based on the work by Balle et al. (2012) as this approach was found computationally efficient. The rows and columns of the Hankel matrix correspond to the substrings, and the cells of the Hankel matrix contain the frequencies of the corresponding substrings. In this approach, the length of these substrings along rows (nR) and along column (nC) are also the hyperparameters of the spectral learning algorithm for WFA (Balle et al., 2014).

4.1 Tuning Hyperparameters

Similar to Larochelle et al. (2007), our tuning method included a combination of multi-

resolution search, coordinate ascent, and manual search, with a significant utilisation of the last method. On all datasets, our method first initialises nR and nC to 4 and n to 5. Note that the actual number of rows (columns) in the Hankel matrix is much larger than nR (nC). In the second step, the algorithm starts the process of tuning the number of states n because this was the most important hyperparameter in our preliminary experiments. Random walk is used to select new values of n with the step size being depended on the size of the domain, i.e., the number of observations and the number of sequences. Thus, when nR and nC were kept constant, the value of n was increased or decreased randomly based on the score $NDCG_5$ (Sec. 3), i.e., a form of coordinate ascent was performed on n. After the highest score was achieved by tuning n, n was frozen, and the algorithm used the same randomised procedure to tune nR. Finally, the same procedure was executed to tune the parameter nC.

On some problems, increasing n, nR and nC to large values was not possible as the algorithm became intractable.

5 Neural Models

Theoretically a single-layer RNN network should be able to approximate any computable function. However, it was observed recently that *empirically* multilayer deep RNNs work better than shallower (single layer) ones on some tasks, specifically on natural language processing tasks. For instance, Zaremba et al. (2014) used a stack of Long Short-Term Memory (LSTM) layers for language modelling and in (Sutskever et al., 2014) a 4-layer deep architecture was crucial in achieving good machine translation performance in an encoder-decoder framework. Apart from considering the number of layers as a hyperparameter, most recent works do not explain the advantage of multilayer RNNs. Moreover, the deep RNN language model (Zaremba et al., 2014) is used by numerous other models including Press and Wolf (2016); Gal and Ghahramani (2016). This was also used as a baseline in the exhaustive (over ten thousand different RNN architectures) architecture search by Jozefowicz et al. (2015) in pursuit of a better architecture, and they did not find architectures that were significantly better than the baselines. Therefore, a two-layer RNN is a strong baseline architecture for certain sequence prediction tasks, especially language modelling, where single-layer RNNs are not so powerful.

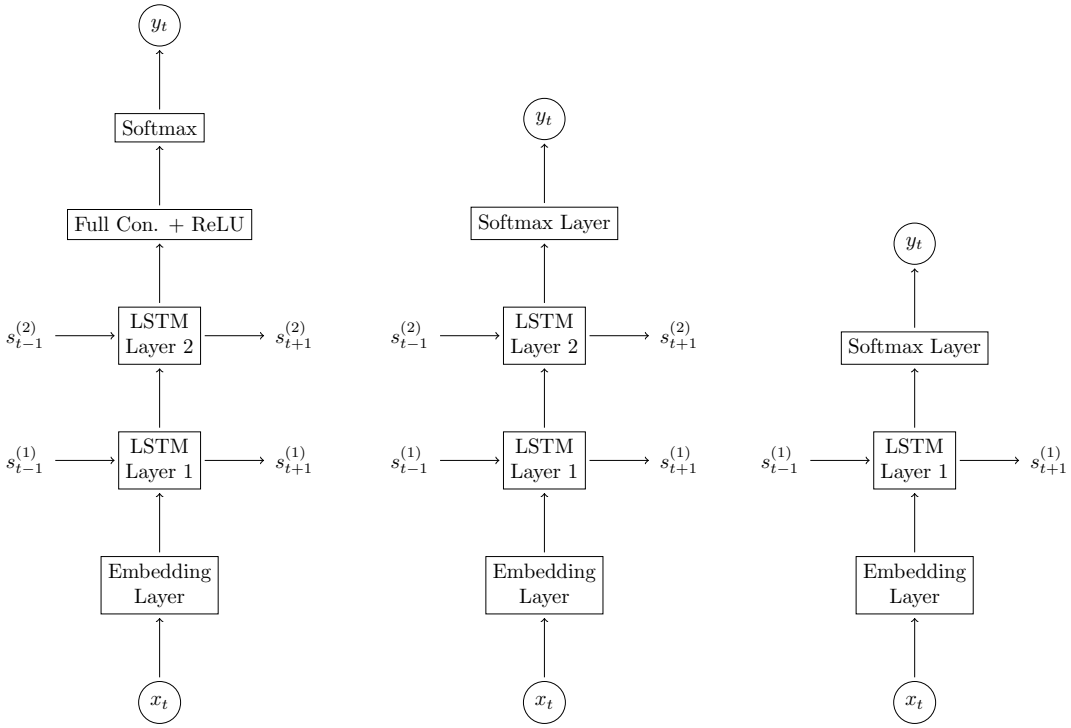
In the SPiCe competition, there were three neural models explored by Shibata and Heinz (2017) that achieved the winning accuracy. Among those models, the basic model is a two-layer stacked LSTM network. There is an all-connected non-linear layer with a Rectified Linear Unit (ReLU) activation function used on top of a stacked LSTM (Fig. 1a). The two-layer LSTM stack was placed on top of the embedding layer that is used to embed each symbol x_t of the sequence at position t. The output layer consists of a softmax layer implementing the softmax activation, which outputs the network's prediction of the next symbol of sequence $y_t = x_{t+1}$.

In this paper, we have simplified the basic architecture in two ways. First, we removed the fully-connected non-linear layer and introduced dropout to all non-recurrent layers (Fig. 1b). Second, we further simplified the model by using just a single layer (Fig. 1c) with dropout at non-recurrent layer. Dropout is an effective regularisation technique for deep neural networks (Hinton et al., 2012). The motivation for removing the all-connected layer is to reduce the number of parameters and the state-of-the-art sequence prediction models (Zaremba et al., 2014) do not usually have an all-connected non-linear layer on top of the stacked LSTM. Stacked LSTMs are expressive enough to capture most of regularities without an additional all-connected non-linear layer. The motivation for applying dropout to all non-recurrent layers is to regularise the whole networks (Zaremba et al., 2014) instead of regularising based on some particular layers (Shibata and Heinz, 2017). We compare against a single-layer LSTM in the results section.

Following Shibata and Heinz (2017) a 'start' symbol and an 'end' symbol were added to both sides of each training sentence. Symbols are fed into the model from the 'start' symbol.

5.1 Relevant Parameters and Parameter Search

Neural models have more hyperparameters than the other models (e.g. WFA). We used the hyperparameters from the baseline work (Shibata and Heinz, 2017) with two major exceptions: we used an LSTM network with one layer and two layers (contrary to the baseline work, we removed fully



(a) Basic architecture (SPiCe) (b) Basic architecture (2-layer) (c) Basic architecture (1-layer)

Figure 1: Neural architectures for our experiments

connected layer as shown in Fig. 1b) and we applied the regularisation (i.e. dropout) differently than the baseline study. We applied dropout to regularise the whole network (except the recurrent layer) instead of just the last two layers (in baseline). The exact values of different parameters can be found in Sec. 6.

6 Experiments

The hyperparameters (see Tab. 1) for the WFA were tuned based on the approach described in Sec. 4.1 to find the best results obtained in this paper.

For the neural models, the weights were initialised with Gaussian samples, each of which has zero mean and standard deviation $\sqrt{\frac{1}{in\_size}}$, where in_size is the dimension of input vectors. The LSTM has 600 hidden nodes, the size of the embedding layer was set to 100, and when a non-linear layer is used between the LSTM and Softmax layers (for the baseline replication), the output dimension was set to 300. These values were set based on the baseline study (Shibata and Heinz, 2017), in which two hidden layer sizes (400 and 600) were used. In Shibata and Heinz (2017, Tables 1 and 2), hidden layer size did not have critical effect on the results. Considering SPiCe'16 datasets and the existing literature, 600 hidden nodes make a large RNN network. We have used two-layer and one-layer LSTM networks in our experiments. The dropout rate was set to 0.5 for all non-recurrent layers, which is known to be close to optimal for a wide range of networks and tasks (Srivastava et al., 2014).

Following the baseline study, for optimisation, we used stochastic gradient descent (SGD) with momentum of 0.9. The learning rate decreased gradually from 0.1 to 0.001, where the number of iterations is 45 and the mini-batch size is 128.

7 Results and Analysis

The main goal of our empirical investigation is to show that correlating the impact of multiple layers in RNN-based neural models with the number of hidden states (quantified using a rank of the SVD) in finite-state automata, we can increase the understanding of the deep neural networks. This way we aim to explain the role of multiple RNN layers in sequence modelling. In this discussion, we will refer to Tab. 1 that reports the scores of the three neural network models described in Sec. 5 and the scores of the WFA models described in Sec. 4.

Table 1: The hyperparameters of WFA, the scores of WFA and neural models, and the score improvement by the best neural model compared to WFA

Data	WFA				Neural Models (NN)			RNN Improv.
#	n	nR	nC	WFA	SPiCe NN	RNN (2 Layers)	RNN (1 Layer)	Gain in score
1	4	5	5	0.8789	0.909	**0.9180**	0.8521	0.0391
2	6	5	5	0.8731	0.920	**0.9210**	0.9183	0.0479
3	5	10	3	0.8248	0.888	**0.8938**	0.8819	0.0690
4	500	5	5	0.5272	**0.619**	0.6131	0.6142	0.0918
5	450	5	5	0.5688	0.8100	**0.8107**	0.7988	0.2419
6	90	6	7	0.8096	0.863	**0.8690**	0.7815	0.0594
7	500	4	4	0.4474	**0.736**	0.7258	0.7176	0.2886
8	60	5	5	0.5426	0.645	**0.6614**	0.6521	0.1188
9	57	8	7	0.9324	0.962	**0.9674**	0.9546	0.0350
10	200	5	5	0.3623	**0.574**	0.5526	0.5604	0.2117
11	100	5	5	0.4147	0.520	**0.5535**	0.5412	0.1388
12	95	4	4	0.8113	0.799	**0.8508**	0.7116	0.0395
13	500	5	5	0.4990	0.592	**0.6007**	0.5357	0.1017
14	2	10	10	**0.4649**	0.350	0.3496	0.3616	-
15	3	6	6	**0.2899**	0.263	0.2655	0.2651	-

Table 1 shows that the proposed one-layer neural network model has achieved competitive results compared with the two-layer stacked network on many of the SPiCe datasets. The additional layer in a two-layer model improved the score most significantly on dataset 12, where the score improved from 0.711 to 0.851 (0.14 units). Another dataset where improvement was observed was dataset 6 where the score improved by 0.088 units. On datasets 1 and 13, the improvements were 0.065 and 0.065 units. In addition to those bigger improvements, a two-layer stacked RNN achieved slight improvement in score (≤ 0.02) on datasets 2, 3, 5, 7, 8, 9, 10, and 11. Still, a one-layer network did better than at least one of the two-layer networks (i.e. SPiCe and the one proposed in this paper) on datasets 4, 8, 10, and 11. Overall, we can see that a one-layer RNN would be a better choice for some of our datasets, although using multiple layers leads to better predictions on other datasets. This means that in order to gain deeper insight into the behaviour of the methods, it is useful to investigate individual datasets in detail and include WFA in our analysis. For this reason, to shed some light on the impact of multiple layers in RNNs, we will analyse datasets 12 and 5 in the subsequent paragraphs. The reason for this choice is that on dataset 12, the score improved significantly using two layers, whereas on dataset 5 a similar improvement was not observed.

Dataset 12 and High Rank The synthetic dataset 12 was the biggest and arguably the most challenging problem in SPiCe 2016. It was initially generated for another competition (PAutomaC) using the PAutomaC data generator (Verwer et al., 2014a). The best performing WFA scored 0.8113 on this dataset with $n = 95$ and $nR = nC = 4$. Although WFA is a Markov model[2] (i.e., a model that may require l-th order representation, which makes predictions based on l the most recent observations, to learn long-range dependencies (Bengio and Frasconi, 1995b; Hinton et al., 2001; Kakade et al., 2016)), on dataset 12, WFA was as good as the RNN models. Our one-layer neural model scored 0.7116 and the two-layer neural model improved the result to 0.8508. So, we can clearly see that on this large dataset, two layers improve the results. We argue that we can use WFA results to explain the improvement of our two-layer neural model. For that we will focus on the rank of WFA (i.e. parameter n) and the maximum length of substrings in its basis (i.e. nR and nC). To score high on dataset 12, WFA had to use 95 hidden states, which is a large number of hidden states for a traditional Baum-Welch algorithm (Siddiqi et al., 2007). This means that

[2]Note that WFA is a generalisation of a Markov model where the next state depends only on the current state (Penagarikano and Bordel, 2004); every Markov model can be represented as a WFA.

in order to solve this problem, a Markov model requires a relatively large number of states. This fact can explain why our two-layer neural model outperformed a single-layer model because the second LSTM layer increased the number of hidden states in the neural model. Moreover, the obtained WFA's score is based on short substrings (i.e. $nR = nC = 4$) in its basis. Therefore, it is fair to expect that dataset 12 does not have long-term dependencies since short substring statistics are sufficient to capture the data-generating distribution in this model. We believe that we can make this claim because our WFA with short substrings works very well on this data. All this means that dataset 12 requires a relatively large number of hidden states, but it does not have long-term dependencies. Our two-layer neural model is sufficient for such problems because two layers increase the number of hidden states, whereas the long-term dependencies are not an issue.

To support our discussion above, we should add that in (Rabusseau et al., 2018) the hidden units of the second-order RNN were shown to be related to the rank or the hidden states of WFA. Note that second-order and higher-order RNNs have their recurrence depth increased by explicit, higher-order multiplicative interactions between the hidden states at previous time steps and input at the current time step. It was shown that any function that can be computed by a linear second-order RNN (Giles et al., 1992) with n hidden units on sequences of one-hot vectors (i.e. canonical basis vectors) can be computed by a WFA with n states. A higher-order RNN has additional connections from multiple previous time steps whereas the classic RNN has connection from one previous time step only. Higher order RNNs allow a direct mapping to a finite-state machine (Giles et al., 1992; Omlin and Giles, 1996). However, a similar connection is not available for classic RNNs and WFA and more importantly for multilayer RNNs and WFA. Based on these theoretical results and our empirical investigation, we can conjecture that the improved score on dataset 12 by using two LSTM layers indicates that the multiple layers helped to model the hidden states more efficiently.

Low Rank To support our arguments about the rank (i.e. the number of hidden states) in our discussion about dataset 12, we can identify a complementary relationship in other results. In particular, when we consider all datasets on which WFA did well having a small rank (this is in contrast to dataset 12 which required a high rank for WFA), a two-layer network does not lead to significant improvement. This pattern can be seen on datasets 1, 2, and 3, and this complements our previous arguments about dataset 12.

Dataset 5 and Long Context Dataset 5 is a real dataset on which the best performing WFA model scored 0.568 with rank $n = 450$ and substring lengths $nR = nC = 5$. This dataset is large for spectral learning and increasing nR and nC above 5 made the method intractable. Our one-layer neural model scored 0.7988 and a two-layer model showed a small improvement scoring 0.8107. This means that on this dataset adding more layers did not change the score significantly. We will attempt to explain the lack of a big improvement of a two-layer neural model using our WFA results.

Dataset 5 corresponds to the NLP character language modelling benchmark from Penn Treebank (Marcus et al., 1993b). The other NLP datasets are 4, 8, 11, and 13. Similar to dataset 5, increasing the number of RNN layers did not significantly improve the score on those datasets. Most NLP data (including dataset 5) have long-term dependencies because there are many training examples of word agreements (with different long-range regularities) which span a large portion of a sentence (Brown and Hinton, 2001). WFA with discrete states have limited memory capacity which gets consumed by having to deal with all the intervening regularities in the sequence. We can clearly see this in our results because in our experiments on WFA, we have many hidden states ($n = 450$). We can see that a large number of hidden states was not sufficient to solve this problem using WFA when $nR = nC = 5$, i.e., when substrings are short. In order to capture long-term dependencies, our WFA would need to be trained on longer substrings (higher nR and nC), but this is infeasible to do on this large dataset because the method becomes intractable. This problem requires the learning algorithm to take care of the long-term context.

We can provide a theoretical justification as to why long substrings (i.e. prefixes and suffixes that define the basis of a Hankel matrix) can lead to a better model given a particular number of hidden states, n. Note that the number of hidden states n corresponds to the number of dimensions that are kept after the SVD of the Henkel matrix. This

means that n most informative latent dimensions (i.e., those that carry the most variance) are used as hidden states. If, given a particular value of n, one model has better performance than another model, it means that it's best n dimensions capture more variance than the best n latent dimensions of the alternative model. This argument explains why one basis of a Hankel matrix can lead to a better model than another basis. Intuitively, it is also natural to expect that long substrings can lead to a better set of hidden states because they can capture longer interactions between input symbols, which should naturally lead to more informative hidden states. If it was computationally feasible to evaluate dataset 5 with larger substring lengths, we could investigate the spectral norm of the empirical Hankel matrix with the increasing length of substrings (i.e. nR, nC). This would shed some light on the quality of the first hidden state in compared models.

Theoretically, a one-layer RNN model can capture infinite context (Siegelmann and Sontag, 1991), but due to training difficulties (e.g. the vanishing gradient problem), the context capture capacity of RNNs is limited. Despite this difficulty, it was shown in the literature that RNNs can capture previous context of up to 200 tokens (Khandelwal et al., 2018). However, other researchers (Pascanu et al., 2014) argue that stacking RNN layers (like in our two-layer model) does not increase the capacity of the model to capture longer contexts. This means that our two-layer model cannot deal with long contexts even when we add more layers. Since WFA did not perform well on dataset 5 having short context and a large number of hidden states, we conjecture that this dataset requires a long context and for this reason two-layers in a neural model do not help. Our results are consistent with other results where long-term contexts are captured by the recurrent layer (Bengio and Frasconi, 1995a). According to the distributed hypothesis (Bengio et al., 2009) stacking multiple layers allows for learning distributed features but not for capturing long-term contexts. Consequently, we assume that the long-term contexts are more important for dataset 5 to make efficient prediction than the pure increase in the number of the hidden states across the space.

Theoretical Considerations The relationship between the number of types of hidden states (discrete, or distributed), long-term dependency and the sequence prediction has been explored by Hinton et al. (2001); Bengio and Frasconi (1995b). For example, the hidden state of a single HMM (a specific version of WFA) can only convey $\log_2 K$ bits of information about the recent history. Instead, if a generative model had a distributed hidden state representation (Williams and Hinton, 1991) consisting of M variables each with K alternative states, it could convey $M \log_2 K$ bits of information. This means that the information bottleneck scales linearly with the number of variables and only logarithmically with the number of alternative states of each variable (Hinton et al., 2001). However, the link between the hidden state modelling and the number of recurrent neural network layers had not been explored before. From this theoretical analysis, we can see that if we have access to a large dataset then increasing the number of layers helps in modelling the hidden states more accurately (as seen in dataset 12), but it does not have to help to capture long-term contexts (dataset 5). In the latter case, one has to use models with high recurrence depth (Zilly et al., 2017; Pascanu et al., 2013), but we leave their exploration for future work since in this paper we wanted to focus on traditional LSTM layers.

Conclusion

Recurrent Neural Networks (RNNs) are a powerful tool for sequence modelling. However, RNNs are non-linear models, which makes them difficult to analyse theoretically. In this paper, we empirically analysed two RNN models (single-layer and two-layer RNNs) to understand the impact of the additional LSTM layers. We used Weighted Finite-state Automata (WFA) trained using the Hankel-based spectral learning algorithm. Based on fifteen benchmark datasets from the SPiCe 2016 competition, our empirical analyses indicate that multiple layers in RNNs help learning distributed hidden states through improved hidden space modelling but have lesser impact on the ability to learn long-term dependencies.

Acknowledgement We thank the four anonymous reviewers for their insightful and professional comments.

References

2010. TYPO CORPUS. http://luululu.com/tweet/. [Online; accessed 05-Feb-2019].

Guy Avni and Orna Kupferman. 2015. Stochastization of weighted automata. In *MFCS (1)*, volume 9234 of *Lecture Notes in Computer Science*, pages 89–102. Springer.

Borja Balle, Xavier Carreras, Franco M Luque, and Ariadna Quattoni. 2014. Spectral learning of weighted automata. *Machine learning*, 96(1-2):33–63.

Borja Balle, Rémi Eyraud, Franco M Luque, Ariadna Quattoni, and Sicco Verwer. 2017. Results of the sequence prediction challenge (spice): a competition on learning the next symbol in a sequence. In *International Conference on Grammatical Inference*, pages 132–136.

Borja Balle, Ariadna Quattoni, and Xavier Carreras. 2012. Local loss optimization in operator models: A new insight into spectral learning. In *ICML*. icml.cc / Omnipress.

Yoshua Bengio and Paolo Frasconi. 1995a. Diffusion of context and credit information in markovian models. *Journal of Artificial Intelligence Research*, 3:249–270.

Yoshua Bengio and Paolo Frasconi. 1995b. Diffusion of credit in markovian models. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 553–560. MIT Press.

Yoshua Bengio et al. 2009. Learning deep architectures for ai. *Foundations and trends in Machine Learning*, 2(1):1–127.

Andrew D Brown and Geoffrey E Hinton. 2001. Products of hidden markov models. In *AISTATS*.

AJ Buchsbaum, Raffaele Giancarlo, and Jeffery R Westbrook. 1998. Shrinking language models by robust approximation. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*, volume 2, pages 685–688. IEEE.

Rafael C Carrasco and José Oncina. 1994. Learning stochastic regular grammars by means of a state merging method. In *International Colloquium on Grammatical Inference*, pages 139–152. Springer.

Olivier Delalleau and Yoshua Bengio. 2011. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674.

Salah El Hihi and Yoshua Bengio. 1996. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pages 493–499.

Jeffrey L. Elman. 1990. Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211.

Robert D Finn, Penelope Coggill, Ruth Y Eberhardt, Sean R Eddy, Jaina Mistry, Alex L Mitchell, Simon C Potter, Marco Punta, Matloob Qureshi, Amaia Sangrador-Vegas, et al. 2015. The pfam protein families database: towards a more sustainable future. *Nucleic acids research*, 44(D1):D279–D285.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Proc. of NIPS*, pages 1019–1027.

C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee. 1992. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Comput.*, 4(3):393–405.

Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Maxout networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1319–III–1327. JMLR.org.

GE Hinton, AD Brown, and Queen Square London. 2001. Training many small hidden markov models. *Proc. of the Workshop on Innovation in Speech Processing*.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Micah Hodosh, Peter Young, and Julia Hockenmaier. 2013. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 47:853–899.

Falk Howar, Malte Isberner, Maik Merten, Bernhard Steffen, Dirk Beyer, and Corina S. Pasareanu. 2014. Rigorous examination of reactive systems - the RERS challenges 2012 and 2013. *STTT*, 16(5):457–464.

Daniel Hsu, Sham M Kakade, and Tong Zhang. 2012. A spectral algorithm for learning hidden markov models. *Journal of Computer and System Sciences*, 78(5):1460–1480.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350.

Sham M. Kakade, Percy Liang, Vatsal Sharan, and Gregory Valiant. 2016. Prediction with a short memory. *CoRR*, abs/1612.02526.

Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. 2018. Sharp nearby, fuzzy far away: How neural language models use context. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 284–294. Association for Computational Linguistics.

Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. 2007. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480. ACM.

Nicolas Le Roux and Yoshua Bengio. 2010. Deep belief networks are compact universal approximators. *Neural computation*, 22(8):2192–2207.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993a. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993b. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Mehryar Mohri. 2004. Weighted finite-state transducer algorithms. an overview. In *Formal Languages and Applications*, pages 551–563. Springer.

Christian W Omlin and C Lee Giles. 1996. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM (JACM)*, 43(6):937–972.

Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2014. How to construct deep recurrent neural networks. *In the Proc. of ICLR*.

Razvan Pascanu, Guido Montufar, and Yoshua Bengio. 2013. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*.

Mikel Penagarikano and German Bordel. 2004. Layered Markov models: a new architectural approach to automatic speech recognition. In *Proceedings of the 2004 14th IEEE Signal Processing Society Workshop Machine Learning for Signal Processing, 2004.*, pages 305–314. IEEE.

Ofir Press and Lior Wolf. 2016. Using the output embedding to improve language models. *In the Proc. of EACL*, 2:157–163.

Guillaume Rabusseau, Tianyu Li, and Doina Precup. 2018. Connecting weighted automata and recurrent neural networks through spectral learning. *arXiv preprint arXiv:1807.01406*.

Jürgen Schmidhuber. 2008. Learning complex, extended sequences using the principle of history compression. *Learning*, 4(2).

Vatsal Sharan, Sham M Kakade, Percy S Liang, and Gregory Valiant. 2017. Learning overcomplete hmms. In *Advances in Neural Information Processing Systems*, pages 940–949.

Chihiro Shibata and Jeffrey Heinz. 2017. Predicting sequential data with lstms augmented with strictly 2-piecewise input vectors. In *International Conference on Grammatical Inference*, pages 137–142.

Sajid M Siddiqi, Geogrey J Gordon, and Andrew W Moore. 2007. Fast state discovery for hmm model selection and learning. In *Artificial Intelligence and Statistics*, pages 492–499.

Hava T. Siegelmann and Eduardo D. Sontag. 1991. Turing computability with neural nets. *Applied Mathematics Letters*, 4:77–80.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Mariona Taulé, Maria Antònia Martí, and Marta Recasens. 2008. Ancora: Multilevel annotated corpora for Catalan and Spanish. In *LREC*. European Language Resources Association.

Sicco Verwer, Rémi Eyraud, and Colin De La Higuera. 2014a. Pautomac: a probabilistic automata and hidden markov models learning competition. *Machine learning*, 96(1-2):129–154.

Sicco Verwer, Rémi Eyraud, and Colin dela Higuera. 2014b. Pautomac: a probabilistic automata and hidden markov models learning competition. *Machine Learning*, 96(1):129–154.

Christopher KI Williams and Geoffrey E Hinton. 1991. Mean field networks that learn to discriminate temporally distorted strings. In *Connectionist Models*, pages 18–22. Elsevier.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2017. Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4189–4198. JMLR. org.