Predicting Sequential Data with LSTMs Augmented with Strictly 2-Piecewise Input Vectors

Chihiro Shibata SHIBATACHH@STF.TEU.AC.JP
School of Computer Science, Tokyo University of Technology

Jeffrey Heinz HEINZ@UDEL.EDU
Dept. of Linguistics and Cognitive Science, University of Delaware

Abstract

Recurrent neural networks such as Long-Short Term Memory (LSTM) are often used to learn from various kinds of time-series data, especially those that involved long-distance dependencies. We introduce a vector representation for the Strictly 2-Piecewise (SP-2) formal languages, which encode certain kinds of long-distance dependencies using subsequences. These vectors are added to the LSTM architecture as an additional input. Through experiments with the problems in the SPiCe dataset (Balle Pigem et al., 2016), we demonstrate that for certain problems, these vectors slightly—but significantly—improve the top-5 score (normalized discounted cumulative gain) as well as the accuracy as compared to the LSTM architecture without the SP-2 input vector. These results are also compared to an LSTM architecture with an input vector based on bigrams.

1. Introduction

In recent years, so-called deep learning technology has developed dramatically and has brought innovations to various areas related to machine learning such as image recognition and natural language processing (LeCun et al., 2015). Recurrent neural networks (RNN) are believed to be the most effective models which learn from time-series data. Among the various types of RNNs, Long Short-Term Memory (Hochreiter and Schmidhuber, 1997) (LSTM) is one of the most popular. They are carefully constructed so that the model can capture long-distance dependencies in sentences or among sequential elements.

Although it is known that sophisticated RNN models like LSTM and other variants (Jozefowicz et al., 2015) are effective for processing time-series data, what happens inside of the learned RNNs is still unclear. It is difficult to understand how they successfully capture long-term dependencies and when they fail to do so.

This paper applied the LSTM architecture to the sequence prediction tasks in the SPiCe competition (Balle Pigem et al., 2016). The object of the competition was to predict the next element of a sequence. The competition scored algorithms on their performance on both real and synthetic data.

This paper reports the results of three LSTMs applied to this data in this competition. The basic model was a multilayered LSTM. The second and third models use the same multilayered LSTM and also included input vectors corresponding to the states of particular deterministic finite-state automata (DFA) which also processed the data. The second model uses a vector embedded with the current state of a Strictly 2-Piecewise DFA

(Heinz and Rogers, 2010). The third model uses a vector embedding the state of a Strictly 2-Local DFA (bigram model) (McNaughton and Papert, 1971).

While all LSTMs performed well, the second model with the SP-2 vector performs slightly but significantly better on some problems and not significantly worse on others. On the other hand, while the third model with the bigram vector also performed significantly better on some problems, it performed significantly worse on others.

2. Basic model: Two-layered LSTM

In this section, we describe the basic architecture common to all three models, including some trivial data preprocessing. Additional details such as the vector length and the size of each layer are described in Section 5.

The basic model we use is a two-layered LSTM network with another non-linear layer on top. Figure 1 shows this structure. Those layers were placed on top of the input vector which embedded each symbol $a(t)$ in the sequence as it appeared at time t. The output layer implements the softmax function, which outputs the network's prediction of the next element of the sequence, $a(t+1)$. Between the softmax layer and the LSTMS is a fully connected layer with a Rectified Linear (ReL) activation function.

A 'start' symbol and an 'end' symbol were added to both sides of each training sentence. Symbols are fed to the model from the start symbol.

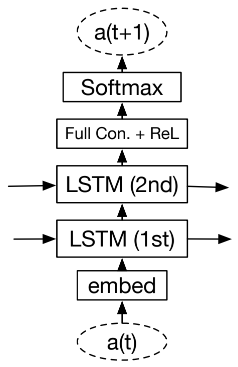

Figure 1: Basic structure: two-layered LSTM for next-element prediction.

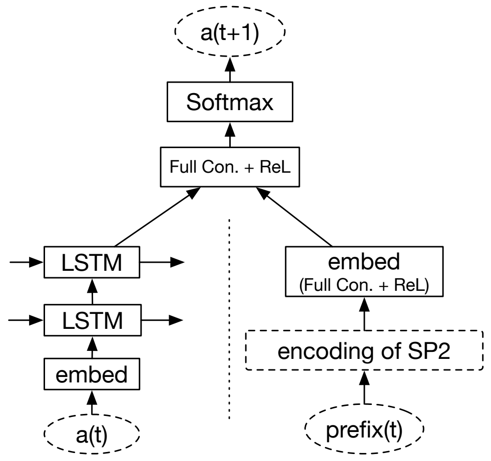

Figure 2: SP-2 model: the LSTM model combined with SP-2 embedding.

3. Strictly Piecewise model

SP-k formal languages describe certain types of long-term dependencies (Heinz and Rogers, 2010). Essentially, they describe long-distance dependencies obtained by forbidding subsequences whose length is bounded by k. For instance, if the ab subsequence is forbidden, then no b may occur after a no matter how much earlier in the word a occurs. SP-k languages can also be characterized by a DFA whose states encode the subsequences of size $k-1$ present in the prefixes leading to each state. They can also be more compactly represented by a set of DFAs operating in parallel (Heinz and Rogers, 2013). These languages (and DFA) are closely related to the well-studied Piecewise Testable languages (Simon, 1975).

In this paper, we introduce a new method which encodes the states of a SP-2 automata into a zero-one vector, which then is fed as additional input to the network. The idea is that explicit representation of the subsequence information can help capture long-term dependencies that may be present in the data.

The vector encoding for SP-2 DFA is straightforward: for an alphabet Σ, the size of the vector is $|\Sigma|$ and each coordinate corresponds to one symbol in the alphabet. Figure 3 shows examples of the SP-2 input vector over some time. As Figure 3 shows, the states of

t	a(t)	prefix(t)	SP-2 input vector				
			[a	b	c	d	e]
0	a	a	[1	0	0	0	0]
1	d	ad	[1	0	0	1	0]
2	a	ada	[1	0	0	1	0]
3	c	adac	[1	0	1	1	0]
4	d	adacd	[1	0	1	1	0]

Figure 3: Examples of the SP-2 input vector over time for an example sequence with $\Sigma = \{a, b, c, d, e\}$.

SP-2 automata are identified from the prefix, and then encoded into a zero-one vector. For SP-2, each coordinate of the zero-one vector represents whether the corresponding symbol previously occurred. Generally, for SP-k, each coordinate of the vector represents the prior occurrence of some subsequence of size $k-1$ (and so the size of the vector would be $|\Sigma|^{k-1}$).

While some sophisticated RNNs like LSTMs are carefully constructed to be able to remember information that may have occurred arbitrarily far in the past, it is still not well-understood what kind of long-distance dependencies RNNs can learn. One of our main goals in this paper is to reveal the effects of explicit encoding of subsequence information as part of the input. As discussed later, the positive impact of including the SP-2 input vector suggests that when it comes to long-distance dependencies, there is room to improve both the performance and our understanding of RNNs.

Figure 2 shows the network architecture when the SP-2 vector is added as input.

To embed the SP-2 zero-one vector into a real-valued multi-dimensional vector, it is fed into a fully connected layer with a non-linear activation function such as ReL. Simultaneously, the input vector that represents $a(t)$ is put into the two-layered LSTM as in the basic model. Finally, we concatenate the output of these layers and the output of the fully connected layer embedding the SP-2 one-zero vector and put them into another fully connected layer layer which feeds finally into the softmax layer.

4. Alternately-taken Bigram Model

In addition to the SP-2 model, we conducted experiments with another network architecture. Generally, the above strategy can be used with different types of input vectors representing different types of information about the characters or words in the prefix. The n-gram model, for example, is a common one used in practical tasks in natural language processing such as sentence generation (Jurafsky and Martin, 2008). It was felt that comparing the SP-2 model with a vector based on bigrams would be interesting. Also, bigram models are essentially stochastic versions of Strictly 2-Local formal languages, which are similar to the Strictly 2-Piecewise languages in that substrings—as opposed to subsequences—are forbidden (Rogers et al., 2013).

For SPiCe problems, since the number of alphabet is less than fifty for almost all problems, we use bigrams as input vectors in addition to unigrams so that bigram patterns are more explicitly taken into the network. We divide a sentence $(a_0 a_1 a_2 a_3 \cdots)$ into two sequences consist of the even bigrams $(a_{01} a_{23} \cdots)$ and the odd bigrams $(a_{12} a_{34} \cdots)$. Although these are fed into the LSTMs separately, the LSTMs themselves are required to have common weights. In this way, the number of samples for learning the weights is doubled. Figure 4 shows the structure of our bigram-based model.

5. The Experiments

We used a package for neural networks called Chainer (`http://chainer.org`) for the implementation. The source code is available at `https://github.com/cshib/rnns_for_spice`.

We let the size of element-embedding, SP-2-embedding, and bigram-embedding vectors be 100 through all experiments. In addition, we take the same vector size for all intermediate layers, such as layers of LSTMs, and the intermediate fully connected layers. The only exception is the the layer before the output layer (softmax), which was half size of other vectors. For each of the three models, we experimented with two vector sizes: 400 and 600 (so the penultimate layer had size 200 and 300, respectively).

Since there are three kinds of network structures and two kinds of vector sizes as described above, there are six combinations of network structures in all.

We used the momentum stochastic gradient descent (SGD) with the momentum 0.9 for the optimization. The stepsize decreased gradually from 0.1 to 0.001, where the number of iterations is 45. We used dropout against the last two layers to prevent overfitting.

Tables 1 and 2 show the top-5 scores for each problem in the SPiCe challenge. While the results of the bigram model are mostly worse than the other models, the differences between the SP-2 model and the basic model are slight. To determine whether the differences were significant, we executed the models about ten times. Among instances obtained from multiple executions, the worst 30% of them in the sense of training error (not test error) were abandoned.

Standard errors are shown in parentheses in Table 1 and 2. The bold numbers show the best scores with the significance level of 5%. The Bigram model clearly decreases the scores in many problems except for Problems 4 and 7, while it improves the scores for those two problems significantly. For the SP-2 model, we can see that the improvements are significant for Problems 1, 6 and 12, while the differences are not significant for the others.

6. Conclusion

For the SPiCe competition, we ran three different network structures with two sizes of vectors, and chose the best one for the test data. Overall, the experiments here show that the SP-2 model was the best one. For some types of data, the explicit representation of subsequence information in the SP-2 network architecture significantly improves prediction accuracy, while for other types of data, it makes no significant differences. On the other hand, while the bigram model improves the accuracy for some problems, it significantly worsens the result in many others.

We believe the narrow advantage of the SP-2 model is due to its explicit representation of long-term dependencies in terms of subsequences. Obviously, careful inspection of the underlying mechanisms which generated the data in the relevant problems is required to evaluate this hypothesis. More generally, we believe different types of formal languages can shed light on the different kinds of long-term dependencies that different types of RNNs can and cannot learn.

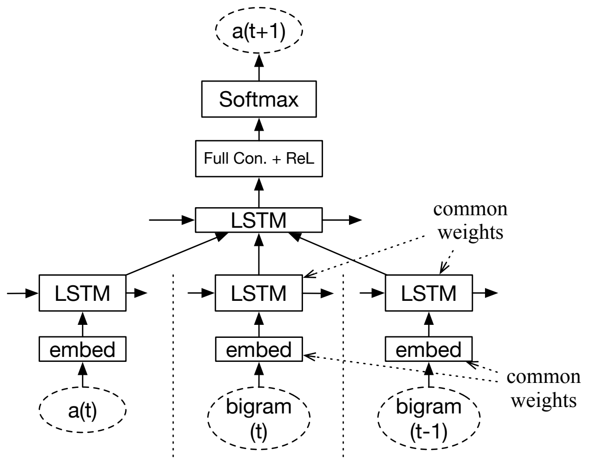

Figure 4: Bigram model: the combination of LSTMs with the unigrams, the odd bigrams and the even.

Acknowledgments

We would like to thank the members of the SPiCe organization team for everything about the competition. The first author is grateful to Rémi Eyraud for kindly responding to queries during the competition.

The implementation, experiments and participation for the SPiCe competition were done by the first author. The idea of SP-2 vector representation is the result of both authors' joint work. Both authors contributed to writing the paper.

This work was partially supported by JSPS KAKENHI 26780123 to the first author and by NIH#R01HD087133 to the second author.

Table 1: comparison of scores with the dim. 400

	simple(400)	sp2(400)	bigram(400)
1	0.906(0.002)	**0.915**(0.000)	0.836(0.009)
2	0.920(0.000)	0.919(0.000)	0.878(0.003)
3	0.884(0.001)	0.884(0.001)	0.848(0.001)
4	0.615(0.001)	0.614(0.002)	**0.633**(0.002)
6	0.848(0.001)	**0.855**(0.001)	0.836(0.001)
7	0.717(0.001)	0.718(0.000)	**0.730**(0.001)
8	0.646(0.001)	0.646(0.001)	0.630(0.001)
9	0.959(0.001)	0.960(0.000)	0.958(0.000)
10	0.575(0.001)	0.577(0.001)	0.569(0.001)
11	0.529(0.001)	0.527(0.001)	–
12	0.782(0.002)	**0.796**(0.000)	0.727(0.003)
13	0.588(0.001)	0.588(0.001)	0.578(0.001)
14	0.344(0.001)	0.346(0.001)	0.332(0.001)
15	0.264(0.001)	0.265(0.001)	0.261(0.000)

Table 2: comparison of scores with the dim. 600

	simple(600)	sp2(600)	bigram(600)
1	0.909(0.002)	**0.915**(0.000)	0.769(0.003)
2	0.920(0.000)	0.920(0.000)	0.838(0.004)
3	0.888(0.001)	0.886(0.001)	0.831(0.001)
4	0.619(0.002)	0.616(0.002)	**0.634**(0.001)
6	0.863(0.001)	**0.867**(0.001)	0.828(0.002)
7	0.736(0.000)	0.736(0.001)	**0.747**(0.001)
8	0.645(0.001)	0.644(0.001)	0.614(0.001)
9	0.962(0.000)	0.962(0.000)	0.959(0.000)
10	0.574(0.001)	0.573(0.001)	0.570(0.002)
11	0.520(0.001)	0.519(0.001)	–
12	0.799(0.002)	**0.807**(0.001)	0.713(0.001)
13	0.592(0.001)	0.590(0.001)	0.581(0.000)
14	0.350(0.002)	0.351(0.002)	0.333(0.002)
15	0.263(0.001)	0.263(0.001)	0.258(0.001)

References

Borja Balle Pigem, Remi Eyraud, Franco M. Luque, Ariadna Quattoni, and Sicco Verwer. Results of the sequence prediction challenge (SPiCe): a competition about learning the next symbol in a sequence. In *Proceedings of the 13th ICGI*, page to appear, 2016.

Jeffrey Heinz and James Rogers. Estimating strictly piecewise distributions. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 886–896, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

Jeffrey Heinz and James Rogers. Learning subregular classes of languages with factored deterministic automata. In Andras Kornai and Marco Kuhlmann, editors, *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, pages 64–71, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

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

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In David Blei and Francis Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350. JMLR Workshop and Conference Proceedings, 2015.

Daniel Jurafsky and James Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics.* Prentice-Hall, Upper Saddle River, NJ, 2nd edition, 2008.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Robert McNaughton and Seymour Papert. *Counter-Free Automata.* MIT Press, 1971.

James Rogers, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. Cognitive and sub-regular complexity. In Glyn Morrill and Mark-Jan Nederhof, editors, *Formal Grammar*, volume 8036 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222. 1975.