MANCHESTER
1824

The University of Manchester

A comparison of LSTM and GRU networks for learning symbolic sequences

Cahuantzi, Roberto and Chen, Xinye and Güttel, Stefan

2021

MIMS EPrint: **2021.9**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

A comparison of LSTM and GRU networks for learning symbolic sequences*

Roberto Cahuantzi[1 [0000−0002−0212−6825]], Xinye Chen[1 [0000−0003−1778−393X]], and Stefan Güttel[1 [0000−0003−1494−4478]]

The University of Manchester, Department of Mathematics, Manchester, M13 9PL, United Kingdom

Abstract. We explore relations between the hyper-parameters of a recurrent neural network (RNN) and the complexity of string sequences it is able to memorize. We compare long short-term memory (LSTM) networks and gated recurrent units (GRUs). We find that an increase of RNN depth does not necessarily result in better memorization capability when the training time is constrained. Our results also indicate that the learning rate and the number of units per layer are among the most important hyper-parameters to be tuned. Generally, GRUs outperform LSTM networks on low complexity sequences while on high complexity sequences LSTMs perform better.

Keywords: recurrent neural network · LSTM · GRU · sequence learning

1 Introduction

Reliable and inexpensive methods to forecast trends and mining the patterns in time series are in high demand and a lot of efforts have gone into developing sophisticated models. Deep learning models are among the more recently employed approaches; see e.g. [16] who compare a sophisticated hybrid neural network model to simpler network models and more traditional statistical methods (such as hidden Markov models) for trend prediction, with the hybrid model achieving the best results. Another hybrid forecasting method which combines recurrent neural networks (RNNs) and exponential smoothing is discussed in [22]. A forecasting method based on an adaptation of the deep convolutional WaveNet architecture is presented in [3]. An interpretable deep learning time series prediction framework is proposed in [19]. Comparisons of LSTM and GRU networks on numerical time series data tasks can be found in [25]. Deep RNNs have also been applied to time series classification; see e.g. [9, 17].

Despite these significant advances in the development of new time series models, there is also a growing literature suggesting that pre-processing the data is just as important to forecasting or classification performance as are model improvements. In this realm, [20] show that discretisation transformations can

* R. C.'s work has been funded by the UK's Alan Turing Institute. S. G. has been supported by a Fellowship of the Alan Turing Institute, EPSRC grant EP/N510129/1.

improve the forecasting performance of neural network models. Another critical aspect are the metrics to assess the forecasting or classification performance of models. The Euclidean distance metric and its variants, such as the mean squared error, are often used in this context. However, these metrics can be sensitive to noise in the data, an effect that becomes even more pronounced with time series of high dimensionality. Hence, [6, 15] argue that symbolic time series representations, which naturally offer dimensionality reduction and smoothing, are useful tools to allow for the use of discrete (i.e. symbolic) modeling.

Here we use experiments to gain insights into the connections between the hyper-parameters of popular RNNs and the complexity of the string sequences to be learned (and forecasted). This study is partly inspired by [7] who evaluate the performance of many variants of LSTM cells via extensive tests with three benchmark problems, all rather different from our string learning task. The Python code used to perform our experiments is publicly available[1]. Among our main findings are that: (1) the learning rate is one of the most influential parameters when training RNNs to memorize sequences (with values near 10^{-2} found to be the best in our setup in terms of training time and forecast accuracy); (2) for the tasks considered here it is often sufficient to use just common RNNs with a single layer and a moderate number of units (such as around 100 units); (3) GRUs outperform LSTM networks on low complexity sequences while on high complexity sequences the order is reversed.

Note that a common approach to use deep learning for time series are global forecasting models (GFMs) which are trained on large groups of time series; see e.g. [2, 11, 18]. While this approach is very attractive due to the improved generalizability of the resulting models, reduced proneness to overfitting, and potentially lower overall training time, the model complexity is significantly higher and the selection of hyper-parameters even more involved. Here we take a different, simpler, approach by training one model for each string sequence. This will provide a more direct insight into the learning capability of a single RNN dependent on the complexity of the string sequences it is meant to learn. As a GFM is expected to be at least as complex as the model required to learn the most complex sequence in a group of time series, we believe that our study also sheds light on some parameter choices for GFMs.

2 Methodology

Our approach to quantify the learning capabilities of RNNs is to generate string sequences of different complexities (with complexity measured in terms of the compressibility of the string), to train RNNs on a part of that string until a predefined stopping criterion is reached, and then to quantify the accuracy of the forecast of the following string characters in an appropriate text similarity metric. Below we provide details for each of these steps.

[1] `https://github.com/robcah/RNNExploration4SymbolicTS`

2.1 String generation and LZW complexity

As training and test data for this study we produce a collection of strings with quantifiable complexities. These strings are here-forth referred to as *seed strings*. A Python library was written to generate these seed strings, allowing the user to choose the target complexity and the number of distinct symbols to be used.

One way to quantify complexity is due to Kolmogorov [14]: the length of the shortest possible description of the string in some fixed universal language without losing information. For example, a string with a thousand characters simply repeating `"ab"` can be described succinctly as `500*"ab"`, while a string of the same length with its characters chosen at random does not have a compressed representation; therefore the latter string would be considered more complex.

A more practical approach to estimate complexity uses lossless compression methods [12, 26]. The Lempel–Ziv–Welch (LZW) compression [23] is widely recognised as an approximation to Kolmogorov complexity. The LZW algorithm serves as the basis of our complexity metric as it is very easy to implement and can be adapted to generate strings of a target compression rate. The LZW algorithm creates a dictionary of substrings and an array of dictionary keys from which the original string can be fully recovered. We define the *LZW complexity* of a seed string as the length of its associated LZW array, an upper bound on the Kolmogorov complexity.

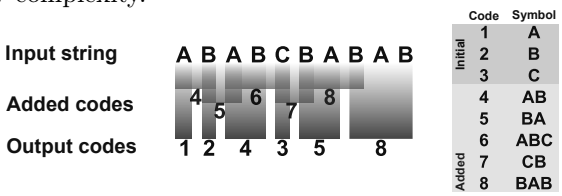

Fig. 1. An illustration of LZW compression. Assume that we have an alphabet of three symbols (A, B, C) and the seed string `"ABABCBABAB"`. As the LZW algorithm traverses the seed string from left to right, a dictionary of substrings is built (table on the right). If a combination of characters already contained in the dictionary is found, the related index substitutes the matching substring. In this case the resulting array is $[1, 2, 4, 3, 5, 8]$ corresponding to an LZW complexity of 6.

2.2 Training, test, and validation data

The data used to train and evaluate the RNN models is obtained by repeating each seed string until a string s of predefined minimal string length is reached. The trailing n characters of s are split off to form a validation string v. The remaining leading characters are traversed with a sliding window of n characters to produce input and output arrays X and y, respectively, for the training and testing. Here, the input array X is of dimension $m \times n \times p$, where m stands for number of input sequences ($m = |s| - 2n$ where s denotes the length of s), n is the length of each input sequence, and p is the dimension of the binary vectors used for the one-hot encoding of each of the distinct characters. The output array y contains the next symbol following each string sequence encoded in X and is of

dimension $m \times p$. The pair (X, y) is split 95% vs 5% to produce the training and test data, respectively. (This rather low fraction of test data is justified as there occur repeated pairs (X, y) in the data due to the repetitions in the string s.) The test data is used to compute the RNN accuracy and loss function values. Finally, the one-hot encoding of the validation string v results in an array of dimension $n \times p$. The trained RNN model is then used to forecast the validation string v, and a text similarity measure quantifies the forecast accuracy.

To exemplify this we can imagine a seed string `"abc"` with $p = 3$ distinct characters, which will be repeated to reach a string s of at least 100 characters length. In this case, $s =$`"abcabcabc...abc"` is of length 102 characters. The trailing $n = 10$ characters are split off for the validation, resulting in $v =$`"cabcabcabc"`. The remaining 92 leading characters of s are then traversed with a sliding window of width $n = 10$ to form the input-output data pairs (X, y) as follows:

```
(abcabcabca,b) (bcabcabcab,c) (cabcabcabc,a) ... (abcabcabca,b)
```

The one-hot encoding $a = [1, 0, 0]$, $b = [0, 1, 0]$, $c = [0, 0, 1]$ results in the final arrays used for the training and testing.

2.3 Recurrent Neural Networks

We consider two types of RNNs based on long short-term memory (LSTM) cells [8] and Gated Recurrent Units (GRUs) [5], respectively. Different versions of these units exist in the literature, so we briefly summarize the ones used here.

A standard LSTM cell includes three *gates*: the forget gate f_t which determines how much of the previous data to forget; the input gate i_t which evaluates the information to be written into the cell memory; and the output gate o_t which decides how to calculate the output from the current information:

$$\begin{aligned} i_t &= \sigma(W_i\ X_t + R_i\ h_{t-1} + b_i) \\ f_t &= \sigma(W_f\ X_t + R_f\ h_{t-1} + b_f) \\ o_t &= \sigma(W_o\ X_t + R_o\ h_{t-1} + b_o). \end{aligned} \tag{1}$$

Here, the W, R and b variables represent the matrices and vectors of trainable parameters. The LSTM unit is defined by

$$\begin{aligned} \dot{C}_t &= \tanh(W_c\ X_t + R_c\ h_{t-1} + b_c) \\ C_t &= f_t \odot C_{t-1} + i_t \odot \dot{C}_t \\ h_t &= o_t \odot \tanh(C_t) \\ y_t &= \sigma(W_y\ h_t + b_y). \end{aligned} \tag{2}$$

In words, the candidate cell state $\dot{C}_t$ is calculated using the input data X_t and the previous hidden state h_{t-1}. The cell memory or current cell state C_t is calculated using the forget gate f_t, the previous cell state C_{t-1}, the input gate i_t and the candidate cell state $\dot{C}_t$. The Hadamard product $\odot$ is simply the element-wise

product of the involved matrices. The output y_t is calculated by applying the corresponding weights (W_y and b_y) to the hidden state h_t.

GRUs are similar to LSTMs but use fewer parameters and only two gates: the update (u_t) and reset (r_t) gates. The gate u_t tunes the update speed of the hidden state while the gate r_t decides how much of the past information to forget by resetting parts of the memory [1]. The GRU unit is defined by the below set of equations. In them $\dot{h}_t$ stands for the candidate hidden state.

$$\begin{aligned} u_t &= \sigma(W_u\ x_t + R_u\ h_{t-1} + b_u) \\ r_t &= \sigma(W_r\ x_t + R_r\ h_{t-1} + b_r) \\ \dot{h}_t &= \tanh(W_h\ x_t + (r_t \odot h_{t-1})R_h + b_h) \\ h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot \dot{h}_t \\ y_t &= \sigma(W_y\ h_t + b_y) \end{aligned} \tag{3}$$

2.4 Text similarity metrics

Due to the non-Euclidean nature of symbolic representations the accuracy of the forecast is best quantified via text edit metrics such as the Damerau–Levenshtein (DL) and Jaro–Winkler (JW) distance. The DL distance counts the number of edit steps required to transform a string into another [4]. The JW distance is a more elaborate metric which is less sensitive to string insertions and changes in character positions; see [24]. Our metrics for the string forecast accuracy are the normalised versions of the DL and JW distances computed using the Python library `textdistance`[2]. The text similarity in this version gives a value of 1.0 for identical strings and a value of 0.0 for "completely different" strings.

3 Results

The following computational tests are performed on a Dell PowerEdge R740 Server with 1.5 TB RAM and two Intel Xeon Silver 4114 processors running at 2.2 GHz. The scripts were written and run in Python 3.7.3 using the libraries Pandas 1.2.3, NumPy 1.19.2, TensorFlow 2.4.1, and TextDistance 4.2.0. In order to reduce the number of parameter configurations to be studied we have divided our tests into three parts. The first initial parameter study on medium-complexity seed strings will be used to fix the number of layers, decide on the stopping criterion for the training, and reduce the number of learning rates considered. The other two tests explore the remaining parameters with seed strings of low and high complexity, respectively.

3.1 Initial parameter test with medium complexity seed strings

We start with an initial parameter study to set the basis for the following in-depth tests. For this test, 12 seed strings were generated using 2, 5, 10 and 20

[2] textdistance 4.2.0, `https://pypi.org/project/textdistance`

symbols; with LZW complexities of 20, 35 and 50. Each of the these seed strings was repeated to produce strings of at least 500 characters length. The trailing 100 characters of each of these strings are used as the validation data, while the other leading characters are used for the training. For each training string, an RNN is trained with different stopping criteria, learning rates, number of layers, and units per layer. Each configuration is trained five times to reduce the effect of the random weight initialisation.

The Adam optimizer [13] is used, motivated by the results of [21] who showed that adaptive learning-rate methods, and in particular Adam, yields the best results for sparse data such as one-hot encoded sequences. The learning rates are varied between $\{0.001, 0.01, 0.1\}$. The maximal number of training epochs is set to 999. Two stopping criteria are evaluated: (i) stop the training when the accuracy reaches a value larger or equal to 0.99, and (ii) stop when the loss function, in this case categorical cross entropy, reaches a value less or equal to 0.1. While the loss function is well known, it is worth to mention that the aforementioned accuracy is calculated by computing the frequency in which the predicted values match the real y values and dividing it by the total predictions, in this case the total elements of y.

After the training is completed, a forecast of 100 characters is produced and its text similarity to the validation string is measured. For both stopping criteria we found that a learning rate of 0.01 led to the smallest training times for all string complexities considered. This is summarized visually in Figure 2.

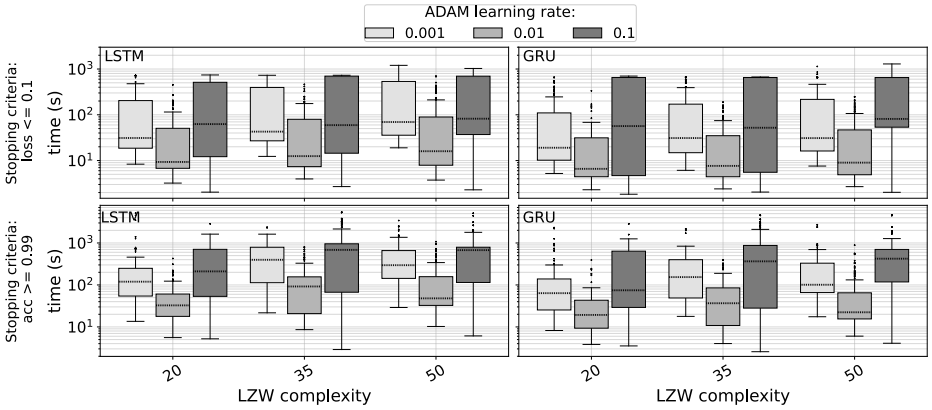

Fig. 2. Total time needed for training LSTM and GRU networks on strings of different LZW complexities and with different stopping criteria and learning rates. The dotted line shows the median, the box represents the interquartile range (IQR, the middle 50%), the whiskers have a length of 1.5·IQR. All points outside the whiskers are considered outliers and are plotted individually. Note the logarithmic scale of the y-axis. A learning rate of 0.01 appears most suitable irrespective of the stopping criterion.

We next explore the string memorization capability of the networks dependent on the number of layers. We train LSTM and GRU networks with $\ell \in \{1, 2, 3\}$ layers and each layer having u units, where u is chosen such that

ℓu is closest to $\{50, 100, 200\}$ (i.e., the total number of units is approximately constant as ℓ varies). The quality of the forecasts measured using DL distances is averaged over all networks with the same number of layers and over all 12 seed strings. In all cases, the loss-based stopping criterion is used and the learning rate is 0.01. The results are shown in Figure 3. The most successful network configuration, in terms of small DL distance and training time, is a single hidden layer network (the results look similar for the JW distance). Although there is a slight improvement in forecast accuracy with each added hidden layer, the observed increase in training time does not seem to justify their addition.

In summary, this initial parameter test trained 3,239 RNNs for the 12 different seed strings, over 5 runs to prevent outlier's biasing, and with the variety of parameters discussed above. A main finding is that the learning rate and the number of hidden units are among the most influential hyper-parameters for the effectiveness of the considered RNNs. This is consistent with findings in [7]. In what follows, we will use single-layer RNNs with a reduced range of considered learning rates and perform larger studies with strings of lower and higher LZW complexities, respectively.

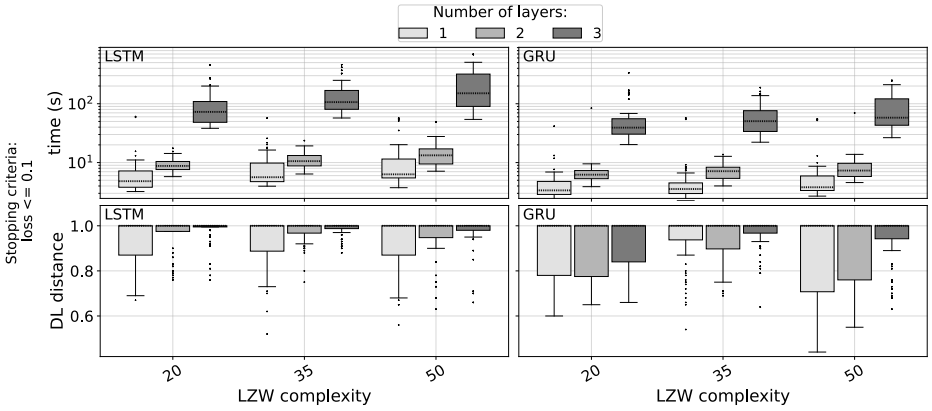

Fig. 3. Studying the dependency on the number of RNN layers, always using the same loss-based stopping criterion and a learning rate of 0.01. Note the logarithmic y-axes on the top. The addition of layers slightly increases accuracy in both DL and JW text similarities (here only DL is shown for simplicity) but the significant increase in training time makes it hard to justify the depth increase.

3.2 Test with seed strings of low complexity

In this low LZW complexity exploration nearly 3,600 RNNs are trained for a total of 37 different seed strings. These seed strings are now repeated to produce sequences of a minimum length of 1,100 characters. Again, the trailing 100 characters are used as validation strings, with the remaining leading strings used for the training and testing. The seed strings have LZW complexities ranging between 2 and 12 and are composed of a number of distinct symbols ranging

between 2 and 6. A single hidden layer is used for both the LSTM and GRU networks. The number of units within the hidden layer is varied between 25 and 250, in ten geometrically-spaced steps. The Adam optimizer is used with learning rates of 0.001 and 0.01 and the aforementioned loss-based stopping criterion. As before, all configurations are run 5 times and averaged.

A visual summary of the results is given in Figure 4. The median training time for all tests with LSTM networks is 37.19 seconds with an interquartile range (IQR) between 15.64 to 75.79 seconds, and for GRU 19.72 seconds with an IQR between 8.48 to 31.70 seconds. We generally find that GRUs are trained faster than LSTM networks to achieve the same loss function value with the same optimizer over all considered learning rates and network complexities, not only in median values but also with less dispersion in general. The forecast accuracy

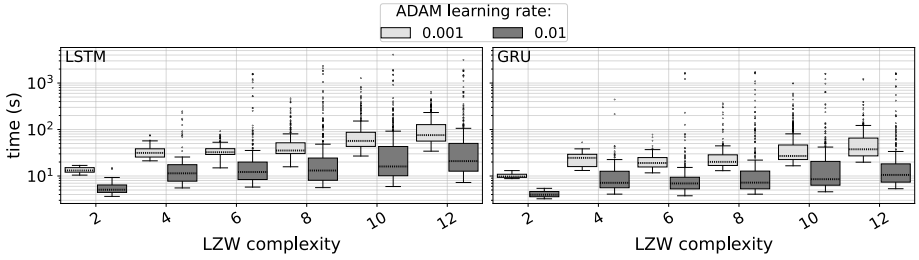

Fig. 4. Training timewhen fitting low complexity seed strings. On average, GRU requires about half the training time compared to LSTM.

with LSTMs and GRUs are comparable. The median distance of both JW and DL metrics for both types of RNNs was 1.0, this is the same value for the third quartile (Q_3) however some differences are appreciated in the first quartile (Q_1). The values for LSTM are 0.93 and 0.97 for DL and JW distance respectively, whereas for GRU, they are 0.88 for DL and 0.96 for JW. This tells us that despite the longer training times LSTM seems to have a small advantage on accuracy. This information is presented visually in Figure 5. One must remember that these text similarities are not Euclidean and small differences for JW usually correspond to more contrasting strings than DL.

3.3 Test with seed strings of high complexity

Our final study uses a total of 300 seed strings with 10, 33 or 52 symbols and LZW complexities ranging between 1,000 to 1,850 (168 linearly spaced steps between these bounds). The seed strings are all at most 2,400 characters long and then repeated to produce string sequences of 5,000, 7,500 and 10,000 characters, respectively. A total number of 4,500 RNNs is trained for this test. We experienced some stagnation in the training of GRUs which was easily fixed by changing the learning rate from 0.01 to 0.0035, while for LSTM the learning rate

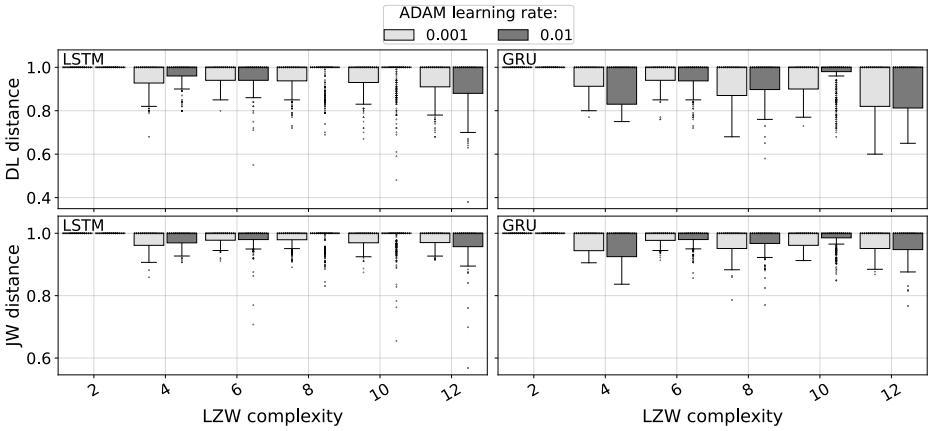

Fig. 5. LSTM and GRU achieve similar forecast performance for low complexity seed strings, with LSTM sightly better but requiring more training time.

0.01 was kept. The stopping criterion, number of units and number of layers are fixed to loss-based, 100 and 1, respectively.

We find that LSTMs are better suited than GRUs for high complexity strings: the median training time is 12.53 seconds for LSTMs and almost double, namely 22.84 seconds, for GRUs, with an IQR between 10.59 and 15.07 and between 18.07 and 29.57 for LSTM and GRU, respectively; see Figure 6. To simplify the plots, all 168 complexities were combined into 8 bins. Note that the data dispersion decreases drastically after the binned complexity of 1,600, which is caused by the larger number of seed strings using 52 symbols. The median and IQR values for both types of RNNs were found to be 1.0; see Figure 7.

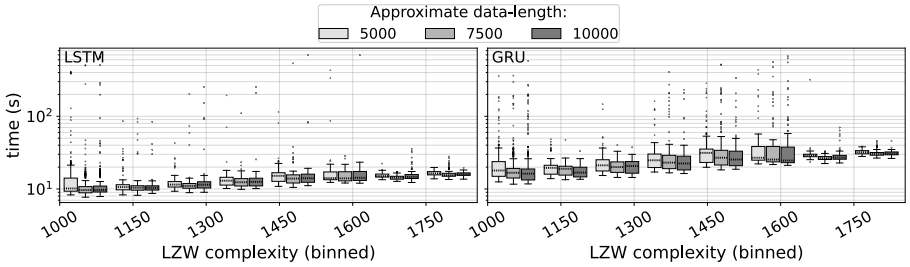

Fig. 6. Results for high complexity seed strings. Now LSTMs are faster to train than GRUs for a similar forecast performance.

4 Discussion

We have used string sequences of quantifiable complexity to gain insights into hyper-parameter choices for two of the most common RNNs. We found that the

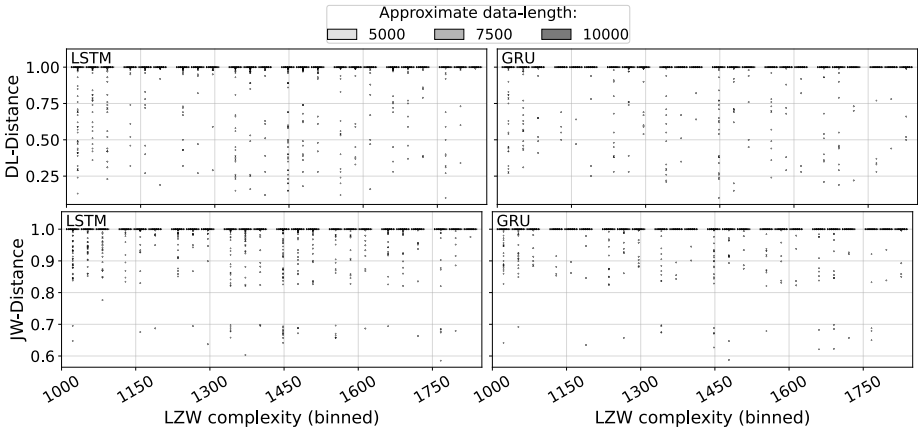

Fig. 7. Results for high complexity seed strings. LSTM and GRU achieve similar forecast accuracy in all cases (but LSTMs are faster to train; see Figure 6).

learning rate is a crucial parameter for the efficient training and that an increase in RNN depth leads to significant increase in training time but not necessarily forecast accuracy. GRUs outperformed LSTMs for low complexity strings while LSTMs performed better on high complex strings. The latter finding is consistent with experiences in language modelling (typically involving very complex strings), where LSTM were also found to perform better than GRUs [10].

In all our tests the networks have been able to learn all sequences to relatively high accuracy. This need not be the case however: if the complexity of a string becomes very large, the network's learning capability might be exceeded. This is manifested by an observed decrease in the mean values of text similarity and an increase in outlier scattering. A demonstration of this is shown in Figure 8.

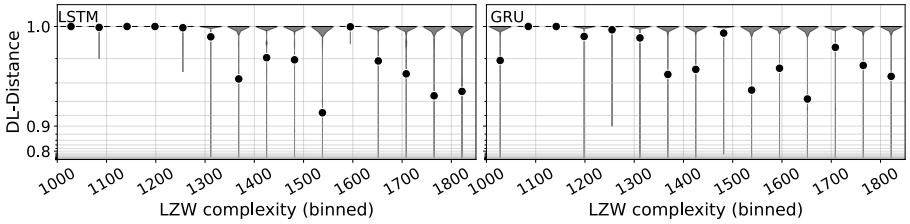

Fig. 8. Saturation of learning capacity as the string complexity increases. Only seed strings of 52 symbols were considered in this test. The gray-shaded areas represent KDEs with bandwidth 0.01, the markers represent the mean. The decreasing trend was observed for both LSTM and GRU with optimized hyper-parameters, trained for a maximum of 999 epochs. Note the exponential scale of the x-axis to emphasize the decrease in mean text similarity, suggesting a degradation of the forecasting quality.

Bibliography

[1] Aggarwal, C.C.: Neural Networks and Deep Learning. Springer (2018)

[2] Bandara, K., Bergmeir, C., Smyl, S.: Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. Expert Systems with Applications **140**, 112896 (2020)

[3] Borovykh, A., Bohte, S., Oosterlee, C.W.: Dilated convolutional neural networks for time series forecasting. Journal of Computational Finance **22**, 73–101 (2018)

[4] Boytsov, L.: Indexing methods for approximate dictionary searching: Comparative analysis. ACM Journal of Experimental Algorithmics **16**, 1.10–1.91 (2011)

[5] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. pp. 1724–1734 (2014)

[6] Elsworth, S., Güttel, S.: Time series forecasting using LSTM networks: A symbolic approach. arXiv 2003.05672 (2020)

[7] Greff, K., Srivastava, R.K., Koutnik, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: A search space odyssey. IEEE Transactions on Neural Networks and Learning Systems **28**, 2222–2232 (2017)

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

[9] Hüsken, M., Stagge, P.: Recurrent neural networks for time series classification. Neurocomputing **50**, 223–235 (2003)

[10] Irie, K., Tüske, Z., Alkhouli, T., Schlüter, R., Ney, H.: LSTM, GRU, highway and a bit of attention: An empirical overview for language modeling in speech recognition. In: Interspeech. pp. 3519–3523 (2016)

[11] Januschowski, T., Gasthaus, J., Wang, Y., Salinas, D., Flunkert, V., Bohlke-Schneider, M., Callot, L.: Criteria for classifying forecasting methods. International Journal of Forecasting **36**, 167–177 (2020)

[12] Kaspar, F., Schuster, H.G.: Easily calculable measure for the complexity of spatiotemporal patterns. Physical Review A **36**, 842–848 (1987)

[13] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (2015)

[14] Kolmogorov, A.N.: On tables of random numbers. Sankhyā: The Indian Journal of Statistics, Series A **25**, 369–376 (1963)

[15] Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. pp. 2–11 (2003)

[16] Lin, T., Guo, T., Aberer, K.: Hybrid neural networks for learning the trend in time series. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence. pp. 2273–2279 (2017)
[17] Malhotra, P., TV, V., Vig, L., Agarwal, P., Shroff, G.: TimeNet: Pre-trained deep recurrent neural network for time series classification. In: Proceedings of 25th European Symposium on Artificial Neural Networks (2017)
[18] Montero-Manso, P., Hyndman, R.J.: Principles and algorithms for forecasting groups of time series: Locality and globality. Tech. rep., Monash University, Department of Econometrics and Business Statistics (2020)
[19] Oreshkin, B.N., Carpov, D., Chapados, N., Bengio, Y.: N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In: International Conference on Learning Representations (2020)
[20] Rabanser, S., Januschowski, T., Flunkert, V., Salinas, D., Gasthaus, J.: The effectiveness of discretization in forecasting: An empirical study on neural time series models. arXiv 2005.10111 (2020)
[21] Ruder, S.: An overview of gradient descent optimization algorithms. arXiv 1609.04747 (2016)
[22] Smyl, S.: A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. International Journal of Forecasting **36**, 75–85 (2020)
[23] Welch, T.: A technique for high-performance data compression. Computer **17**, 8–19 (1984)
[24] Winkler, W.E.: Overview of record linkage and current research directions. Tech. rep., Bureau of the Census (2006)
[25] Yamak, P.T., Yujian, L., Gadosey, P.K.: A comparison between ARIMA, LSTM, and GRU for time series forecasting. In: Proceedings of the 2nd International Conference on Algorithms, Computing and Artificial Intelligence. pp. 49–55. ACM (2019)
[26] Zenil, H.: A review of methods for estimating algorithmic complexity: Options, challenges, and new directions. Entropy **22**, 1–28 (2020)