Is Timing Critical to Trace Reconstruction?

Javier Perez Tobia
Department of Computer Science
University of British Columbia
BC, Canada
javipt21@mail.ubc.ca

Apurva Narayan
Department of Computer Science
School of Engineering
University of British Columbia
BC, Canada
apurva.narayan@ubc.ca

***Abstract*—Dynamic analysis of real-world software systems is challenging due to imperfections, noise and data loss. Moreover, these systems evolve with time and their requirements are usually either not clearly specified or unknown, which makes it hard to analyze them. Therefore, it is important to create models that can learn to behave similarly to these systems to enable us to predict their actions, recover missing data, or detect potential failures ahead of time.**

Several models have been proposed to model sequential data, but the vast majority of them only have a qualitative notion of time or no notion of it at all. In this paper, we extend the work on incorporating a quantitative notion of time to RNN and introduce Time GRU. This modified GRU can learn the behaviour of complex software systems to a very high degree of accuracy. Our approach is scalable and has shown state-of-the-art performance on industry-strength software with real operating logs from Blackberry's QNX real-time operating system. The proposed model can predict upcoming sequences of events more than 100 timesteps ahead in time with more than 90% accuracy. This allows for significant improvement in trace reconstruction and failure explainability.

I. Introduction

Deep Learning is undoubtedly one of the most useful tools of the last decade. Despite the wonderful results deep learning can achieve on synthetic data, all of that work is not useful if it cannot be generalized to solve real-life problems. One of the application areas of deep learning is utilizing data from software systems to not only understand but also predict their behaviour. Such models help in various tasks such as anomaly detection, runtime monitoring, automated reasoning, etc. in complex software systems. This is crucial for systems that are deployed in safety-critical environments such as aircrafts, autonomous vehicles, health devices, etc, in which the slightest malfunction of the system software can lead to catastrophic outcomes. To protect systems against these imperfections there are two main approaches: using noise removal techniques to preprocess real-world data, and using sequential deep learning algorithms that can adapt to imperfections to model real systems.

Noise removal is an active area of research of paramount importance in the sequence analysis domain. Most of the effort however is devoted towards noise removal techniques for numerical sequences. From the most common filters like the mean, median or Gaussian filters, to more specialized filters like the Savitzky–Golay filter [1] or wavelet denoising [2] which have been used to help in the modelling of real life systems like cloud services [3], they are all aimed at removing noise from numerical sequences. While there have been some categorical denoising methods proposed [4] [5], their performance has not been widely studied. For this reason, we did not incorporate noise removal as part of this study and we identify it as a possible future research avenue.

Data from real-time operating systems is usually obtained in the form of trace logs or sequences of events. Therefore, it is extremely important for models to be able to interpret sequential inputs and to learn the relationships between different events. This is usually done passively by learning and mining the qualitative temporal dependencies between different events. Among all the proposed architecture possibilities, Recurrent Neural Networks (RNN) are the standard approach across industry and academia to handle sequential data. RNN cells act like loops in which previous activations are used as inputs in future steps. This enables them to store information about past states of the model and use it to make better predictions than by just using information about the most recent state. Their performance on a wide variety of domains like natural language processing [6], [7], stock market recommendation systems [8], [9] or even computer vision [10], proves why recurrent architectures are one of the best kinds of models for sequential-data modelling.

Many different approaches have been proposed to improve modelling performance on sequential problems, e.g., combining Temporal Convolutional networks and Long short-term memory (LSTM) to model network traffic [11], combining Bi-directional LSTM with Grid LSTM to predict workload requirements of cloud services [12]. Sucholutsky et.al. [13] propose a trade restoration algorithm combining LSTMs with a Timed-Regular Expression Mining (TREM) framework [14], [15] or QRE mining frameworks [16]. Despite their successes, almost all these methods focus on combining RNNs in innovative ways but not in altering how RNNs work on a basic level. Standard RNN architectures interpret traces as ordered sets of events equally spaced in time. Only their order is significant. This limits their applicability for trace analysis tasks in safety-critical systems because, in the real world, events are not just ordered sets of events but they also have continuous-time dependencies between them. Hence, having a quantitative notion of time is critical to the safe and reliable operation of these systems. Most of the

sequential models proposed so far lack in modelling such timing characteristics [**?**], [17]. We see this as a weakness of most RNN architectures and we believe that incorporating a quantitative notion of time in sequential models is important to be able to reconstruct missing traces of events or predict potential failures successfully.

Blackberry's real-time operating system (RTOS) QNX is one of the most widely used software systems in the safety-critical systems domain. QNX is used in ventilators, train controls, medical robots and over 175 million vehicles on the road today. It is a microkernel system with a Unix-like architecture. It comes with an excellent Tracelogger that records several features of every state or event of the system. These traces provide developers with a lot of information about how software is running and a possibility to restore missing data or study malfunctions [18] [19]. This paper is a case study of the performance of several RNN models with a quantitative notion of time on the task of predicting upcoming traces of events from BlackBerry's QNX system.

Our contributions are:

1) A novel approach for trace restoration using Time LSTM and Time Gated Recurrent Units (GRU).
2) An empirical study of the necessity of continuous-time information for improved performance of trace restoration models.

II. Related Work

Previous work makes evident the difficulty of reconstructing event traces from real-life systems like QNX. One attempt to restore missing data and detect anomalies uses a Continuous Bag of Words (CBOW) Word2Vec [20] model [18]. Even though this model can predict single QNX events from previous traces, its accuracy decreases rapidly as the predicted sequence size increases. A possible reason for this limited performance is that CBOW fails to learn the continuous-time dependencies between events, which are a crucial element of real-life systems. For safety-critical systems like QNX, it is important to make predictions several steps ahead into the future to detect and prevent any possible malfunctions as well as to reconstruct missing data. Therefore, improving the performance of trace reconstruction techniques for this system is still an open problem with only a partial solution.

Recurrent Neural Networks are one of the best approaches to model sequential data. LSTM networks were one of the first types of RNN to be proposed [21]. Since then, they have become one of the preferred architectures for problems with sequential data due to their success in tasks like machine translation [22] or speech recognition [23]. What makes LSTM cells perform so well on sequential data is that, on top of an $input$ and an $output$ gate like a standard RNN, they have a $memory$ and a $forget$ gate. The architecture is shown in Figure 1. The $memory$ gate learns and stores dependencies between events that are far apart in time. However, it is the $forget$ gate that makes this architecture so brilliant as it allows the LSTM to update its memory to

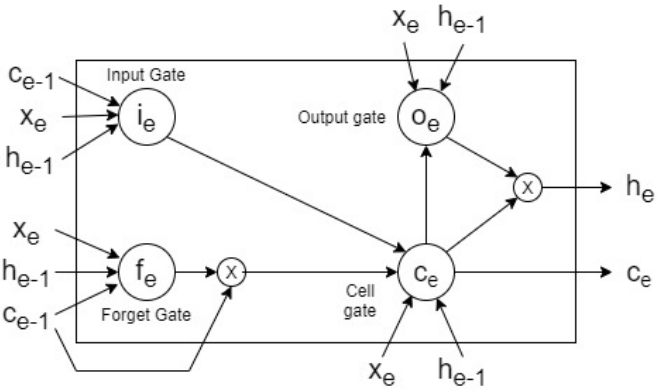

Fig. 1. LSTM architecture

adapt to new changes in the sequences. With this architecture, LSTM can choose to omit information from very far in the past and pay more attention to more recent events as indicated by the activations of the inner state of the cells. GRU networks were proposed in 2014 [23] as a simpler alternative to LSTM. GRUs only have three gates: an $update$ gate, which controls what is learnt and what is omitted; a $reset$ gate, which controls what information is retrieved for predictions, and a $memory$ gate. Figure 2 shows the architecture of a standard GRU cell. Despite the different architectures, both GRU and LSTM obtain similar results on a wide range of tasks and their relative performance usually depends on the application [24]–[26].

As already mentioned, incorporating time into RNN is crucial if we want to model time-sensitive sequential data accurately. Bailer-Jones et. al. [27], propose adding time intervals as a weighing factor that controls the influence of the previous prediction on the current prediction. Their work is aimed at modelling real-life temporal processes sampled at discrete-time intervals. In their work, they claim that their discrete-time RNN can be applied to real-life problems; however, they only verify its performance on noise-free synthetic data generated from a simulation of a dynamical system with only two input variables, namely a decaying harmonic oscillator. Many real-world systems are more complex and so the performance of models on them needs to be empirically evaluated with real data.

Another attempt to incorporate time into RNN, proposed by M. C. Mozer et.al., are continuous-time GRU (CT-GRU) and CT-RNN [28]. CT-GRU stores information about events in each unit with multiple memory traces for different time scales. Additionally, their model has time-dependent decay rates that target specific time scales to exploit the different continuous dependencies and enable the model to better assess the influence of past events to make predictions. In spite of its promising architecture, the model is unsuccessful at beating the performance of standard LSTM and GRU. This shows that incorporating time into RNN is not a trivial problem so if we want to improve their performance, new ideas of how to give models a quantitative notion of time

new ideas need to be proposed.

The best RNN architecture to use time explicitly in its architecture is Time LSTM, which has been used before to create better recommendation systems [29]. Y. Zhu et.al. suggest that, even though LSTM is already good at modelling short and long-term dependencies in sequential data, its performance can be improved by using time explicitly to control the influence of previous events on future predictions. To do so, they propose incorporating additional $time$ gates to the LSTM architecture that, along with the $input$ gate, filter the $cell$ state of the network and put more or less focus on information from previous states. While their results on recommendation systems show that their model is effective, Y. Zhu et.al. suggest the verification of their work on other fields and types of systems.

In this paper, we continue the work of Zhu et.al [29] with another RNN, a Time GRU and verify its performance, and that of Time LSTM, on the reconstruction of real traces of a safety-critical system. We also evaluate the necessity of continuous-time data for these kinds of systems and improve upon the work of Lakhani et. al. [18] on predicting upcoming sequences of real data from QNX collected using its tracelogger utility.

III. Time Gated Recurrent Unit

The architecture of a standard GRU is:

$$r_e = \sigma_r(x_e W_{xr} + h_{e-1} W_{hr} + b_r) \quad (1)$$

$$z_e = \sigma_u(x_e W_{xz} + h_{e-1} W_{hz} + b_z) \quad (2)$$

$$\tilde{h}_e = tanh(x_e W_{x\tilde{h}} + r_e \odot h_{e-1} W_{h\tilde{h}} + b_{\tilde{h}}) \quad (3)$$

$$h_e = (1 - z_e) \odot \tilde{h}_e + z_e \odot h_{e-1} \quad (4)$$

Where x_e, h_{e-1} and h_e correspond to the current event and the last prediction and the current prediction of the network respectively; r_e, z_e, $\tilde{h}_e$ are the $reset$, $update$ and $memory$ gates; σ_r, σ_z and are a sigmoid function and $tanh$ the hyperbolic tangent function; W_{xr}, W_{hr}, W_{xz}, W_{hz}, $W_{x\tilde{h}}$, $W_{h\tilde{h}}$, b_r, b_z, $b_{\tilde{h}}$ are the weight and bias parameters of the network. The prediction layer has two parts added together: the previous state of the network $\tilde{h}_e$ controlled by the update vector z_e through element-wise multiplication ($\odot$), and the other is the information from the previous state h_{e-1} controlled by the update vector z_e through element-wise multiplication. This is shown in Figure 2.

As done for Time LSTM 1 [29] we add a single time gate T_e. The proposed Time GRU modifies a standard GRU cell as:

$$r_e = \sigma_r(x_e W_{xr} + h_{e-1} W_{hr} + b_r) \quad (5)$$

$$z_e = \sigma_u(x_e W_{xz} + h_{e-1} W_{hz} + b_z) \quad (6)$$

$$\mathbf{T_e} = \sigma_{\mathbf{t}}(\mathbf{x_e W_{xt}} + \sigma_{\mathbf{\Delta t}}(\mathbf{\Delta t_e W_{tt}}) + \mathbf{b_t}) \quad (7)$$

$$\tilde{h}_e = \mathbf{T}_e \odot tanh(x_e W_{x\tilde{h}} + r_e \odot h_{e-1} W_{h\tilde{h}} + b_{\tilde{h}}) \quad (8)$$

$$h_e = (1 - z_e) \odot \tilde{h}_e + z_e \odot h_{e-1} \quad (9)$$

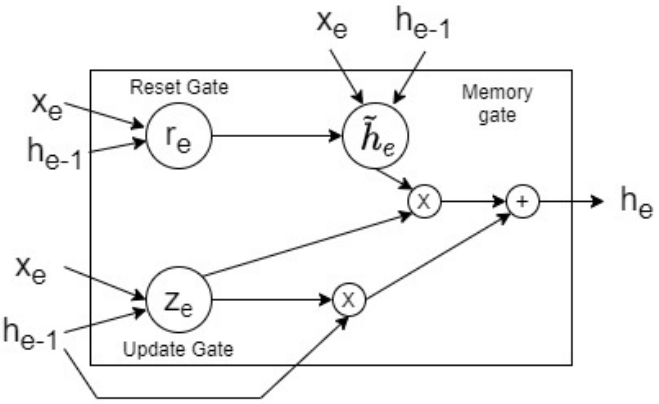

Fig. 2. GRU architecture

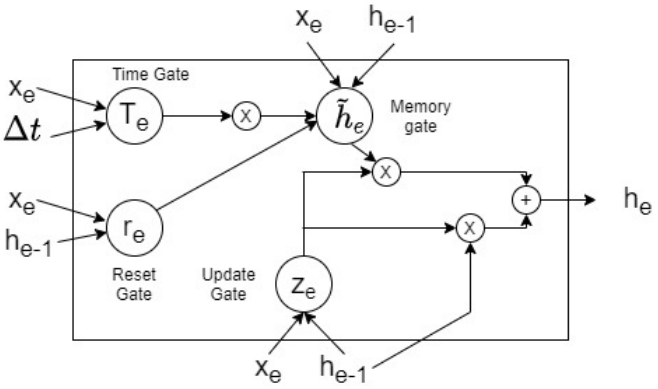

Fig. 3. Time GRU architecture

Where Δt_e corresponds to the time interval between the current event and the previous event and T_e is the $time$ gate. This architecture is shown in Figure 3. This modified GRU can not only learn from the order of traces of events but it can also acquire a quantitative notion of time that can be used to control the influence that previous states of the network have on future predictions.

IV. Methodology

The dataset used in this study is generated from real data from the real-time operating system QNX embedded in a car. The dataset consisted of a list of events of 93 different classes and their timestamps in nanoseconds collected over several minutes of operation. Using absolute times was not useful because they depend on the start time, which can change between different logs of data. The time intervals between events were used instead, which made it easier to work with time and proved to be more useful for the model.

We approach the system trace reconstruction problem as a sequence prediction problem. The goal is to be able to predict sequences of upcoming events, given previous sequences of events as inputs. After experimenting with a few different architectures, we found that good results could be achieved by stacking two RNN units and a linear layer after them to

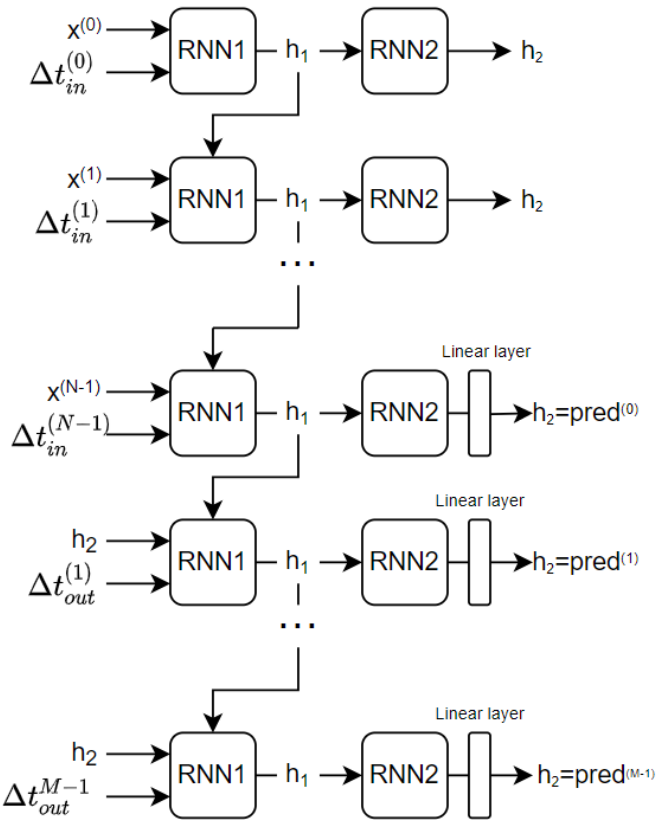

Fig. 4. Timed RNN workflow with N input events and M output events.

make predictions. The size of the hidden state of the LSTMs was set to be equal to the number of classes. Figure 4 shows the workflow of our model. Given a sequence of inputs x of size N, each element of the input, $x^{(i)}$ and the time interval since the last event $\Delta t^{(i)}$ is passed through the first RNN. The output state of the first RNN, h_1 is used as the input state for the first RNN for the next event. When the network arrives at the last event of the input sequence, it passes the output state of the second RNN h_2 through a linear layer to make a new prediction $pred^{(i)}$. This prediction is then used as the input event for the next RNN1. The prediction finishes when a selected number of M events have been predicted. The non-timed RNNs used in this study are built similarly but they do not take time intervals as inputs.

V. Experiments

This section is divided into three experiments: a comparison of different types of RNNs on the task of event trace prediction; a comparison between the use of discrete-time data and continuous-time data; and a study of the generalization from the trained input/output sequence sizes to different input/output sequence sizes.

All experiments were performed in a Python environment with a single P100 GPU, one Intel Xeon CPU with 2.30GHz and 12.5GB of RAM. All models were generated using PyTorch. The code for this paper is available at: https://github.com/Idsl-group/Time-RNNs.

A. Time GRU vs other RNNs

We compared the performance of Time GRU against a Markov chain, used as a benchmark, and four other RNNs: a standard LSTM, a standard GRU and two Time LSTMs [29] on the same task: predicting future events from a given input trace of events. In order to minimize bias, all the networks have a similar architecture with two RNN cells as shown in Figure 4. Due to the high variance of time intervals, time intervals were encoded as one-hot vectors before being passed through the Time RNNs to help the models learn more from them. All models were trained for 80 epochs to predict output sequences of 50 and 100 events from input sequences of only 25 events. Table I shows the results of this experiment.

Model	Output trace size			
	50 events		100 events	
	Acc.(%)	κ	Acc.(%)	κ
Markov	16.29	0.0522	14.32	0.0261
LSTM	50.38	0.4688	36.41	0.3087
GRU	47.19	0.4345	33.63	0.2765
Time LSTM 1	88.02	0.8727	87.13	0.8632
Time LSTM 2	88.10	0.8736	87.66	0.8688
Time GRU	86.58	0.8572	85.83	0.8493

TABLE I
Training accuracy (Acc.) and Cohen's Kappa coefficient (κ) [30] after 80 epochs of training

Time GRU is able to outperform by far the Markov Chain as well as the standard LSTM and standard GRU. This shows that, as suggested by Y. Zhu et. al [29], the Time RNN architecture can be easily generalized to other RNN architectures. Not even when trained for up to 200 epochs could LSTM or GRU surpass their timed counterparts. While RNNs perform great in some domains, they cannot get past 50% accuracy when trying to model the QNX system. This shows the importance and the necessity for recurrent models to have a quantitative notion of time. The three Time RNNs achieve accuracies above 85% even when predicting sequences four times as large as the input sequence. Although in this case Time LSTM 1 and Time LSTM 2 perform slightly better than Time GRU, their performance is similar and, as is the case for the standard RNNs [24], Time GRU is comparable to Time LSTM. We additionally show that the Kappa coefficient [30] of each test is high and very close to the accuracy, which ensures that the reported accuracies are not biased due to imbalances in the dataset.

During training, we noticed an unexpected difference in convergence speeds between the Time RNNs (see Figure 5). Both standard LSTM and standard GRU converge at a similar rate. However, both Time LSTMs start converging

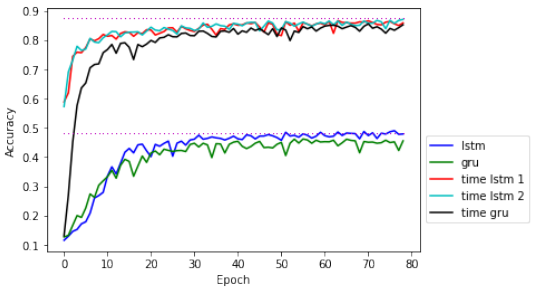

Fig. 5. Convergence speed of the 5 RNNs

much more rapidly before reaching a region of slower convergence while Time GRU converges more slowly but does not plateau as much and keeps improving in later training steps. While we have not found an explanation for this unexpected behaviour, it motivated the comparison of the accuracies of all the models when trained for the same amount of time.

It is known that GRU cells are faster than LSTM cells because they have fewer parameters. We evaluated the training times of the Time RNNs and found that: Time GRU is 25% faster than Time LSTM 1 and 43% faster than Time LSTM 2. When given the same time to train, Time GRU achieves a much more similar accuracy to both Time LSTMs. Table II compares the performance of the 5 recurrent networks when trained for just 2 hours. Under this constraint, Time GRU and both Time LSTMs achieve a much more similar around of 84% when predicting output sequences of 100 events. This serves as further evidence of the similarity between Time GRU and Time LSTM. It is evident that in safety-critical environments one should choose whichever model can maximize accuracy for that specific setting. However, in other settings where maximizing accuracy is not critical and optimizing resources and evaluation speed is, Time GRU proves as a great alternative to Time LSTM as it can achieve similar results with the same training time and fewer parameters.

Model	Output trace size			
	50 events		100 events	
	Acc.(%)	κ	Acc.(%)	κ
Markov	16.29	0.0522	14.32	0.0261
LSTM	48.53	0.4493	26.84	0.1873
GRU	45.36	0.4134	28.80	0.2158
Time LSTM 1	86.01	0.8542	84.97	0.8460
Time LSTM 2	85.88	0.8498	84.53	0.8354
Time GRU	84.90	0.8400	84.37	0.8337

TABLE II
TRAINING ACCURACY (ACC.) AND COHEN'S KAPPA COEFFICIENT (κ) AFTER 2 HOURS OF TRAINING

B. Discrete-time vs continuous-time representations

In the previous section, time intervals were discretized by encoding them as one-hot vectors because of the high variance and different orders of magnitude of the time intervals. Our intuition suggested that increasing the dimensionality of time intervals would help to better model the different time scales and would improve the accuracy. However, we found that this was not the best approach and that it was better to preserve time continuity. Keeping time intervals as continuous vectors of size one helped the model learn more about the time dependencies of the system. This time, we trained the model to predict sequences of 50 events from input sequences of 25 events for only 50 epochs using continuous-time vectors of size 1. This resulted in Time GRU with an accuracy of 93% with a Kappa coefficient of 0.91, which is considerably better than the best result from the previous section, 88%, achieved by Time LSTM 2. This shows that, on top of it being important to incorporate a quantitative notion of time into RNN models in safety-critical environments, it is also important to maintain time continuity. For evident reasons, we used a continuous-time representation in the next experiment as well.

C. Time GRU generalization to different sequence sizes

Lastly, we evaluated the generalization ability of Time GRU to different input/output sequence size combinations than what it had been trained on. That is, we wanted to find if Time GRU would maintain a similar behaviour to the system independent of how many previous events were known. The same Time GRU used in the previous section was trained on input traces of 25 events to predict future traces of 25 events for 80 epochs. Then, this network was evaluated on different input/output trace size combinations. As Table III shows, the model maintained accuracies above 90% and Kappa coefficients higher than 0.89 for all the input/output combinations, even when the output sequence size was 20 times larger than the input sequence size. This shows that Time GRU can learn the behaviour of the system and the time dependencies between its actions to a high degree of accuracy. The model is not just an event sequence predictor. In a sense, it can be thought of as a simpler representation of the system that can be used to verify and recover event traces from very few previous events.

VI. CONCLUSION

We have shown the usefulness of incorporating a quantitative notion of time in Recurrent Neural Networks a new real-life setting, trace reconstruction of the real time operating system QNX. We have continued the work on Time RNN using three different models, two Time LSTM from [29] and the proposed Time GRU. From our experiments we extract three main conclusions: Firstly, we show that Time GRU performs is comparable in performance to Time LSTM 1 or Time LSTM 2 on the task of trace reconstruction. Secondly, we found that models that take continuous representations of time as inputs, significantly outperform those who take discrete representations of time or even no time information

Results of trained model

Input size	Output size	Acc.	κ
25	25	92.54	0.9210

Validation of trained model

Input size	Output size	Acc.	κ
25	50	91.92	0.9144
25	100	91.18	0.9062
10	100	90.52	0.8993
10	200	89.58	0.8896
50	25	92.24	0.9178
100	25	92.13	0.9167

TABLE III
ACCURACY(ACC.) AND COHEN'S KAPPA COEFFICIENT(κ) OF DIFFERENT INPUT/OUTPUTS SIZE COMBINATIONS AFTER TRAINING WITH INPUT AND OUTPUT SIZE OF 25 EVENTS FOR 80 EPOCHS.

at all. This should serve as evidence of the essential need for sequential models of real-life systems to take time information into account. Lastly, we showed that a continuous-time GRU can learn the behaviour of the system to a very high degree of accuracy and generalize to different input/output size sequences.

Using Time GRU for system trace reconstruction of the QNX system, we have significantly outperformed previous attempts by being able to predict sequences of events 10 times larger than input sequences with an accuracy of 91% and a similarly high Kappa coefficient. Our results suggest that Time GRU (and consequently other Time RNN models) can become a very close representation of a real-life system like QNX. In future work, we want to extract LTL properties from this RNN and build a finite state machine representation of the system.

REFERENCES

[1] A. Savitzky and M. J. Golay, "Smoothing and differentiation of data by simplified least squares procedures." *Analytical chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[2] M. Lang, H. Guo, J. E. Odegard, C. S. Burrus, and R. O. Wells, "Noise reduction using an undecimated discrete wavelet transform," *IEEE Signal Processing Letters*, vol. 3, no. 1, pp. 10–12, 1996.

[3] J. Bi, H. Yuan, and M. Zhou, "Temporal prediction of multiapplication consolidated workloads in distributed clouds," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 4, pp. 1763–1773, 2019.

[4] R. Gupta and R. N. Rao, "Towards semantic noise cleansing of categorical data based on semantic infusion," *arXiv preprint arXiv:2002.02238*, 2020.

[5] F. Nijweide, "Autoencoder-based cleaning of non-categorical data in probabilistic databases," B.S. thesis, University of Twente, 2020.

[6] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[7] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and rnn for natural language processing," *arXiv preprint arXiv:1702.01923*, 2017.

[8] M. Nabipour, P. Nayyeri, H. Jabani, A. Mosavi, E. Salwana *et al.*, "Deep learning for stock market prediction," *Entropy*, vol. 22, no. 8, p. 840, 2020.

[9] L. Qi, M. Khushi, and J. Poon, "Event-driven lstm for forex price prediction," *arXiv preprint arXiv:2102.01499*, 2021.

[10] C. Zhao, J. G. Han, and X. Xu, "Cnn and rnn based neural networks for action recognition," in *Journal of Physics: Conference Series*, vol. 1087, no. 6. IOP Publishing, 2018, p. 062013.

[11] J. Bi, X. Zhang, H. Yuan, J. Zhang, and M. Zhou, "A hybrid prediction method for realistic network traffic with temporal convolutional network and lstm," *IEEE Transactions on Automation Science and Engineering*, 2021.

[12] S. Li, J. Bi, H. Yuan, M. Zhou, and J. Zhang, "Improved lstm-based prediction method for highly variable workload and resources in clouds," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2020, pp. 1206–1211.

[13] I. Sucholutsky, A. Narayan, M. Schonlau, and S. Fischmeister, "Deep learning for system trace restoration," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[14] L. Schmidt, A. Narayan, and S. Fischmeister, "Trem: A tool for mining timed regular specifications from system traces," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 901–906.

[15] A. Narayan, G. Cutulenco, Y. Joshi, and S. Fischmeister, "Mining timed regular specifications from system traces," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 2, pp. 1–21, 2018.

[16] P. K. Mahato and A. Narayan, "Qmine: A framework for mining quantitative regular expressions from system traces," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2020, pp. 370–377.

[17] P. Chatigny, J.-M. Patenaude, and S. Wang, "Spatiotemporal adaptive neural network for long-term forecasting of financial time series," *International Journal of Approximate Reasoning*, vol. 132, pp. 70–85, 2021.

[18] K. Lakhani and A. Narayan, "A neural word embedding approach to system trace reconstruction," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 2019, pp. 285–291.

[19] A. Narayan and S. Fischmeister, "Mining time for timed regular specifications," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 2019, pp. 63–69.

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215*, 2014.

[23] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 4520–4524.

[24] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[25] R. Perumal and T. L. van Zyl, "Comparison of recurrent neural network architectures for wildfire spread modelling," in *2020 International SAUPEC/RobMech/PRASA Conference*. IEEE, 2020, pp. 1–6.

[26] S. Mangal, P. Joshi, and R. Modak, "Lstm vs. gru vs. bidirectional rnn for script generation," *arXiv preprint arXiv:1908.04332*, 2019.

[27] C. A. Bailer-Jones, D. J. MacKay, and P. J. Withers, "A recurrent neural network for modelling dynamical systems," *network: computation in neural systems*, vol. 9, no. 4, pp. 531–547, 1998.

[28] M. C. Mozer, D. Kazakov, and R. V. Lindsey, "Discrete event, continuous time rnns," *arXiv preprint arXiv:1710.04110*, 2017.

[29] Y. Zhu, H. Li, Y. Liao, B. Wang, Z. Guan, H. Liu, and D. Cai, "What to do next: Modeling user behaviors by time-lstm." in *IJCAI*, vol. 17, 2017, pp. 3602–3608.

[30] A. Ben-David, "Comparison of classification accuracy using cohen's weighted kappa," *Expert Systems with Applications*, vol. 34, no. 2, pp. 825–832, 2008.