RECURRENT NEURAL NETWORK ARCHITECTURE BASED ON DYNAMIC SYSTEMS THEORY FOR DATA DRIVEN MODELLING OF COMPLEX PHYSICAL SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

While dynamic systems can be modelled as sequence-to-sequence tasks by deep learning using different network architectures like DNN, CNN, RNNs or neural ODEs, the resulting models often provide poor understanding of the underlying system properties. We propose a new recurrent network architecture, the Dynamic Recurrent Network (DYRNN), where the computation function is based on the discrete difference equations of basic linear system transfer functions known from dynamic system identification. This results in a more explainable model, since the learnt weights can provide insight on a system's time dependent behaviour. It also introduces the sequences' sampling rate as an additional model parameter, which can be leveraged, for example, for time series data augmentation and model robustness checks. The network is trained using traditional gradient descent optimization and can be used in combination with other state of the art neural network layers. We show that our new layer type yields better results than other recurrent layer types on several system identification tasks.

1 INTRODUCTION

Dynamic systems occur in many different areas of life (Isermann & Münchhof (2011)). From biology, engineering, medicine to economics and more: Often, if a system changes its state based on a external input, this system can be viewed as a dynamic system. Dynamic system identification is the process of modelling the system's properties. Such models can be used, for example, for anomaly detection, controller design or outcome prediction. For linear systems, this identification task is already well understood and state of the art methods exist.

However, if a system exhibits non-linear behaviour, for example slip-stick-effects due to mechanical friction, the applicability of these methods is limited. In this case different approaches implemented in the state of the art range from white-box to black-box models. Generally, increasing system complexity raises the need for more powerful and often less understandable model architectures in order to produce satisfactory results: White box (based on differential equations or numerical simulations of the physical system components), black box systems (like Gaussian processes, deep neural networks, Support Vector Machines) and grey box models, which often employ a mix of linear and non-linear building blocks. One example of a tool used in engineering are Hammerstein-Wiener models which are a combination of linear and (prior known) non-linear equations (shown in Figure 1). The linear model parameters are determined based on the training data. The non-linear behaviour of models is modelled using lookup tables or user defined non-linear functions.

In this work we present a new type of recurrent neural network layer called the Dynamic Recurrent Network (DYRNN). It is designed for data based modelling of dynamic systems in a sequence-to-sequence manner based on input ($x(t)$) and output ($y(t)$) data. With it, we intend to bridge the gap between dynamic systems theory and recurrent neural networks. The layer's internal computation is based on elemental transfer blocks from linear system identification. This way, the model can offer background knowledge about the examined system's internal properties. Additionally, while the model is trained on sampled data of one sampling rate it can be applied to data of the same

system at a different sampling rate. This can be used to check the robustness of the model or to save time during training. We show that our network produces results which are better than other recurrent networks (RNN, LSTM, GRU) on three different problem datasets. Since the layer can be implemented to be compatible to current deep learning frameworks, it can be combined with state of the art neural network layers (like convolutional or fully connected layers) and training techniques.

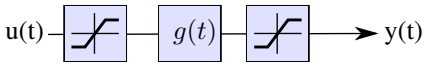


Figure 1: Hammerstein-Wiener model. Static non-linearities before and after a linear differential equation model $g(t)$ can be used to model non-linear dynamic systems.

2 Related Work

Dynamic system identification can be viewed as a sequence-to-sequence task of the modelling of a systems' output measurements based on certain input measurements. Isermann & Münchhof (2011), for example, list several different tools like ARIMA processes for linear systems and multiple neural network architectures for non-linear systems. Examples for the latter are locally recurrent locally feedforward networks (LRGF), Multi Layer Perceptrons (MLP) and Radial Basis Function (RBF) networks of different types of dynamics. These model structures are generalized, however, and as we will show, further theoretical background on linear systems theory could be leveraged in the construction of these models.

Generally, Deep learning offers multiple neural network layer types that can be employed when dealing with sequence-to-sequence problems, like fully connected (FC) networks, convolutional networks (CNN) or recurrent networks. The latter are also known as sequential models (like RNN, LSTM by Hochreiter & Schmidhuber (1997) and GRU by Cho et al. (2014)) and are state of the art layer types for text based sequence-to-sequence problems like machine translation or text processing. Wang (2017) demonstrates a concept of LSTM for dynamic system identification by using several parallel LSTM layers which predict the systems behaviour based on its input and prior predictions and their derivatives. A different approach of modelling dynamic systems are neural ordinary differential equations (ODEs) by Chen et al. (2018). These networks learn dy/dt of a function f with $y(t) = f(x(t))$ and the resulting ODE model is used an numerical integrator/solver (like Runge-Kutta) to compute $y(t)$. This has the advantage of a varying sampling step size which is determined by the solver, but these methods are agnostic of dynamic systems theory knowledge. Similarly, Raissi et al. (2019) use deep learning to learn Partial Differential Equations (PDE) of physical systems in a FC model combined with a numerical integrator. Furthermore, since the evaluation of ODE/PDE models is done using a numerical integrator, the model is difficult to apply in combination with other neural network layers like for example convolutional or recurrent layers. In terms of sampling frequency of the measurement data, recurrent network architectures can only be trained on one specific data frequency, and do not provide the functionality to generalize to other sampling rates of the same system. In such a case one would have to resample the new data to the frequency of the training set.

Explainability approaches for sequential models in text processing deduce which parts of the sentences are relevant for the model's prediction based on the activation of the internal gates (as shown by e.g.Krakovna & Doshi-Velez (2016)). Interpretability of RNN/LSTM or GRU models for continuous measurement data has not been explored yet to our knowledge.

3 Dynamic Recurrent Network

In the following, we first explain the theoretical background from dynamic systems theory used in this work. Then, we describe the structure of our DYRNN. Finally, we show different advantages that result from this structure like training and prediction at different signal sampling rates and interpretability of the models.

3.1 LAYER DESIGN

Our layer's computation function is derived from dynamic system's theory. The background knowledge in this section is documented by Föllinger et al. (2016) and Yarlagadda (2010). In dynamic systems theory a linear system with the external input $u(t)$ and resulting system output $y(t)$ is expressed as the differential equation

$$y(t) + a_1 \cdot \dot{y}(t) + \ldots + a_n \cdot y^{(n)}(t) = b_0 \cdot u(t) + b_1 \cdot \dot{u}(t) + \ldots + b_n \cdot u^{(n)}(t). \tag{1}$$

Therefore, a linear system acts as transformation of the input $u(t)$ based to a convolution ($*$) with the system's linear transfer function $g(t)$ (shown in e.g. Yarlagadda (2010)) to produce the system output $y(t)$ with

$$y(t) = u(t) * g(t). \tag{2}$$

The transfer function $g(t)$ is used in engineering for, amongst others, controller design, system stability analysis or frequency response estimation. Larger, more complicated transfer functions can be seen as several basic functions which are interconnected in a circuit type matter in parallel or in series (see Figure 2).

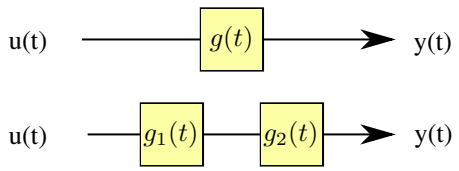


Figure 2: Examples for transfer function circuits. Linear systems are modelled by chaining together several transfer functions.

Dynamic systems theory identifies five basic linear transfer functions from which all linear dynamic systems can be modelled, which we use in the construction of the DYRNN: P, I, D, PT1 and PD (see also Appendix Section A). They have the following functionalities:

- P: **P**roportional gain of the input, implemented as a multiplication with a constant
- I: **I**ntegrating component, meaning the step-wise summation of an input over time.
- D: **D**ifferential component acting as a high-pass filter
- PT1: **P**roportional multiplication of input with **t**ime delay, used to model e.g. voltage in capacitors in RC circuits. This function also acts as a low-pass filter
- PD: **P**roportional increase with a **d**ifferential component

In the following equations K stands for a constant with influence on the output amplitude, while T is a time constant which influences the speed of the system's reaction. For usage in a recurrent layer, these differential equations are discretized with – in our case – first degree forward Euler. This is common in many engineering applications, and means replacing $\dot{y}(t)$ with

$$\dot{y}(t) = \frac{y(k) - y(k-1)}{\Delta t}. \tag{3}$$

This results in discrete recurrence equations, with the sample number k and the time distance between samples Δt. The respective equations of the basic transfer functions are as follows:

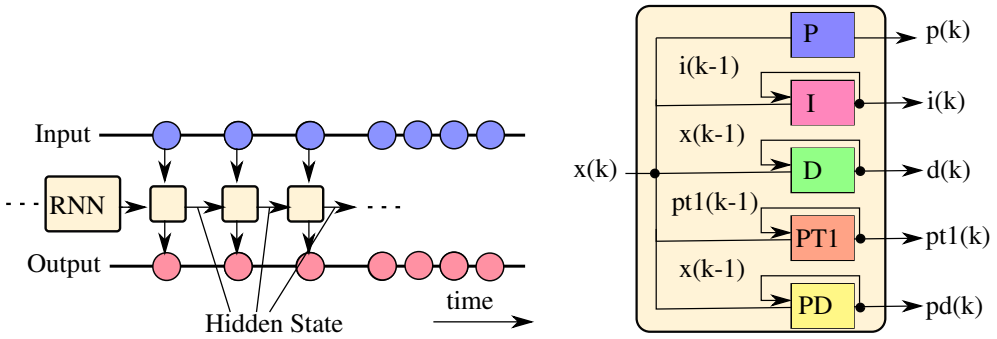
$$p(k) = K_P \cdot x(k) \tag{4}$$

$$i(k) = i(k-1) + \frac{\Delta t}{T_I} \cdot x(k) \tag{5}$$

$$d(k) = \frac{T_D}{\Delta t} \cdot (x(k) - x(k-1)) \tag{6}$$

$$pt1(k) = pt1(k-1) + (K_{PT1} \cdot x(k) - pt1(k-1)) \cdot \frac{\Delta t}{\Delta t + T_{PT1}} \tag{7}$$

$$pd(k) = K_{PD} \cdot (x(k) + \frac{T_{PD}}{\Delta t} \cdot (x(k) - x(k-1))), \tag{8}$$



(a) Example of sequence-to-sequence translation using a recurrent network layer. The layers hidden state is used for the computation of the next timestep's result.

(b) Design of DYRNN unit with five different inner equations. This architecture results in multiples of five output channels.

Figure 3: Visualization of how a recurrent neural network processes a sequence and how our unit type is structured.

with all $K, T > 0$. The equations above are implemented as the computation function of a recurrent network layer as described in Appendix Section A.1. Recurrent models incorporate a unit which iterates over the layer's input sequence while keeping additional data in a hidden state to provide information necessary for the computation of the next step's result (see Figure 3a). We formulate one unit's computations based on the number of input channels n_{ic} and the number of outputs per basic transfer function type n_{oc}. The number of output channels per layer n_{layer} amounts to (base component count * n_{oc} = 5 * n_{oc}), as shown in Figure 3b. The fact that DYRNN can be implemented with an interface compatible with other recurrent networks allows the modelling of a systems properties by training in a sequence-to-sequence fashion using out of the box backpropagation through time, gradient descent and optimizer functions as implemented in state of the art frameworks like Pytorch (by Paszke et al. (2019)) and Tensorflow (by Mart\'ın Abadi et al. (2015)). In the following we present an example architecture using DYRNN and several different features that can be leveraged due from this design.

3.2 Example Architecture

Since each input of the layer is connected to five basic components, stacking DYRNN layers results in an increasing number of output channels, as shown in Figure 4. The architecture we used in our experiments is a cascading row of DYRNN layers, which means that all inputs and intermediate results are concatenated and passed to a per-time-step FC network as shown in Figure 5. Using non-linear activation functions, this design results in a similar structure as known from Hammerstein-Wiener models for modelling of static non-linearities in the data. In case of noisy input data, for example due to sensor noise, additional convolutional layers could be used as well.

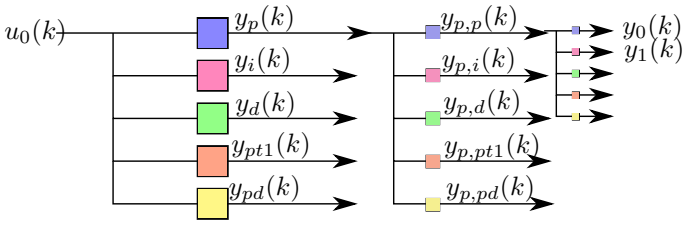


Figure 4: Resulting model when stacking DYRNN layers. Each of the output channels would produce another five output channels for each connected sub-component.

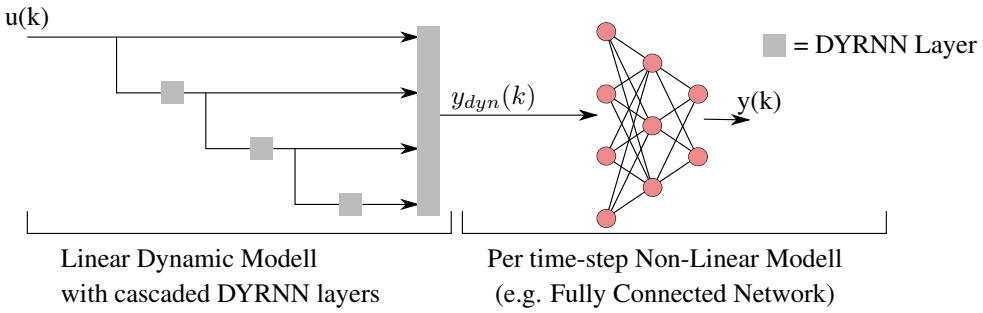


Figure 5: Combination of DYRNN and non-linear fully connected network. The output channels of the DYRNN part of the model are concatenated before being passed into a per-timestep non-linear fully connected network.

3.3 Generalization for Different Signal Sampling Rates

The new parameter Δt means that a model can be used on datasets with a different sampling rates by adjusting the Δt when interacting with the model. Resampling can be leveraged as a new means to perform time series data augmentation. Training on different rates and testing on another sampling rate can also provide insight about a model's robustness. Additionally, training time can be reduced by subsampling the dataset during training, and using the original sampling rate while inferencing for greater prediction accuracy (see Figure 6).

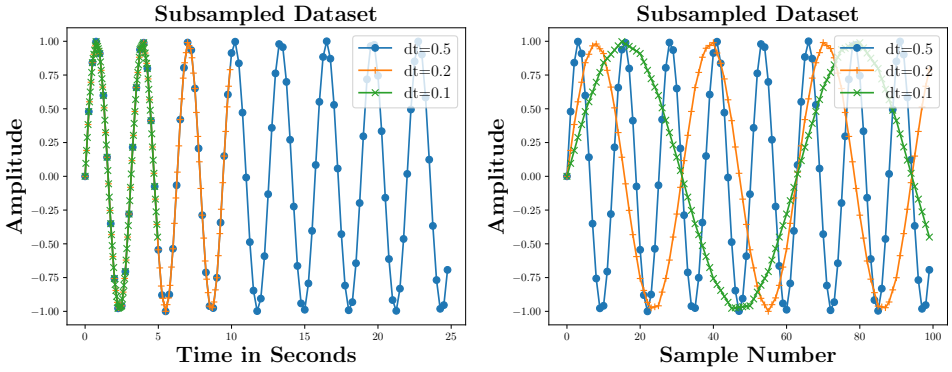


Figure 6: Example dataset y(k) = sin(x(k)) at different sampling rates. Keeping the amount of samples the same, a larger sampling rate results in a larger range of covered time.

3.4 Interpretability of Resulting Models

Based on architectures like in Section 3.2, it is possible to interpret the significance of a variable towards the final dataset as well as the properties of its influence. While the DYRNN layers learn the time dependant system behaviour, a per-timestep fully connected network selects the relevant dynamics of the model. The higher the FC's learnt weight connected to a specific subunit, the more significant it is for the modelling of the dynamic system (see Figure 7).

The FC layer's weights are used globally and in a time independent manner. Thereby the edge weight can be interpreted as the global importance of a DYRNN sub-computation in the modelling of the system. Figure 7 visualizes this. For example, given a network of one DYRNN layer and a fully connected layer, if the subunit P, I and PT1 are weighted by the FC with 0.5, 1.5 and 0.8 as opposed to the D and PD with 0.1 and 0.2, this shows that the modelled system mostly displays P, I and PT1 properties. This architecture means that we do not rely upon the DYRNN's hidden states for interpretation. Instead, the static part of the network is designed to select relevant system properties, while enabling the modelling of non-linear parts of the system. Nonetheless interpreting the results

will become more difficult if complexity of the static part of the network increases. Further methods like L1/L2 regularization in the FC layers can also be of use in this process.

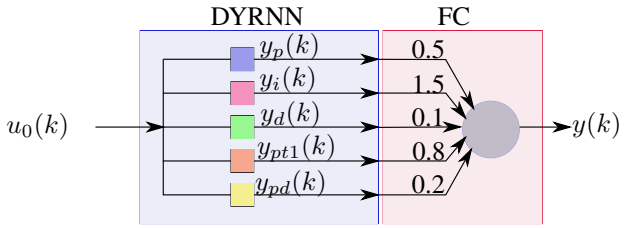


Figure 7: Visualization of the interaction of a DYRNN layer and a fully connected (FC) network layer. Via the FC layers learnt weights, it is possible to gain insight of the underlying system dynamics.

4 EXPERIMENTS

We evaluated our network on three different datasets. The first is an electical circuit simulated using using Matlab Simulink, as shown in the Appendix in Figure 12. The input of our model is the input voltage of the circuit, while the output value is the voltage measured over the capacitor. The simulation's sampling rate is 0.005s. We simulated three time series for training, validation and testing with length of 2 seconds. The input consists of a square signal with randomly changing amplitudes from 0.5s to 1.5s. The remainder of the time, the input voltage is 0V. The other two datasets are from DaISy (De Moor B.L.R. (a) and De Moor B.L.R. (b)). Further information about the experiments is documented in Appendix Section A.2.

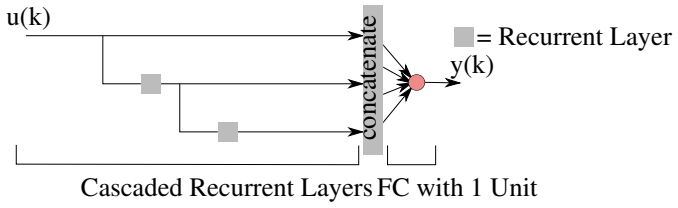


Figure 8: Architecture used in our experiments.

The first goal of our experiments is to show that our model performs well compared to other recurrent networks, namely RNN, LSTM and GRU as implemented in Tensorflow. The network architecture is as shown in Figure 8. The recurrent part of the network is built in a cascaded fashion of two recurrent layers which are concatenated with the original input. Its results are passed to a linear fully-connected layer with one unit. The number of units in the recurrent layers were chosen to result in a similar parameter distribution to DYRNN, as shown in Table 1 for the first dataset (Appendix Table 5 and 6). Additionally, for the Circuit dataset we train the model on different sampling rates and examine its ability to generalize to other sampling rates. For this, we subsampled the dataset at different frequencies using linear interpolation. The constellation of sampling rates in these three runs are described in Table 2. All networks were trained with the Adam Optimizer by Kingma & Ba (2014) and the Huber Loss function with standard parameters defined in Tensorflow. The amount of training iterations was kept at a maximum of 5000 epochs for Circuit and 3000 for the DaISy datasets. For Circuit, training was terminated once overfitting was detected compared to the validation set. Evaluation of results is done on a seperate testing dataset.

5 RESULTS

After 10 runs each, the different recurrent networks performance on the testing set is shown in Figure 10. Figure 9 shows the best and worst runs on the testing set for the Circuit dataset. On this dataset, DYRNN performs consistently better than each of the recurrent networks. This is to be expected since an idealized capacitor acts as a PT1 element, so the DYRNN is predestined for this modelling

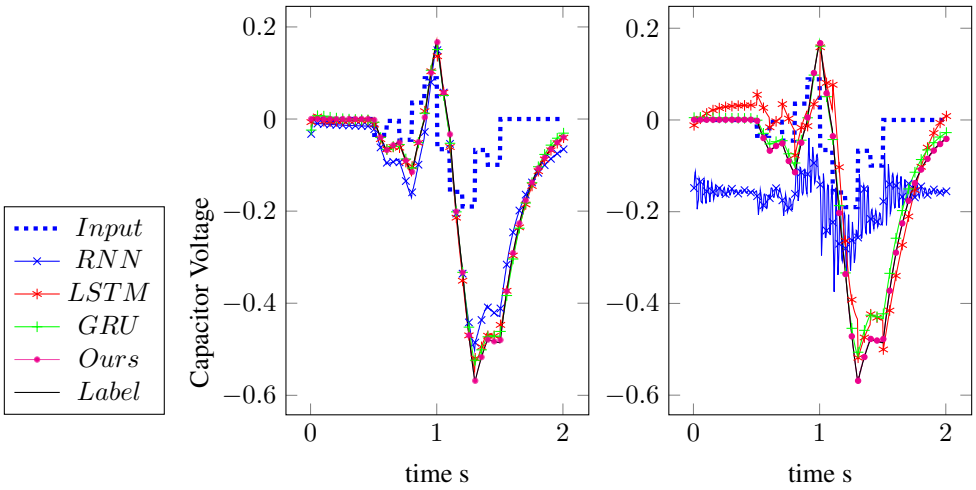
task. Nonetheless, all layer types preform well on the dataset, if training was successful. The RNN network, for example, sometimes (depending on the randomly initialized parameters) was unable to fit the data in a meaningful way (see Figure 9b). For each of our experiments, we used a similar count of parameters in the recurrent part of the network and the same layer constellations (Table 1,5 and 6). Due to the design of our network, its FC part contains more parameters than the other networks. Since we omitted an activation function in the FC layer, it amounts to a linear combination of all channels from the cascaded part. Therefore, we assume that the improved performance of our networks stems from the DYRNN computation functions instead of the increased count of parameters in the FC layer. Given a different network configuration and more parameters, it may be possible for the other layer types to achieve results closer to ours for the shown datasets.

Table 1: Amount of units per network layer and recurrent layer type for the the circuit dataset. Values in parentheses denote the parameter count of that layer.

Units (Param. Count)	RNN	LSTM	GRU	Ours (DYRNN)
Units in first recurrent layer	3 (15)	2 (32)	2 (30)	1 output per s.u. (7)
Units in second recurrent layer	5 (45)	2 (40)	2 (36)	1 output per s.u. (35)
Number of parameters in FC layer	10	6	6	32
Number of parameters total	70	78	72	74

Table 2: Amount of parameters per network layer and recurrent layer type.

Run	Training set resampled at	Prediction set resampled at
A	0.7 Δt_{orig}	1.0 Δt_{orig}
B	1.0 Δt_{orig}	0.7 Δt_{orig}
C	0.5 Δt_{orig}, 1.0 Δt_{orig}	0.7 Δt_{orig}

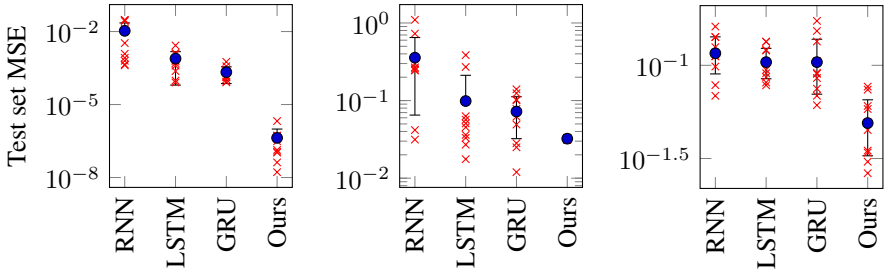


(a) Best results after 10 runs.

(b) Worst results after 10 runs.

Figure 9: Comparison of prediction on the test set of the different recurrent layer types. Ten models were trained per layer type for 5000 epochs.

Our experiments concerning resampling to different sampling rates show that training for 0.7 Δt and predicting to 1.0 Δt performs worse than the other two variants. From this we conclude that the total amount of time covered by the training data is more important than the training at different sampling rates for this dataset. It also shows that the model is able to extrapolate for each of the different sampling rates to another. Our implementation which allows the per-batch sampling rate



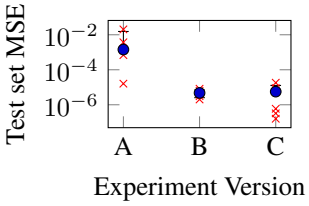
(a) Capacitor Circuit (b) Heating System (c) Thermal Resistance

Figure 10: Results on different evaluation datasets after 10 runs each using the same architecture with varying layer types.

Table 3: MSE and Standard deviation of the different layer types on the experiment datasets.

	RNN	LSTM	GRU	Ours
Circuit	1.1e-2±1.2e-2	7.8e-4 ± 7.1e-4	2.1e-4 ± 1.4e-4	**4.3e-7 ± 5.6e-7**
Heating Sys.	3.6e-1 ± 2.9e-1	9.9e-2± 1.1e-1	7.2e-2 ± 4.0e-2	**3.2e-2 ± 6.3e-4**
Thermal Res.	1.2e-1 ± 2.6e-1	1.0e-1±1.9e-2	1.0e-1± 3.4e-2	**4.9e-2 ± 1.6e-2**

during training is computationally more expensive than the one which relies upon a fixed sampling rate, so the inexpensive version is a possible alternative for larger datasets.



	A	B	C
Mean Loss	1.43e-3	**4.76e-06**	5.63e-06
Loss Std.-dev.	1.4e-3	2.21e-06	7.11e-06

Figure 11: Results on the test set. A: $0.7\Delta t_{orig} \rightarrow 1.0\Delta t_{orig}$; B: $1.0\Delta t_{orig} \rightarrow 0.7\Delta t_{orig}$; C: $0.5\Delta t_{orig}, 1.0\Delta t_{orig} \rightarrow 0.7\Delta t_{orig}$

6 Future Work

We have shown a new type of recurrent layer designed for the modelling of dynamic systems. Based on this, we see several potential areas for future work, for example in the domain of automated architecture search. To enhance the models expressiveness for the user and quantify model uncertainty, a combination with Bayesian methods could also be evaluated. We showed how our network could be interpreted; but have not actually evaluated this yet. Here, a study with a variing array of users would be of interest.
Our experiment showed good results on three datasets of system identification. A different area of application can be model based reinforcement learning tasks, since the layers' computation blocks are commonly used in control engineering as well. This should also be evaluated.

7 Concluding Remarks

In this work we present a new recurrent network architecture which is based on dynamic systems theory. We show that the learned system dynamics can produce models which can extrapolate to different sampling rates in the data. Due to the specific meaning of the cells' computation function, the learned model can be leveraged to gain insight on the underlying system dynamics.

REFERENCES

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations, 2018. URL http://arxiv.org/pdf/1806.07366v5.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation: GRU, 2014. URL http://arxiv.org/pdf/1406.1078v3.

De Moor B.L.R. DaISy: Database for the Identification of Systems: Heat flow density through a two layer wall, Thermic Systems, [99-011], a. URL http://homes.esat.kuleuven.be/~smc/daisy/.

De Moor B.L.R. DaISy: Database for the Identification of Systems: Heating system, Thermic Systems, [99-001], b. URL http://homes.esat.kuleuven.be/~smc/daisy/.

Otto Föllinger, Ulrich Konigorski, Boris Lohmann, Günter Roppenecker, and Ansgar Trächtler. *Regelungstechnik: Einführung in die Methoden und ihre Anwendung*. Lehrbuch Studium. VDE Verlag GmbH, Berlin and Offenbach, 12., überarbeitete auflage edition, 2016. ISBN 978-3-8007-4201-1.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

Rolf Isermann and Marco Münchhof. *Identification of Dynamic Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-540-78878-2. doi: 10.1007/978-3-540-78879-9.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2014. URL http://arxiv.org/pdf/1412.6980v9.

Viktoriya Krakovna and Finale Doshi-Velez. Increasing the Interpretability of Recurrent Neural Networks Using Hidden Markov Models, 2016. URL http://arxiv.org/pdf/1611.05934v1.

Mart\'ın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL https://www.tensorflow.org/.

MATLAB. Version 9.5.0.944444(R2019b), 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc, 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 00219991. doi: 10.1016/j.jcp.2018.10.045.

Yu Wang. A new concept using LSTM Neural Networks for dynamic system identification. In *2017 American Control Conference ACC*, pp. 5324–5329. 2017. doi: 10.23919/ACC.2017.7963782.

Radha K. Rao Yarlagadda. *Analog and digital signals and systems*. Springer, New York, 2010. ISBN 978-1-4419-0033-3. URL http://www.loc.gov/catdir/enhancements/fy1210/2009929744-d.html.

A APPENDIX

BASE FUNCTIONS IN DYNAMICAL SYSTEMS THEORY

In dynamic systems theory, the basic elements are visualized using their response towards an input unit step function $\sigma(t)$ starting at t=0 with the amplitude of 1. In Table 4, we show a short summary over the blocks used in our network layer. Blocks display a specific behaviour, which is visualized by looking at the response towards the Unit step function.

A.1 IMPLEMENTATION DETAILS

Sequential models keep a hidden state while iterating over an input sequence, to encode information necessary for the computation of the next step's result.

To deal with multiple input channels and the possibility of several base components of the same type, we define the computations as follows:

$$p(k) = x(k) \circ |K_P| \tag{14}$$

$$i(k) = i(k-1) + \left(i(k) \bullet \frac{1}{\Delta t}\right) \circ \frac{1}{|K_I|} \tag{15}$$

$$d(k) = (x(k) - x(k-1)) \bullet \frac{1}{\Delta t} \circ |T_{PD}| \tag{16}$$

$$pt1(k) = pt1(k-1) + (x(k) \circ |K_{PT1}| - pt1(k-1)) * \frac{1}{\frac{1}{\Delta t} \circ |T_{PT1}| + 1} \tag{17}$$

$$pd(k) = \left(x(k) + ((x(k) - x(k-1)) \bullet \frac{1}{\Delta t}) * |T_{PD}|\right) \circ |K_{PD}|. \tag{18}$$

$$\tag{19}$$

With $x(t)$'s dimensionality of [batch $\times$ sample $\times$ input channels] and $\Delta t(t)$ as [batch $\times$ sample $\times$ 1] or Δt as [batch $\times$ 1 $\times$ 1] , the multiplications expressed in Einstein's sum formation (which can be implemented as described in e.g. https://www.tensorflow.org/api_docs/python/tf/einsum) are: $\circ$ as 'bi, ij $\rightarrow$ bij', $\bullet$ as 'bi, b $\rightarrow$ bi' and $*$ as 'bij, bij$\rightarrow$bij', i.e. the Hadamard product operator. The dimensions of the trainable parameters result to

$$K_P \in \mathbb{R}^{ic \times oc}$$
$$K_I \in \mathbb{R}^{ic \times oc}$$
$$T_D \in \mathbb{R}^{ic \times oc}$$
$$K_{PT1} \in \mathbb{R}^{ic \times oc}, T_{PT1} \in \mathbb{R}^{ic \times oc}$$
$$K_{PD} \in \mathbb{R}^{ic \times oc}, T_{PD} \in \mathbb{R}^{ic}$$

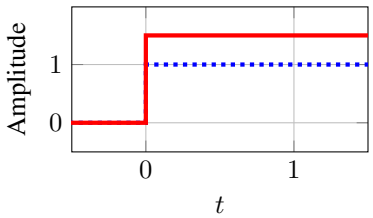
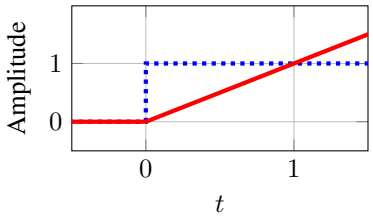
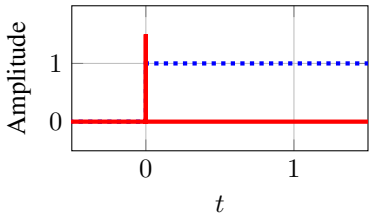
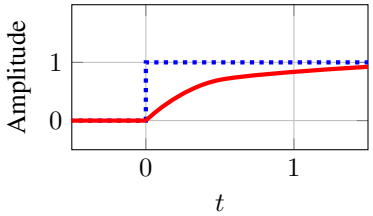
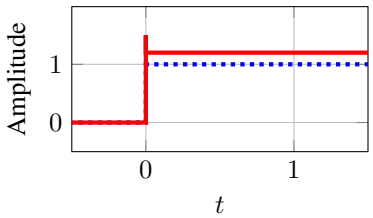
given this setup. The weights were initialized in a random uniform fashion, with a range $[0.1, 0.2]$.

A.2 FURTHER EXPERIMENT DOCUMENTATION

We evaluated our network on three different datasets. One is our own dataset from a Matlab (MATLAB (2018)) simulation as shown in Figure 12, the other two are the "Heat flow density through a two layer wall" (De Moor B.L.R. (a)) and the "Heating system" (De Moor B.L.R. (b)) datasets from the DaISy system identification benchmarks:

- Heating System Benchmark: Prediction of halogen lamp temperature based on input voltage

Table 4: Basic linear components, input $\sigma(t)$ ····· and system response $y(t)$ ——. Step response and differential equations of the used linear components.

System	Unit Step Function Response	General Block Output Function
P		$y(t) = K \cdot u(t)$ (9)
I		$\dot{y}(t) = K \cdot u(t)$ (10)
D		$y(t) = K \cdot \dot{u}(t)$ (11)
PT1		$T \cdot \dot{y}(t) + y(t) = K \cdot u(t)$ (12)
PD		$y(t) = K \cdot (\dot{u}(t) \cdot T + u(t))$ (13)

- Two Walls Benchmark: Prediction of heat flow through two walls based on measurements before and between the walls

For our circuit dataset we trained until overfitting on a validation set or until 5000 epochs. For the remaining two DaISy datasets, the models were trained for 3000 epochs on a training set without early stopping, because we noticed that for example RNN the training would terminate prematurely, yielding worse results than when trained until completion. After training, the models were evaluated on a testing set split from the complete dataset. The DaISy Datasets to not have a zero-level starting interval like in our circuit dataset. For simplicity's sake, our chosen network architecture does not take the initial value of the output time series into account. Therefore an offset could be seen in predictions on the testing set for all layer types. In a post-processing step this offset was computed using

the median difference between prediction and label data, and subtracted from the prediction prior to the final MSE evaluation. The amount of units and parameters are shown in Tables 5 (Heating System) and 6 (Thermic Wall Res.).

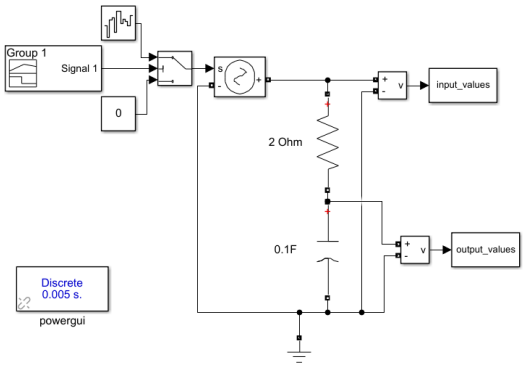


Figure 12: Simulation used for the generation of experiment data. It contains a resistor of 2 Ω and a capacitor of 0.1F. Input sequence is the source voltage, while output to predict is the voltage measured over the capacitor.

Table 5: Amount of units per network layer and recurrent layer type for the Heating System dataset.

Units (Param. Count)	RNN	LSTM	GRU	Ours (DYRNN)
Units in first recurrent layer	3 (15)	2 (32)	2 (30)	2 output per s.u. (14)
Units in second recurrent layer	7 (77)	3 (72)	3 (63)	1 output per s.u. (70)
Number of parameters in FC layer	12	7	7	63
Number of parameters total	104	111	100	147

Table 6: Amount of units per network layer and recurrent layer type for the Thermic Wall Res. dataset.

Units (Param. Count)	RNN	LSTM	GRU	Ours (DYRNN)
Units in first recurrent layer	3 (18)	2 (40)	2 (36)	1 output per s.u. (14)
Units in second recurrent layer	7 (77)	3 (72)	3 (63)	1 output per s.u. (70)
Number of parameters in FC layer	13	8	8	63
Number of parameters total	108	120	107	147