Machine learning to approximate solutions of ordinary differential equations: Neural networks vs. linear regressors

Georg Engel[0000−0001−6405−9384]

Christian Doppler Laboratory for Quality Assurance
Methodologies for Autonomous Cyber-Physical Systems,
Institute for Software Technology, Graz University of Technology
engel@ist.tugraz.at

Abstract. We discuss surrogate models based on machine learning as approximation to the solution of an ordinary differential equation. Neural networks and a multivariate linear regressor are assessed for this application. Both of them show a satisfactory performance for the considered case study of a damped perturbed harmonic oscillator. The interface of the surrogate model is designed to work similar to a solver of an ordinary differential equation, respectively a simulation unit. Computational demand and accuracy in terms of local and global error are discussed. Parameter studies are performed to discuss the sensitivity of the method and to tune the performance.

Keywords: ordinary differential equations, machine learning, surrogate model, neural network, multivariate linear regressor

1 Introduction

Machine learning techniques are successful in many fields such as image and pattern recognition. In recent years, interest increases in applying these techniques also to other fields of science, which are conventionally dominated by e.g. physically motivated models. The success of these techniques is usually tied to the amount and quality of data available for training the model. In engineering and science, these data can be generated using measurements or simulations.

The present paper discusses machine learning techniques in the context of simulations based on ordinary differential equations. These are are often used to describe physical or engineering systems. Solving these equations can be expensive in practice, in particular for complex systems and when solutions are required repeatedly, like in optimization problems. The computational costs encountered are often too high, e.g. when combining these models with the real world, like in cyber-physical systems, where real-time performance is required. The purpose of model order reduction or surrogate models is to reduce the computational costs, taking into account some limited reduction of accuracy.

A prominent method for model order reduction is proper orthogonal decomposition, which was successfully applied already many years ago, e.g. to analyze

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_13

turbulent flows [1]. In the present work, we follow a different approach, adopting the perspective of co-simulation. Co-simulation denotes the dynamic coupling of various simulation units, which exchange information during simulation time, for a review see [2, 3]. Several co-simulation interfaces have been defined and implemented, e.g. the Functional Mock-Up Interface standard [4], for a discussion see [5]. A discussion and comparison was performed e.g. in [6]. Typically, in such a setup the computational costs are dominated by only few of the individual simulation units. It is intriguing to replace only the costly simulation units by cheaper surrogate models using machine learning techniques. The framework of co-simulation might be exploited in this context in a way that the master algorithm is left unaltered. However, as models generated by machine learning do not explicitly incorporate a solver, the meaning of co-simulation fades and one could use a notion of model-coupling instead.

Machine learning has been applied to differential equations before, some work dates back more than twenty years, but a considerable boost appeared very recently. Artificial neural networks have been proposed to derive analytical solutions for differential equations by reformulating them as optimization problem [7]. Deep reinforcement learning was suggested for general non-linear differential equations, where the network consists of an actor that outputs solution approximations policy and a critic that outputs the critic of the actor's output solution [8]. An approximation model for real-time prediction of non-uniform steady laminar flow based on convolutional neural networks are proposed in [9]. Considering the simulation of buildings, several efforts have been made to reduce the computation costs, in particular within the IBPSA 1 project [10]. The input and output data of simulators is used to train neural networks in a co-simulation setting (referred to as "intelligent co-simulation"), and the model is compared to model order reduction by proper orthogonal decomposition [11]. A similar approach called "component-based machine learning", for the same field of application, is pursued in [12]. The same setting was investigated using deep-learning neural-networks [13], where a considerable computational speedup over the physical model at similar accuracy is claimed.

The present work focuses on simple case studies where extensive parameter studies are feasible and more insights related to the structure of the method and the case study can be achieved. A first discussion in this respect, considering neural networks for the Dahlquist equation and the Van der Pol oscillator was presented in [14]. The main contribution of this work is as follows:

- The general method for using data models to approximate solutions of ordinary differential equations is discussed.
- A neural network and a multi-variate regressor are introduced as specific examples.
- Accuracy and computational costs are discussed.
- Parameter studies are performed to discuss the sensitivity of the method.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_13

2 Method

Data models shall be trained as surrogate model for a simulation unit. The surrogate model obtains the same input and output variables as the original simulation unit. The input variables are represented as explicit function of time. For simplicity, the output shall be the state variable itself. Output variables as functions of the state variable mean a straightforward generalization of the method. The data model contains short-term memory units to store previous values of the state variable. These serve as inputs for the calculation in addition to the present value and the explicit function of time to predict the next value of the state variable.

This way, the model builds a trajectory, where errors can easily accumulate, which is typical for solutions of differential equations. As training data, we make use of the solution to differential equations generated by a traditional solver. For the latter, we employ the routine *odeint* from the python package *scipy* [15].

For the neural network, the multi-layer perceptron of the package scikit-learn is chosen [16]. Unless stated otherwise, we use one hidden layer with one neuron, a tolerance of 10^{-5} and the solver "lbfgs". Model selection on a test data set is used to apply 100 different random seeds in the optimizer of the neural network, which is common practice and crucial here to obtain a satisfactory validation. While this is satisfactory, note that it can be expected that the true optimum is not found, and hence some "random" additional error is found in the validation when comparing different settings. The multivariate linear regressor is constructed using the Moore-Penrose pseudo-inverse from the python package *numpy*. The regressor is limited to linear functions for the case study considered in the present work. An extension to polynomials is very straightforward but not necessary if the underlying differential equation is linear.

The method is designed to allow for generalization in several respects to be considered in future. In particular, a wrapper for simulation units in a co-simulation can encapsulate the method as a surrogate model. In such applications, it might be better to take more arbitrary snapshots of the data for training, which means a further extension to the consecutive data in simulation time considered here. Finally, for the simple cases considered here, the output coincides with the state variables of the model. In actual applications, only the output as some function of the state variable might be available for training the data model.

3 Model / Case Study

For testing the method, we consider a damped harmonic oscillator amended by an explicit function of time as perturbation,

$$\ddot{y} + 0.1\dot{y} + y + 0.1\sin^2\left(\sqrt{2}t\right) = 0\,, \tag{1}$$

with the initial condition $y(0) = 1$ and $\dot{y}(0) = 0$. The damping and the perturbation are chosen such that the behaviour of $y(t)$ changes significantly from the

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_13

training to the validation phase, which is supposed to challenge the method. For the same purpose, the frequencies of the harmonic oscillator and the perturbation are chosen differently. Note, however, that the model is chosen such that the solution $y(t)$ is bounded for all t within a range which can easily be covered during the training phase. Hence, even though the $y(t)$ changes qualitatively from training to validation, the data model can be considered to remain within the "'interpolation" range, where the training data should provide a good basis for prediction. The perturbation term represents an input for a simulation unit in a co-simulation scenario of an actual application, e.g. weather data in case of a building performance simulation.

4 Results

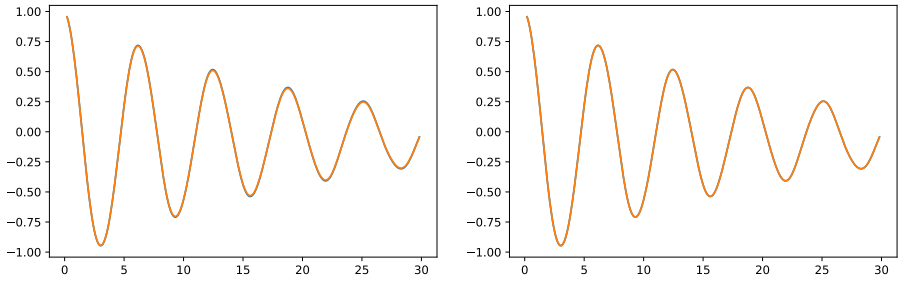

Fig. 1. Training results. The blue curve shows the "exact" solution generated by the traditional solver, the orange curve shows the result for the data model. If only one curve is visible, the predicted one lies one top of the "exact" solution. Left: neural network; right: linear regressor. Neither model shows a significant deviation.

We choose the interval $t \in [0, 100)$ and 1000 time steps. $t \in [0, 30)$ is used for training, $t \in [30, 50)$ as testing for potential model selection and $t \in [50, 100)$ for validation. The Figures shown in this section always show results for the neural network on the left hand side and results for the linear regressor on the right hand side.

Results for training, testing and validation are shown in Figure 1, Figure 2 and Figure 3, respectively. For the chosen parameters, the validation works very well. Note how the behaviour of the function $y(t)$ changes qualitatively from the training to the test and to the validation phase. It is remarkable that the prediction during the validation phase, dominated by the perturbation term, is satisfactory even though the perturbation is barely visible for the eye during the training phase.

After successful training, the predictive quality of a data model relates to the similarity of the input data for training and prediction. This similarity can be discussed in terms of distance between the data, as shown in Figure 4. The

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_13

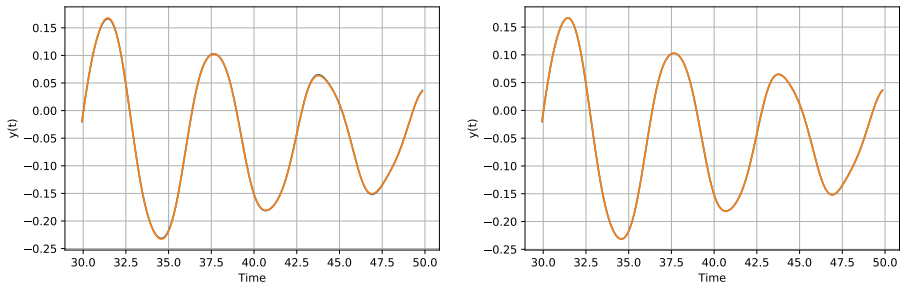

Fig. 2. Like, Figure 1, but for test results, used for model selection in case of the neural network. Note that the effect of the perturbation becomes visible as distortion of the oscillation.

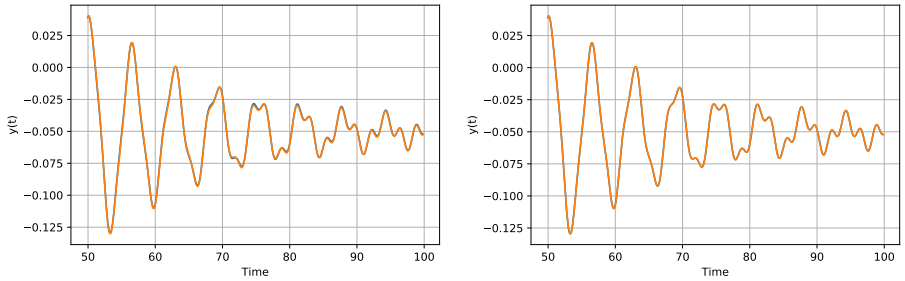

Fig. 3. Like, Figure 1, but for validation results. Note how the behaviour of the solution $y(t)$ is qualitatively different from the training phase. For the neural network (right hand side), we see some small deviation where the blue curve becomes visible below the orange one. These deviation are better visible in Fig. 5.

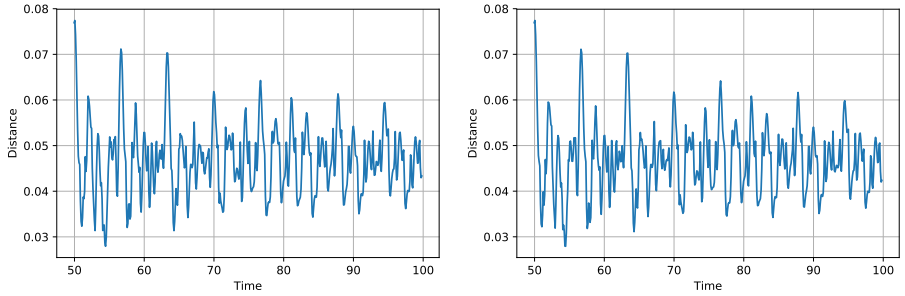

Fig. 4. The distance in theory space from input variables during validation compared to training data is shown. The distance is defined as Euclidean distance from the nearest point from the training data set. Left: Results for the neural network. Right: Same as left, but for the linear regressor.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_13

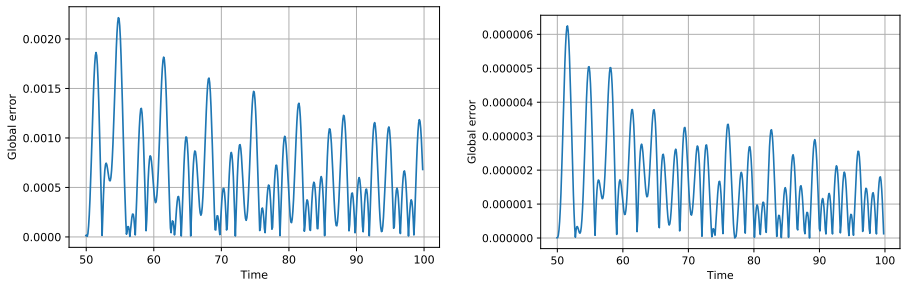

Fig. 5. The global integration error generated by the data model along the trajectory, determined by comparison to a trajectory generated by a traditional solver. Left: Results for the neural network. Right: Same as left, but for the linear regressor. Note the different scale on the y-axis in the two plots.

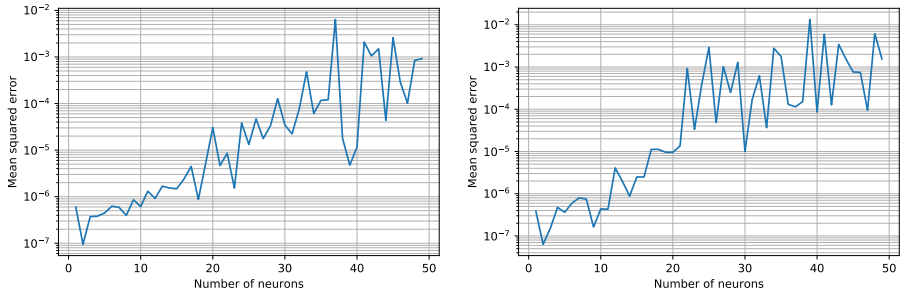

Fig. 6. The mean squared error during the validation, shown versus the number of neurons for one hidden layer. Left: Results for the neural network with one hidden layer. Right: Results for the neural network with two hidden layers.

distance is defined as Euclidean distance of the input variables during the validation from the nearest point of the input variables during the training. The distance does not accumulate over time and the model can be considered as being interpolation and not extrapolation. The global error of the model can easily be estimated considering the deviation of the predicted trajectory from the trajectory generated by the traditional solver, shown in Figure 5. Using this measure, the linear regressor performs considerably better (three orders of magnitude) than the neural network. We conjecture that the linear regressor performs so well because the model and the solver are linear except for the perturbation. The non-linearity of the latter is not modelled by the regressor, since it is used as one of the input variables.

Parameter studies have been performed to discuss the sensitivity of the method. The mean square global error during the validation is shown versus the number of neurons in Figure 6 for one and two hidden layers (each with the same number of neurons). There is a significant amount of noise, which might

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_13

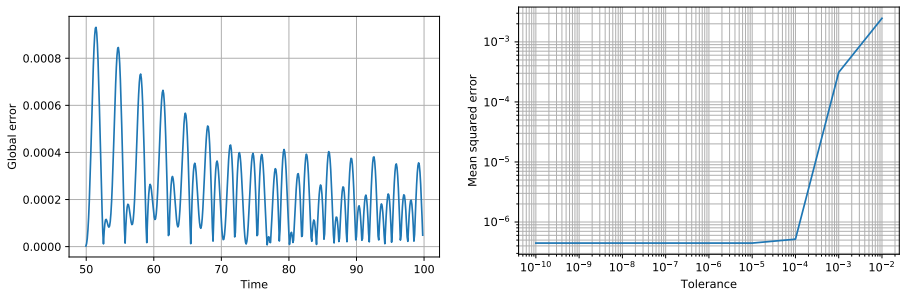

Fig. 7. Left: Global integration error for the validation of the neural network using two neurons in one hidden layer, which is the best neural network found in this study in the sense of least mean squared error, compare Figure 5 and Figure 6. Right: The mean squared error during the validation, shown versus the required tolerance during the training of the neural network.

relate to the difficulty of finding the global optimum in the training phase, as discussed earlier. However, a trend is observed for the method to worsen towards a higher number of neurons, independent of the number of hidden layers. It seems that superfluously increasing of the complexity of the data model is disadvantageous. Different sources of error might accumulate, including the difficulty of finding the optimum. This is true at least at this level of accuracy and for simple models like the considered one here.

The best performing neural network appears to use two neurons in one hidden layer, for which we show results for the global error in Figure 7, left hand side. Still, the global error is worse than that of the linear regressor by three orders of magnitude. The dependence on the tolerance used in the optimizer of the neural network is shown in Figure 7, right hand side. We conclude that a further improvement of the performance of the neural network cannot be achieved by such measures for this simple case study. However, for more complex applications, it is expected that larger neural networks will be required.

The computational costs have been measured using the python package *timeit*. The traditional solver took about 7 ms, the neural network 31 ms and the linear regressor 26 ms on a 2.4 GHz Intel Core i5. Note that the computational costs for training the data model are neglected here. This is reasonable, as the training could be done offline or at least easily parallelized. The high computational costs of the data models compared to the traditional solver are not surprising, as no proper model order reduction is applied. Benefits in terms of saved computational costs are however expected for more complicated models where model order reduction is desirable and can be achieved by the proposed method.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_13

5 Discussion / Outlook

Machine learning surrogate models for ordinary differential equations have been investigated considering a neural network and a multivariate linear regressor. A damped and perturbed harmonic oscillator is introduced as case study to test the method. It was shown that a rather small neural network can describe these simple models to fairly good accuracy, and the linear regressor to even better accuracy. The case study was specifically designed such that the validation phase differs from the training phase significantly, where the performance of the method is remarkable. For the neural network, it was crucial to apply various random seeds for the optimizer of the neural network during the training phase to obtain satisfactory performance.

As next step, non-linear and hybrid models (discrete/continuous) shall be discussed, like the Dahlquist's test equation and the bouncing ball equation. The method presented here shall then be tested further for more complicated models, where model order reduction is desired. A realistic case study will be considered in the field of thermal energy engineering. It will be discussed which problems are particularly suited for this kind of surrogate models. The method shall be applied in the context of co-simulation, replacing a simulation unit after sufficient training on the fly.

6 Acknowledgements

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged. We further acknowledge fruitful discussions with Gerald Schweiger, Claudio Gomes and Philip Ohnewein.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_13

Bibliography

[1] G Berkooz, P Holmes, and J L Lumley. The Proper Orthogonal Decomposition in the Analysis of Turbulent Flows. *Annual Review of Fluid Mechanics*, 25(1):539–575, jan 1993.

[2] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: State of the art. *CoRR*, abs/1702.0, feb 2017.

[3] Gerald Schweiger, Cláudio Gomes, Georg Engel, Irene Hafner, Josef Schöggl, and Thierry S Nouidui. Co-Simulation: An empirical survey identifies promising standards, current challenges and research needs. *Submitted*, 2018.

[4] T Blochwitz, M Otter, M Arnold, C Bausch, C Clauß, H Elmqvist, A Junghanns, J Mauss, M Monteiro, T Neidhold, D Neumerkel, H Olsson, J V Peetz, and S Wolf. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *8th International Modelica Conference 2011*, pages 173–184, 2009.

[5] Gerald Schweiger, Cláudio Gomes, Georg Engel, Irene Hafner, Josef-Peter Schoeggl, and Thierry Stephane Nouidui. Functional Mockup-Interface : An empirical survey identifies research challenges and current barriers. In *American Modelica Conference 2018*, 2018.

[6] Georg Engel, Ajay S. Chakkaravarthy, and Gerald Schweiger. A General Method to Compare Different Co-Simulation Interfaces: Demonstration on a Case Study. In Janusz Kacprzyk, editor, *Simulation and Modeling Methodologies, Technologies and Applications*, chapter 19. Springer, 2018.

[7] I.E. E. Lagaris, A. Likas, and D.I. I. Fotiadis. Artificial Neural Networks for Solving Ordinary and Partial Differential Equations. *IEEE Transactions on Neural Networks*, 9(5):1–26, 1997.

[8] Shiyin Wei, Xiaowei Jin, and Hui Li. General solutions for nonlinear differential equations: a deep reinforcement learning approach. Technical report, 2018.

[9] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional Neural Networks for Steady Flow Approximation. In *Proceedings of the 22nd {ACM} {SIGKDD} International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 481–490, 2016.

[10] IBPSA. IBPSA Project 1 - https://ibpsa.github.io/project1/.

[11] Julien Berger, Walter Mazuroski, Ricardo C.L.F. Oliveira, and Nathan Mendes. Intelligent co-simulation: neural network vs. proper orthogonal decomposition applied to a 2D diffusive problem. *Journal of Building Performance Simulation*, pages 1–20, feb 2018.

[12] Philipp Geyer and Sundaravelpandian Singaravel. Component-based machine learning for performance prediction in building design. *Applied Energy*, 228:1439–1453, oct 2018.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_13

[13] Sundaravelpandian Singaravel, Johan Suykens, and Philipp Geyer. Deep-learning neural-network architectures and methods: Using component-based models in building-design energy prediction. *Advanced Engineering Informatics*, 38(May):81–90, oct 2018.
[14] Georg Engel. Neural networks to approximate solutions of ordinary differential equations. In *Computing Conference 2019 Springer series "Advances in Intelligent Systems and Computing"*, 2019.
[15] Eric Jones, Travis Oliphant, Pearu Peterson, and Others. {SciPy}: Open source scientific tools for {Python}.
[16] F Pedregosa, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. Scikit-learn: Machine Learning in {P}ython. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_13