Electronic Transactions on Numerical Analysis.
Volume 56, pp. 209–234, 2022.
Copyright © 2022, Kent State University.
ISSN 1068–9613.
DOI: 10.1553/etna_vol56s209

ETNA
Kent State University and
Johann Radon Institute (RICAM)

A HYBRID OBJECTIVE FUNCTION FOR ROBUSTNESS OF ARTIFICIAL NEURAL NETWORKS—ESTIMATION OF PARAMETERS IN A MECHANICAL SYSTEM[*]

JAN SOKOLOWSKI[†‡], VOLKER SCHULZ[†], HANS-PETER BEISE[§], AND UDO SCHROEDER[‡]

Abstract. In several studies, hybrid neural networks have proven to be more robust against noisy input data compared to plain data driven neural networks. We consider the task of estimating parameters of a mechanical vehicle model based on acceleration profiles. We introduce a convolutional neural network architecture that given sequential data, is capable to predict the parameters for a family of vehicle models that differ in the unknown parameters. This network is trained with two objective functions. The first one constitutes a more naive approach that assumes that the true parameters are known. The second objective incorporates the knowledge of the underlying dynamics and is therefore considered as hybrid approach. We show that in terms of robustness, the latter outperforms the first objective on unknown noisy input data.

Key words. system identification, parameter estimation, convolutional neural networks, sequential data, prediction robustness, mathematical modelling, dynamical systems

AMS subject classifications. 68T07, 93B30, 34A30

1. Introduction.

Physical, biological, and chemical models are important tools in nearly all fields of the engineering disciplines. Often, a major barrier for their application in practical cases is the lack of sufficient knowledge about the actual parameters [15].

As a consequence, a lot of technical knowledge in terms of well established equations cannot be exploit. The resulting problems to determine unknown parameters of a mathematical system from observations are typically described in the field of *System Identification* [5, 17].

In this work, we consider the case that a linear dynamical system is given in terms of a multi-dimensional ordinary differential equation. More specifically, we address the challenge of identification of parameters in a mechanical vehicle model, that is a coupled mass-spring-damper system [21].

The practical application that we are targeting at is as follows. Given a family of such models, we assume that only the parameters that scale with the occupant's mass are unknown and the remaining parameters are fixed. We want to create a common model that predicts the unknown parameters based on acceleration profiles induced by realistic but randomly generated road profiles.

Deep neural networks have proven to bear great potential in many complex tasks. The remarkable improvements in computer vision [16, 19, 36] and natural language processing [9, 23] are undoubtedly among the most famous achievements in this context. In the course of this progress, deep neural networks have successfully applied in various other disciplines like reinforcement learning [25] or practical applications in health informatics [33] and the prediction of energy consumption [22].

In a considerable line of publications, researchers attempted to tackle problems emerging in the context of physical equations. Mostly, data driven approaches can be used to generate approximative functions of a PDE structure [34] or to simulate the dynamical behaviour of

[*]Received September 30, 2020. Accepted April 7, 2021. Published online on March 14, 2022. Recommended by Axel Klawonn.

[†]Department of Mathematics, Trier University, Trier. ({s4jasoko, volker.schulz}@uni-trier.de)
[‡]Department of Basics & Mathematical Models, IEE S.A. Bissen.
({jan.sokolowski, udo.schroeder}@iee.lu)
[§]Dept. of Computer Science, Trier University of Applied Sciences, Trier.
(h.beise@hochschule-trier.de)

time-dependent ODEs [28, 35]. An appropriate network structure that fits the nature of the considered problem is often the key to successful results [7, 40].

A common challenge naturally occurring in those works, is the open question of how to combine neural networks with prior knowledge. That is here, the understanding of the underlying physical laws [8, 27, 30, 31] that define the connection of the neural network's input (observations) and the output (system properties).

In the present work, we use convolutional neural networks to process acceleration profiles. In that respect, we follow [39] and [41]. By doing so, we show that, based on simulated data, this kind of network can predict the unknown parameters. Our work is mostly related to [11, 3, 29].

This work provides the following contributions: We compare the performance of a convolutional neural network architecture for two different optimization processes. For both training processes, the target is to approximate a subset of the parameters of a system matrix that describes a set of ordinary differential equations. The first naive approach uses the true coefficients of the system matrix as labels, the second one recomputes the input data to indirectly approximate the parameters that are hidden within the data. We observe improved robustness against noisy test samples when using the second approach for neural network training.

The paper is organized as follows: We discuss a methodology to compute an appropriate dataset that can be used to (partially) identify the system matrix of the underlying differential equations in Section 2. Therefore, we describe, how to model the displacement of the road (Section 2.1) that can be used to compute the displacement of a passenger in an approximative vehicle model (Section 2.2), mathematically defined via a system of second order ordinary differential equations. Structure-preserving numerical algorithms like semi-implicit Euler methods (Section 2.3) can then help to generate synthetic datasets, consisting of the discrete acceleration profiles for the system of second order ODE. Then, this sequential data can be used and processed by convolutional neural networks (Section 2.4), using multiple input channels to achieve good approximations of the true system parameters in the output layer. The concept of Section 2 is validated in Section 3, using a neural network to predict the missing parameters that are necessary to describe the acceleration of the observable states. We can further assume that the true parameters are known for the training process (Section 3.1) as a labelled learning approach and compare it to unlabelled learning; Section 3.2. For the unlabelled approach, the output of the neural network is used to reproduce the acceleration in the neural network's output space. Then, the true parameter values should result from minimizing the distance of true and reproduced acceleration. We can compare the performance of these two objectives for clean training and clean test data to clean training and noisy test data; Section 3.3. Finally, we can draw a conclusion, when to prefer a labelled or an unlabelled approach with prior knowledge in Section 4.

2. Methodology.

We use a general non-homogeneous system of ordinary differential equations, mathematically defined by

$$\dot{\rho}(t) = A \cdot \rho(t) + f(t), \tag{2.1}$$

where $\rho \in \mathcal{C}^1([t_0, t_N], \mathbb{R}^d)$, with $N, d \in \mathbb{N}$, is a multi-dimensional time-dependent state variable and the derivative with respect to time is denoted by $\dot{\rho}(t) = \frac{\mathrm{d}\rho(t)}{\mathrm{d}t}$, an exterior term $f \in \mathcal{C}([t_0, t_N], \mathbb{R}^d)$ that reacts on the dynamical system and $A \in \mathbb{R}^{d\times d}$ the so called system matrix with $\mathbb{T} := [t_0, t_N] \subset [0, \infty)$. We, therefore, discuss in this section, how to model a dynamical system of coupled rigid bodies, as mathematically described by (2.1) in order to develop appropriate data and a sufficient neural network architecture for robust parameter estimation [14, 26, 32] for elements of the system matrix.

Appropriate data samples that represent a realisation of (2.1) for varying system matrices A can be computed by numerically solving the differential equation. Accordingly, we choose a physical model that can accurately be described by a simple system of second order ODEs. This system of second order ODEs can easily be reduced to a first order ODE system, which can in general be described using a representation as given by (2.1). Using a structure-preserving numerical solution algorithm then helps to generate data samples with a discrete representation of the dynamical behaviour as described by the underlying ODEs.

In this context, we use acceleration that can physically be described in terms of state and velocity with constant coefficients.

2.1. Road modelling. Before describing the dynamical system itself, we discuss in detail, how an appropriate non-homogeneous term $f(t)$ with $t \in [t_0, t_N]$ can be derived in a meaningful way for a mechanical system. The following description of modelling road profiles complient to ISO 8606 standard is based on [37].

Assume the distance, a car reaches on a specific road with absolute constant velocity, can be described by the continuous interval $\mathbb{S} := [s_0, s_N] \subset [0, \infty)$, where s_N is the maximum distance with respect to the starting point s_0. We define the state $r : \mathbb{S} \to \mathbb{R}$ that describes the displacement of the road at a point s via:

$$r(s) = \sum_{i=1}^{M} A_i \sin(\omega_i s - \varphi_i),$$

where $M \in \mathbb{N}$ is the number of relevant frequencies and the amplitude

$$A_i = \sqrt{\Phi(\omega_i)\left(\frac{\Delta\omega}{\pi}\right)}, \ i = 1, 2, \ldots, M$$

depends on the degree of roughness

$$\Phi(\omega_i) = \Phi(\omega_0)\left(\frac{\omega_i}{\omega_0}\right)^{-2}, \ i = 1, 2, \ldots, M$$

with

$$\Phi(\omega_0) = 2^k \cdot 10^{-6}$$

and $k \in \{0, 2, 4, 6, 8\}$. We say that the road is of class A, if $k = 0$, of class B, if $k = 2$, up to class E with $k = 8$. It is obvious then that a higher value for k results in a higher general amplitude of the road displacement $r(s)$ for all $s \in \mathbb{S}$ and therefore describes a road with a higher degree of roughness. Furthermore, the frequency domain is defined by the vector $\omega = (\omega_0, \omega_1, \ldots, \omega_M)^T$ with $\omega_0 = 1$, $\omega_1 = 0.02\pi$, $\omega_M = 6\pi$, and $\omega_i = \omega_1 + (i-1) \cdot \Delta\omega$ for $i \in \{2, 3, \ldots, M-1\}$, where $\Delta\omega = \frac{\omega_M - \omega_1}{M-1}$. The phase is given by realisations φ_i with $i \in \{1, 2, \ldots, M\}$ of a uniformly distributed random variable $\varphi \sim \mathcal{U}([0, 2\pi))$.

In order to discuss time-dependent dynamical models, we need to switch from a constantly increasing distance domain described by set $\mathbb{S}$ to a time domain $\mathbb{T}$. Therefore, we assume that the vehicle drives with constant velocity $v > 0$. It is obvious that for all $s \in \mathbb{S}$, we have $s = t \cdot v$ for all $t \in [t_0, t_N]$. And consequently, the road profile can be defined time-dependently by

$$r(t) = \sum_{i=1}^{M} A_i \sin(\omega_i t v - \varphi_i). \tag{2.2}$$

The state as given by (2.2) can now be used to induce a force as described by (2.1).

2.2. Quarter-Car-Model. In a less formal way, the vehicle model can be described as follows: A vehicle, as schematically given by Figure 2.1, drives along a specified road, whose displacement at a time $t \in [t_0, t_N]$ is given by a real valued scalar term $r(t)$. This displacement drives the dynamics of the wheel suspensions $x(t)$ and from there moves the remaining components $y(t)$ and $z(t)$ in the coupled system. The states are connected hierarchically by elements called springs and dampers. Regarding (2.3)–(2.5), we see that the number of coefficients corresponds to the number of connected springs and dampers as shown in Figure 2.1. Due to the coupled structure of the model, the rigid bodies are not reacting simultaneously, but following a hierarchical, time-dependent structure.

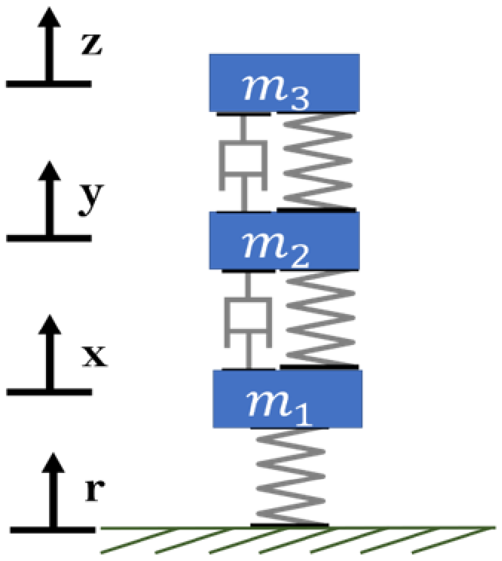

FIG. 2.1. *Scheme of a Quarter-Car-Model: The approximative model of a car consists of the wheel suspensions (mass m_1 with state x), the car body (mass m_2 with state y) and the passenger on seat (mass m_3 with state z). The states are given by vertical displacement over time. The rigid masses are coupled using a total number of three springs and two dampers. A dynamical behaviour is experienced by the displacement of the road r that results in a movement of the coupled mass-spring-damper system.*

In this section we introduce a system of second order ordinary differential equations that can be employed as a Quarter-Car-Model (QCM) [12, 20], which constitutes an approximation of a Half-Car-Model [2] or Full-Car-Model [24]. Our parameters are taken from [21]. The accelerations at a time $t \in [t_0, t_N] \subset [0, \infty)$ of the three rigid bodies of the QCM can then be described by the following equations

$$\ddot{z}(t) = -\frac{C_3}{m_3}\big(\dot{z}(t) - \dot{y}(t)\big) - \frac{K_3}{m_3}\big(z(t) - y(t)\big), \tag{2.3}$$

$$\begin{aligned}\ddot{y}(t) = -\frac{C_3}{m_2}\big(\dot{y}(t) - \dot{z}(t)\big) - \frac{C_2}{m_2}\big(\dot{y}(t) - \dot{x}(t)\big)& \\ - \frac{K_3}{m_2}\big(y(t) - z(t)\big) - \frac{K_2}{m_2}\big(y(t) - x(t)\big)&,\end{aligned} \tag{2.4}$$

$$\ddot{x}(t) = -\frac{C_2}{m_1}\big(\dot{x}(t) - \dot{y}(t)\big) - \frac{K_2}{m_1}\big(x(t) - y(t)\big) - \frac{K_1}{m_1}\big(x(t) - r(t)\big), \tag{2.5}$$

where C_2 and C_3 are the damping constants with $C_2 = 4741\ \frac{\mathrm{Ns}}{\mathrm{m}}$ and $C_3 = 615\ \frac{\mathrm{Ns}}{\mathrm{m}}$, K_1, K_2, and K_3 are the spring constants with $K_1 = 40\,000\ \frac{\mathrm{N}}{\mathrm{m}}$, $K_2 = 149\,171\ \frac{\mathrm{N}}{\mathrm{m}}$, and $K_3 = 98\,935\ \frac{\mathrm{N}}{\mathrm{m}}$ and $m_1 = 145$ kg is the mass of the wheel suspensions, $m_2 = 2160$ kg the mass of the car body and $m_3 \in \{50, 51, \dots, 200\}$ kg the mass of the passenger plus the seat's mass. A more complex dynamical human model can be developed following [1] but the simple version is sufficient for the observations considered in this approach. Furthermore, the states x, y, and z describe the relative displacement of the rigid bodies with masses m_1 for x, m_2 for y, and m_3 for z. The velocities are given by $\dot{x}(t) = \frac{\mathrm{d}x(t)}{\mathrm{d}t}$, $\dot{y}(t) = \frac{\mathrm{d}y(t)}{\mathrm{d}t}$, and $\dot{z}(t) = \frac{\mathrm{d}z(t)}{\mathrm{d}t}$ and finally the accelerations by $\ddot{x}(t) = \frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2}$, $\ddot{y}(t) = \frac{\mathrm{d}^2y(t)}{\mathrm{d}t^2}$, and $\ddot{z}(t) = \frac{\mathrm{d}^2z(t)}{\mathrm{d}t^2}$. The term $r(t)$ describes the displacement of the road as defined in Section 2.1.

Referring back to the initial statement that a dynamical system can be represented by (2.1), it is obvious to see that the system being described by (2.3)–(2.5) can be equivalently written in the form

$$\underbrace{\begin{pmatrix} \ddot{z}(t) \\ \ddot{y}(t) \\ \ddot{x}(t) \\ \dot{z}(t) \\ \dot{y}(t) \\ \dot{x}(t) \end{pmatrix}}_{=:\dot{\rho}(t)} = \underbrace{\begin{bmatrix} -\frac{C_3}{m_3} & \frac{C_3}{m_3} & 0 & -\frac{K_3}{m_3} & \frac{K_3}{m_3} & 0 \\ \frac{C_3}{m_2} & -\frac{C_2+C_3}{m_2} & \frac{C_2}{m_2} & \frac{K_3}{m_2} & -\frac{K_2+K_3}{m_2} & \frac{K_2}{m_2} \\ 0 & \frac{C_2}{m_1} & -\frac{C_2}{m_1} & 0 & \frac{K_2}{m_1} & -\frac{K_1+K_2}{m_1} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}}_{=:A} \underbrace{\begin{pmatrix} \dot{z}(t) \\ \dot{y}(t) \\ \dot{x}(t) \\ z(t) \\ y(t) \\ x(t) \end{pmatrix}}_{=:\rho(t)} + \underbrace{\begin{pmatrix} 0 \\ 0 \\ \frac{K_1}{m_1} r(t) \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{=:f(t)}. \tag{2.6}$$

Therefore, equation (2.6) can be seen as a system of first order ordinary differential equations. We assume that the states have a continuous dynamical behaviour, thus the system can (numerically) be solved, if the initial values

$$\rho(t_0) = (\dot{z}(t_0), \dot{y}(t_0), \dot{x}(t_0), z(t_0), y(t_0), x(t_0))^T = (\dot{z}_0, \dot{y}_0, \dot{x}_0, z_0, y_0, x_0)^T$$

are known. Hold in mind that due to the hierarchical structure of the model, it can be preferable to use a structure preserving integration scheme, instead of a simple forward Euler integration method. Hence, we use a symplectic (semi-implicit) Euler integration scheme.

2.3. Dataset. To generate a synthetic dataset, the above ODE system, defined by (2.6), is numerically solved using a symplectic Euler scheme for geometric integration [13]. This guarantees that the interdependent relation between the coupled states is considered when computing the approximate solution of the differential equations. For simplification, we set $\dot{z} = w$, $\dot{y} = v$, and $\dot{x} = u$, which then results in an exact description of (2.6) as a first order non-homogeneous system of ODE. We can then use the following iterative structure to come to an appropriate numerical solution of our system:

$$\begin{aligned} u_{k+1} &= u_k + h\left(\tfrac{C_2}{m_1} v_k - \tfrac{C_2}{m_1} u_k + \tfrac{K_2}{m_1} y_k - \tfrac{K_1+K_2}{m_1} x_k + \tfrac{K_1}{m_1} r_k\right) \\ v_{k+1} &= v_k + h\left(\tfrac{C_3}{m_2} w_k - \tfrac{C_2+C_3}{m_2} v_k + \tfrac{C_2}{m_2} u_{k+1} + \tfrac{K_3}{m_2} z_k - \tfrac{K_2+K_3}{m_2} y_k + \tfrac{K_2}{m_2} x_k\right) \\ w_{k+1} &= w_k + h\left(-\tfrac{C_3}{m_3} w_k + \tfrac{C_3}{m_3} v_{k+1} - \tfrac{K_3}{m_3} z_k + \tfrac{K_3}{m_3} y_k\right) \\ x_{k+1} &= x_k + h u_{k+1} \\ y_{k+1} &= y_k + h v_{k+1} \\ z_{k+1} &= z_k + h w_{k+1} \end{aligned} \tag{2.7}$$

We use a discrete time scheme to compute the approximate solution with the above equations. Note that for instance $u_k = u(t_k)$ with $t_k = kh$, where $k \in \{0, \dots, N\}$ with $N \in \mathbb{N}$ the number of discrete time steps and $h = \frac{t_N - t_0}{N}$ the step-width. The same scheme is analogously applied to $v(t_k), w(t_k), x(t_k), y(t_k), z(t_k)$ and also to the road displacement $r(t_k)$ for all $k \in \{0, \dots, N\}$. The above iterative structure ensures a representation of the dynamical behaviour of a QCM.

We can now generate a synthetic dataset using the above described symplectic scheme and further assume that the dynamical system is in equilibrium at $t = t_0$. Consequently, we initially have $u_0 = v_0 = w_0 = x_0 = y_0 = z_0 = 0$, interpreted as relative states and velocities to the corresponding rigid masses. We further assume that the following experiment can be described by the dataset: The spring and damping parameters are equal for all samples of the dataset. This means that we always regard the same vehicle for all samples. The shape of the road's displacement is equal for a total number of $L_m \in \mathbb{N}$ samples. For each sample, the mass of the seat and the passenger is drawn from a uniformly distributed random variable constraint by the mass being integer-valued.

We compute $L_r \in \mathbb{N}$ random shapes of the road's displacement as described in Section 2.1 for discrete time steps t_k. Deviations from profile to profile are guaranteed by the randomly drawn phase for all $M \in \mathbb{N}$ sine waves for all time steps. In addition to that, the degree of roughness, characterized by $\Phi(\omega_0)$ is also randomly drawn with respect to one of the five acceptable classes A–E. Then, for one of the road profiles r^j, $j \in \{1, 2, \dots, L_r\}$, we generate m_3^{ij}, $i \in \{1, 2, \dots, L_m\}$ and apply the scheme given by (2.7). Thus, we get $\hat{u}^{ij}, \hat{v}^{ij}, \hat{w}^{ij}, \hat{x}^{ij}, \hat{y}^{ij}, \hat{z}^{ij} \in \mathbb{R}^N$ as discrete solution of the non-homogeneous system with road profile r^j and mass m_3^{ij}.

Then, the discrete accelerations $\hat{\ddot{z}}^{ij}$ and $\hat{\ddot{y}}^{ij}$ can be computed using (2.3) and (2.4) by

$$\begin{aligned} \hat{\ddot{z}}^{ij} &= -\frac{C_3}{m_3^{ij}} \left(\hat{w}^{ij} - \hat{v}^{ij}\right) - \frac{K_3}{m_3^{ij}} \left(\hat{z}^{ij} - \hat{y}^{ij}\right), \\ \hat{\ddot{y}}^{ij} &= -\frac{C_3}{m_2} \left(\hat{v}^{ij} - \hat{w}^{ij}\right) - \frac{C_2}{m_2} \left(\hat{v}^{ij} - \hat{u}^{ij}\right) - \frac{K_3}{m_2} \left(\hat{y}^{ij} - \hat{z}^{ij}\right) - \frac{K_2}{m_2} \left(\hat{y}^{ij} - \hat{x}^{ij}\right). \end{aligned} \tag{2.8}$$

The discrete accelerations $\hat{\ddot{z}}^{ij}$ and $\hat{\ddot{y}}^{ij}$ can then be interpreted to be recordings of two g-sensors, one measuring acceleration of the seat and one measuring acceleration of the car body in vertical direction. Therefore, computing the system's accelerations for different passenger's masses, simulates a car driving on different roads with different passengers.

We can then define our labelled dataset by

$$\begin{aligned} \mathbb{X} &= \left\{ \left(\hat{\ddot{z}}^{ij}, \hat{\ddot{y}}^{ij}\right) \in \mathbb{R}^{N \times 2} \mid i \in \{1, 2, \dots, L_m\}, j \in \{1, 2, \dots, L_r\} \right\}, \\ \mathbb{M} &= \left\{ m^{ij} \in \{50, 51, \dots, 200\} \mid i \in \{1, 2, \dots, L_m\}, j \in \{1, 2, \dots, L_r\} \right\}, \end{aligned}$$

both with a cardinality of $L = L_r \cdot L_m$ samples per set. Then, the labelled dataset can be described by $(\mathbb{X}, \mathbb{M}) \subset \mathbb{R}^{L \times N \times 2} \times \mathbb{R}^L$. Hold in mind that $(\hat{\ddot{z}}^{ij}, \hat{\ddot{y}}^{ij})$ for $i \in \{1, 2, \dots, L_r\}$ corresponds to the discrete vertical accelerations of the Quarter-Car-Model induced by the road profile r^j for $j \in \{1, 2, \dots, L_r\}$. Then, we can separate the index set $\{1, 2, \dots, L_r\}$ that identifies the road profiles using an integer-valued separator $L_b \le L_r$ such that we can define the labelled training dataset by

$$\begin{aligned} \mathbb{X}_{\text{train}} &= \left\{ \left(\hat{\ddot{z}}^{ij}, \hat{\ddot{y}}^{ij}\right) \in \mathbb{R}^{N \times 2} \mid i \in \{1, 2, \dots, L_m\}, j \in \{1, 2, \dots, L_b\} \right\}, \\ \mathbb{M}_{\text{train}} &= \left\{ m^{ij} \in \{50, 51, \dots, 200\} \mid i \in \{1, 2, \dots, L_m\}, j \in \{1, 2, \dots, L_b\} \right\}, \end{aligned}$$

and the labelled test dataset by

$$\begin{aligned}\mathbb{X}_{\text{test}} &= \left\{ \left(\hat{\ddot{z}}^{ij}, \hat{\ddot{y}}^{ij} \right) \in \mathbb{R}^{N \times 2} \mid i \in \{1, 2, \dots, L_m\}, j \in \{L_b + 1, L_b + 2, \dots, L_r\} \right\}, \\ \mathbb{M}_{\text{test}} &= \left\{ m^{ij} \in \{50, 51, \dots, 200\} \mid i \in \{1, 2, \dots, L_m\}, j \in \{L_b + 1, L_b + 2, \dots, L_r\} \right\}.\end{aligned}$$

Now it can be guaranteed that no road profile used to generate training data simultaneously is used for the test dataset. As far as our investigations are concerned, we choose the parameters $N = 6000$, $L_r = 100$, $L_m = 100$, and $L_b = 8000$. Consequently the dataset simulates a Quarter-Car-Model on a total number of 100 roads, where for each road there are 100 passengers with arbitrary weight in the previously defined interval. A total number of 80 roads are used for generating the training dataset and 20 for the test dataset. Therefore, for our following investigations the relation of training to test samples is $4 : 1$.

2.4. Neural Network. The previous section described precisely, how an appropriate sequential dataset representing realizations of a coupled mass-spring-damper system can be generated using a symplectic integration technique like the semi-implicit Euler method. Convolutional neural networks have shown to achieve comparable results like recurrent neural networks [4] for sequential data processing [10]. In addition, convolutional neural networks with multi input channels can easily be used to process a set of sequential data in parallel. Therefore, we also consider a convolutional neural network architecture to analyze the dataset we developed in Section 2.3.

Comparable to image recognition tasks, the pair $(\hat{\ddot{z}}, \hat{\ddot{y}})$, which either belongs to the training or the test set as defined in Section 2.3, can technically be processed like one row of an MNIST sample [6], using a one-dimensional convolutional operation. In this case, one dimensional means that the convolution is applied only to one direction.

We use augmentation strategies combined with batch-optimization to achieve a better generalization performance for unknown test samples. Therefore, instead of choosing the complete sample $(\hat{\ddot{z}}, \hat{\ddot{y}}) \in \mathbb{R}^{N \times 2}$, where the discrete values of the acceleration are by definition as described in (2.7) restricted to the index set $I := \{0, 1, \dots, N - 1\}$, we randomly choose subintervals of the whole sample as an augmentation method. Then, a subset $I_l \subset I$ can be defined using a smaller frame of $\bar{N} = 500$ discrete steps with randomly drawn starting point index $l \in \{0, 1, \dots, 5500\}$, due to 5500 is the maximum index number, such that a vector of size 500 can be described within the index set I.

In detail, the subinterval is given by $I_l = \{l, l + 1, \dots, l + \bar{N} - 1\}$. Therefore, following the notation of Section 2.3, we can define the randomly chosen subset of the input by

$$\left(\hat{\ddot{z}}, \hat{\ddot{y}} \right)_{I_l} = \begin{bmatrix} \hat{\ddot{z}}_l & \hat{\ddot{y}}_l \\ \hat{\ddot{z}}_{l+1} & \hat{\ddot{y}}_{l+1} \\ \vdots & \vdots \\ \hat{\ddot{z}}_{l+\bar{N}-1} & \hat{\ddot{y}}_{l+\bar{N}-1} \end{bmatrix}.$$

Assume that there are two unknown entries within the system matrix A, given by the two-dimensional vector

$$p = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} \frac{C_3}{m_3} \\ \frac{K_3}{m_3} \end{pmatrix},$$

consisting of the two parameters that describe the acceleration $\ddot{z}$ of the passenger in (2.6).

Then, a deep convolutional neural network can be defined by the following function

$$\begin{aligned} g_\theta : \mathbb{R}^{500\times 2} &\longrightarrow \mathbb{R}^2 \\ \left(\hat{\ddot{z}}, \hat{\ddot{y}}\right)_{I.} &\longmapsto \begin{pmatrix} \tilde{p}_1 \\ \tilde{p}_2 \end{pmatrix}, \end{aligned}$$

where θ describes the set of weights and biases of the convolutional neural network, $\tilde{p}_1$ is the prediction of the first parameter p_1 and analogously $\tilde{p}_2$ is the prediction for the second parameter p_2. This means that there are two unknown parameters within our system matrix A, we want the neural network to predict from the dataset.

Therefore, we use the following network structure, also described by Figure 2.2: The first layer is a convolutional one with input dimension according to the domain of g_θ with 100 filters of size 50 and the hyperbolic tangent as activation function. This is followed by a second convolutional layer with another 100 filters but of size 10. The output is then flattened and further processed using three fully-connected layers with 100, 10, and 2 neurons with fitting weighting matrices and biases. For the output layer, we map the computational results of weighting and biasing to the absolute value in order to guarantee positive predictions of the parameters.

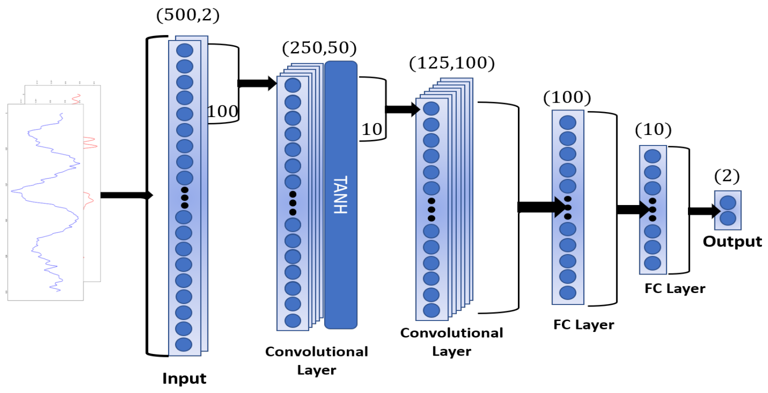

FIG. 2.2. *Neural network for parameter prediction, consisting of one convolutional layer with* 50 *filters of size* 100 *and hyperbolic tangent as activation, one convolutional layer with* 100 *filters of size* 50*, followed by one fully-connected layer with* 100 *neurons, one fully-connected layer with* 10 *neurons and a fully-connected layer with* 2 *neurons as output layer.*

2.5. Loss functions. Following the notation of the previous section, the output of the neural network can be denoted by $g_\theta(\hat{\ddot{z}}, \hat{\ddot{y}}) = \left(\tilde{p}_1, \tilde{p}_2\right)^T = \tilde{p}$. Therefore, the neural network's output can be separated into the prediction for the first parameter $\tilde{p}_1 = g_{\theta;1}(\hat{\ddot{z}}, \hat{\ddot{y}})$ and for the second one $\tilde{p}_2 = g_{\theta;2}(\hat{\ddot{z}}, \hat{\ddot{y}})$, respectively. We will now introduce two loss functions J_L and J_U. The first of which measures the deviation of system parameter predictions to the correct parameter, i.e., labels, and the second measures the deviation of the data reconstruction that results from mapping the predicted system parameters to the input data. Both training processes can be compared later on, as they use the same neural network architecture.p Since the first loss requires labels, we will refer to it as *labelled loss* and as the second one does not require the parameter labels, we will consequently refer to it as *unlabelled loss*.

Labelled Loss J_L. For the first objective, we assume that the true underlying parameter values are known for the training process. Therefore, a neural network to predict the two parameters can be trained using the labelled objective

$$J_L(p, g_\theta(\hat{\ddot{z}}, \hat{\ddot{y}})) = \sum_{i=1}^{d} |p_i - g_{\theta;i}(\hat{\ddot{z}}, \hat{\ddot{y}})|,$$

where for this special case we have $d = 2$ being the number of parameters to predict.

Unlabelled Loss J_U. As mentioned in the beginning, the second loss function intends to minimize the distance in the data space. We first explain, how acceleration data is generated based on the system parameters predicted by the neural network. Afterwards, we can explicitly define the unlabelled objective J_U. We know by (2.3) that the acceleration of the passenger can be computed by

$$\ddot{z} = -\frac{C_3}{m_3}(\dot{z} - \dot{y}) - \frac{K_3}{m_3}(z - y) = -p_1(\dot{z} - \dot{y}) - p_2(z - y).$$

We assume that the device recording the acceleration data delivers it without noise, or more realistic, we assume that the acceleration data already has been de-noised. Then, it is possible to get a discrete approximation of the velocities $\dot{y}$, $\dot{z}$ and consequently also approximations of the states y, z with a numerical integration scheme. For a discrete time frame $t_0 < \ldots < t_N$ with $h = t_{k+1} - t_k$ for all $k \in \{0, 1, \ldots, N-1\}$ the recorded acceleration is given by

$$\hat{\ddot{z}}_k = \ddot{z}(t_k)$$

and the same holds for the recorded acceleration of the car body $\hat{\ddot{y}}_k = \ddot{y}(t_k)$. Consequently, using a symplectic Euler integration scheme, we get $\hat{\dot{z}}_k$ for all $k \in \{1, 2, \ldots, N-1\}$ by

$$\hat{\dot{z}}_k = \hat{\dot{z}}_{k-1} + \int_{t_{k-1}}^{t_k} \hat{\ddot{z}}_{k-1}\, \mathrm{d}s = \hat{\dot{z}}_{k-1} + h \cdot \hat{\ddot{z}}_{k-1}$$

and analogously the state $\hat{z}_k$ by

$$\hat{z}_k = \hat{z}_{k-1} + \int_{t_{k-1}}^{t_k} \hat{\dot{z}}_k\, \mathrm{d}s = \hat{z}_{k-1} + h \cdot \hat{\dot{z}}_k,$$

which reduces the approximation error that follows from simple forward Euler integration [38]. The same scheme can be applied to compute the integrated values of the car body's acceleration $\hat{\ddot{y}}_k$. Then, using the output of the neural network for these specific values, we can make a prediction of the original acceleration, which corresponds to the computation of the discrete acceleration for the dataset in (2.8), by

$$\tilde{\ddot{z}}_k = -g_{\theta;1}(\hat{\ddot{z}}, \hat{\ddot{y}})(\hat{\dot{z}}_k - \hat{\dot{y}}_k) - g_{\theta;2}(\hat{\ddot{z}}, \hat{\ddot{y}})(\hat{z}_k - \hat{y}_k)$$

for all $k \in \{0, 1, \ldots, N-1\}$. Finally, we can define an objective for unlabelled learning by

$$J_U(\hat{\ddot{z}}, g_\theta(\hat{\ddot{z}}, \hat{\ddot{y}})) = \sum_{l \in I_l} |\hat{\ddot{z}}_l - (-g_{\theta;1}(\hat{\ddot{z}}, \hat{\ddot{y}})(\hat{\dot{z}}_l - \hat{\dot{y}}_l) - g_{\theta;2}(\hat{\ddot{z}}, \hat{\ddot{y}})(\hat{z}_l - \hat{y}_l))|^2,$$

where I_l is the randomly drawn index set described in Section 2.4. Comparable to an autoencoder neural network, the original data is rebuilt out of a data representation in a low-dimensional (latent) space [18], here being described by the parameter vector $\tilde{p}$.

3. Numerical examples.

In the preciding section, it is explained how neural networks are employed to predict a set of parameters of a dynamical system using partial observations of the system. In this section, we report results from numerical experiments minimizing the loss functions J_L and J_U for comparable tasks.

3.1. Labelled system identification. In a first experiment, the objective is to minimize J_L. Therefore we discuss the initial results of the training dataset before any of the network's parameters have been optimized. We statistically evaluate our methods, with respect to absolute and relative mean error with corresponding variance.

We predict the values of the parameters p_1 and p_2 for the training dataset with the help of a one-dimensional convolutional neural network, with objective J_L for training. Figure 3.1 and Figure 3.3 show the predictions and the true values for p_1 (Figure 3.1) and for p_2 (Figure 3.3), where Figure 3.2 and Figure 3.4 show the relative deviations between the true parameter values and the predictions coming from the output layer of the neural network. More precisely, as far as Figure 3.1 and Figure 3.3 are concerned, we sorted the training dataset according to the value of the true parameter. Thus, we can recognize numbers reaching from 1 to 8000 on the abscissa and a sequence of growing parameter values on the ordinate, represented by the green curve. This curve then describes the true parameter value for p_1 (Figure 3.1) and p_2 (Figure 3.3) for the entire training dataset. In contrast, the red points correspond to the network's prediction. Consequently, for each green point on the "curve" there is exactly one corresponding red point on the same vertical level.

Optimally, the red points should therefore fit the green line of the true parameter values. This property can be interpreted as prediction being equal to the underlying label of the input data.

Besides, Figure 3.2 and Figure 3.4 show the relative deviation of prediction to true parameter value plotted as histograms. The relative deviation of a true parameter p_j with respect to the neural network's prediction $\tilde{p}_j$ for $j \in \{1, 2\}$ is then given by

$$\frac{|p_j - \tilde{p}_j|}{p_j}.$$

The histograms then show the relative deviation on the abscissa, where the number of samples, that approximately lie within the same error range is shown on the ordinate. The four plots show the results at $i = 0$ optimization steps. Therefore, we actually cannot see any valuable results, but the initialization is adequate to get an impression, how the optimal solution should look like. Weights and biases are randomly initialized, therefore we also get random outputs of the neural network. Optimally, the figures in Figure 3.1 and Figure 3.3 should show an approximation of the red point cloud to the green label line, while the bars of the histograms of Figure 3.2 and Figure 3.4 should be close to zero on the x-axis, corresponding to a small error for all samples of the training set.

The results in Figure 3.5 and Figure 3.6 show the prediction of the parameters and the true parameter values for the entire training dataset after $i = 500\,000$ optimization steps, using Adam optimization for the training process with learning-rate $\eta = 0.001$ and batch-size $M = 100$. As already described for $i = 0$ optimization steps, while training, the red point cloud should converge to the green label line. As we can see from both Figure 3.5 as well as from Figure 3.7, the red cloud indeed comes closer to the optimal label values.

In addition also the histograms of the relative deviation, as can be seen in Figure 3.6 and Figure 3.8, show the expected result: The main samples have a relative deviation close to zero as can be seen by a left-skewed distribution of the relative deviation. Precisely, we have a relative mean deviation of $\mu = 0.062$ and a mean standard deviation of this error of $\sigma = 0.091$. For the second parameter, we have $\mu = 0.043$ and $\sigma = 0.083$.

The architecture of the convolutional neural network is therefore sufficient as far as the task of labelled learning for the given dataset is concerned.

Furthermore, we also want to discuss the generalization quality of the neural network. Therefore, we analyze the same plots as already discussed for the training data, but now for the unknown test dataset. Consequently, instead of 8000 samples, as shown on the abscissa for Figure 3.1, Figure 3.3, Figure 3.5, and Figure 3.7, there is a range of 2000 samples for the test data, as can be seen in Figure 3.9 and Figure 3.11. The results are similar to those of the training dataset, as far as prediction accuracy and maximum relative deviation (Figure 3.10 and Figure 3.12) are considered. For the first parameter p_1 we have on average $\mu = 0.060$ and $\sigma = 0.079$, where for p_2 we get $\mu = 0.042$ and $\sigma = 0.084$.

In summary, we conclude that the mean training and test error are close to each other, as well as the corresponding standard deviation. Therefore, the convolutional neural network generalizes well with the given architecture using objective function J_L for the data of our Quarter-Car-Model.

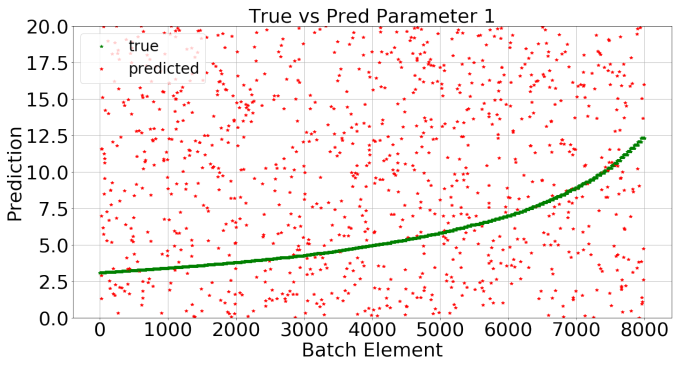

FIG. 3.1. *Index of the training samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_1 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 0$ w.r.t. the objective function J_L.*

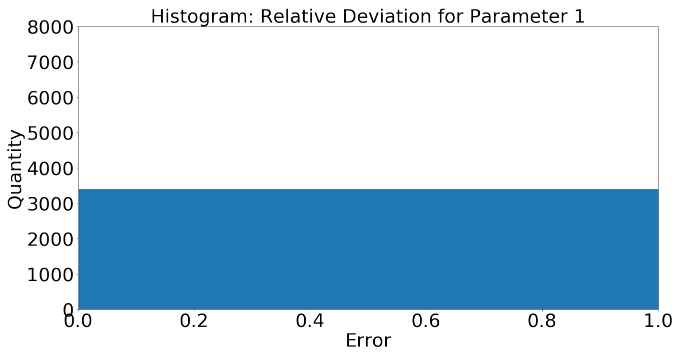

FIG. 3.2. *Relative error (abscissa) between neural network's prediction and true label for p_1 and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all training samples at optimization step $i = 0$ for the objective function J_L.*

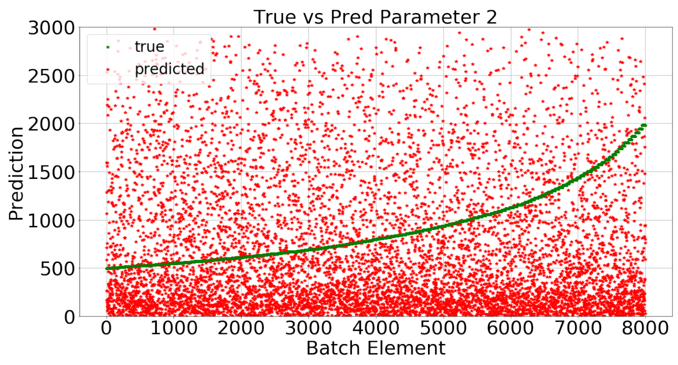

FIG. 3.3. *Index of the training samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_2 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 0$ w.r.t. the objective function J_L.*

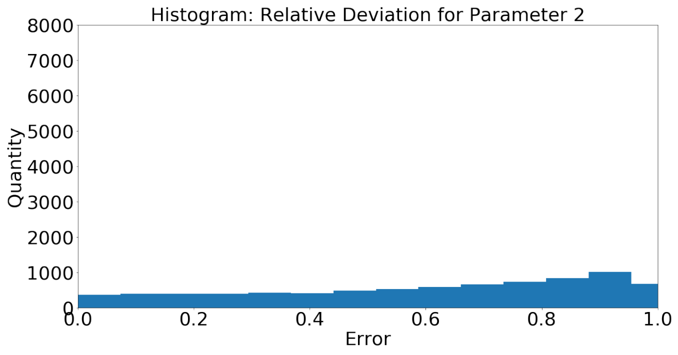

FIG. 3.4. *Relative error (abscissa) between neural network's prediction and true label for p_2 and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all training samples at optimization step $i = 0$ for the objective function J_L.*

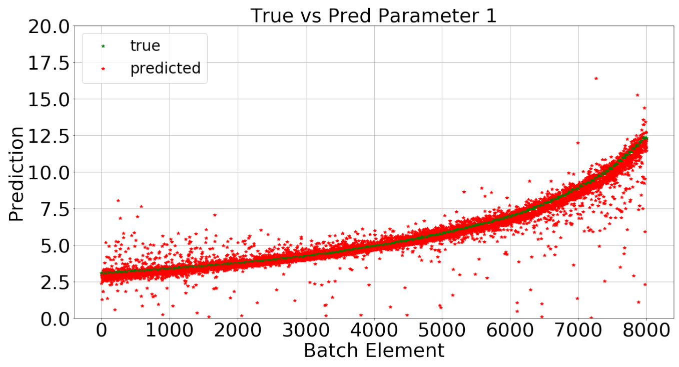

FIG. 3.5. *Index of the training samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_1 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 500\,000$ w.r.t. the objective function J_L.*

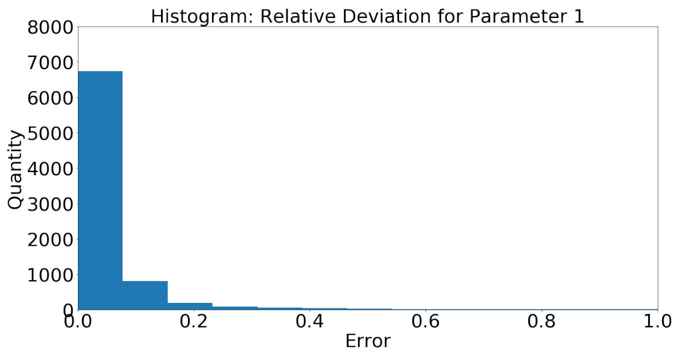

FIG. 3.6. *Relative error (abscissa) between neural network's prediction and true label for* p_1 *and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all training samples at optimization step* $i = 500\,000$ *for the objective function* J_L.

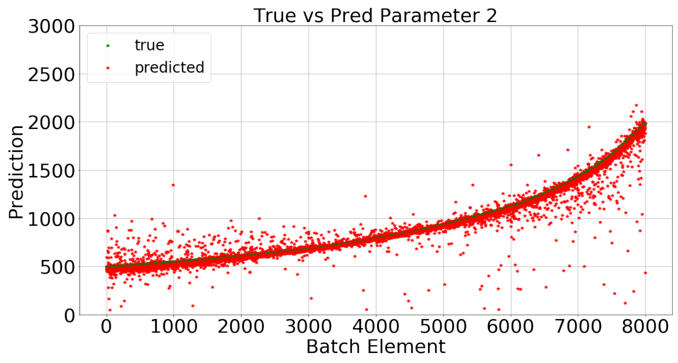

FIG. 3.7. *Index of the training samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter* p_2 *compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step* $i = 500\,000$ *w.r.t. the objective function* J_L.

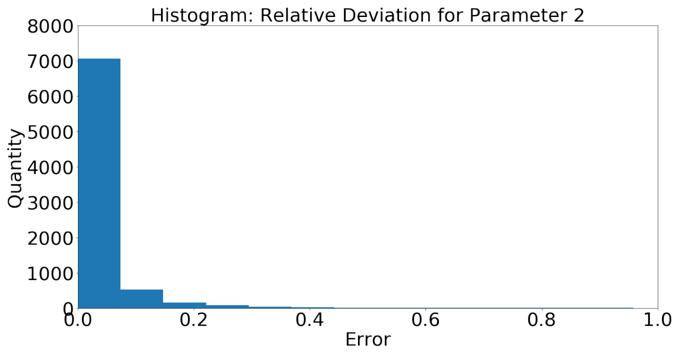

FIG. 3.8. *Relative error (abscissa) between neural network's prediction and true label for* p_2 *and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all training samples at optimization step* $i = 500\,000$ *for the objective function* J_L.

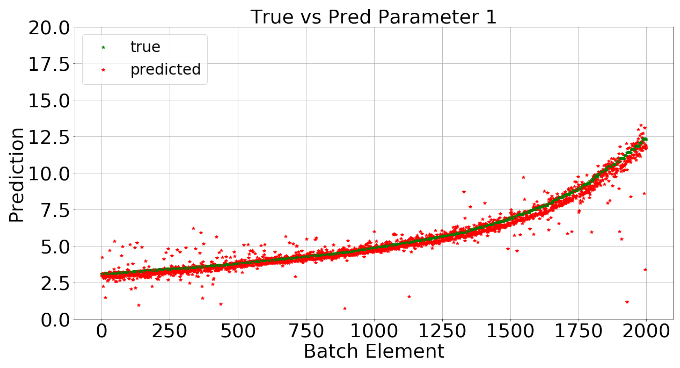

FIG. 3.9. *Index of the test samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_1 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 500\,000$ w.r.t. the objective function J_L.*

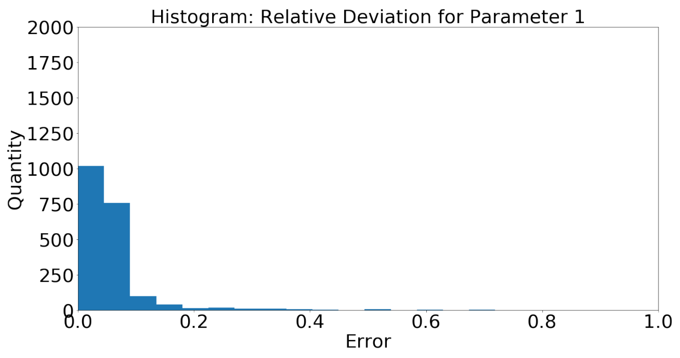

FIG. 3.10. *Relative error (abscissa) between neural network's prediction and true label for p_1 and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all test samples at optimization step $i = 500\,000$ for the objective function J_L.*

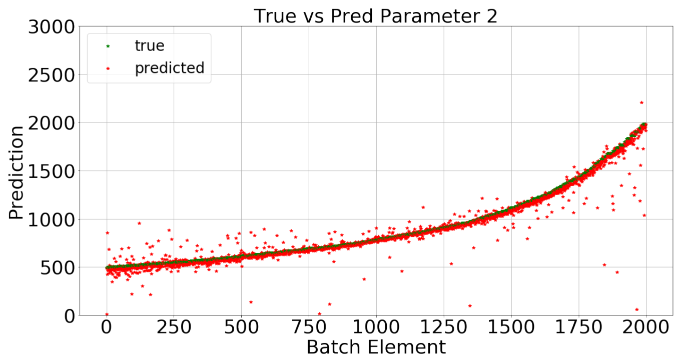

FIG. 3.11. *Index of the test samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_2 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 500\,000$ w.r.t. the objective function J_L.*

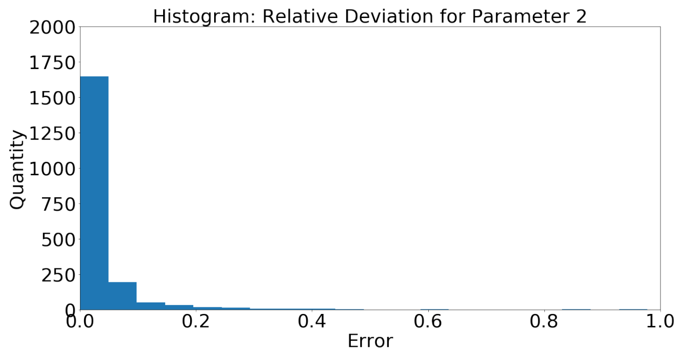

FIG. 3.12. *Relative error (abscissa) between neural network's prediction and true label for* p_2 *and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all test samples at optimization step* $i = 500\,000$ *for the objective function* J_L.

3.2. Unlabelled system identification. Let us now assume that the true values of the parameters p_1 and p_2 are not known for any sample of the training dataset. Nevertheless, we want to identify the unknown parameters via usage of our deep convolutional neural network. Therefore, as initially described, we use our unlabelled objective function J_U to minimize the squared difference of $\hat{\tilde{z}}_l$ and the reproduction $\tilde{\tilde{z}}_l$ for $l \in I_l$.

Compared to the labelled approach, we skip the initialization analysis. We directly have a look at the usual plots for our experiments for the entire training data after $i = 500\,000$ optimization steps. Again we use Adam optimization with the same learning-rate and batch-size as described for the first experiment. Here, the absolute deviation of parameter p_1 and p_2 are shown in Figure 3.13 and Figure 3.15, where the corresponding relative deviations are shown by the histograms in Figure 3.14 and Figure 3.16. Comparable to the first experiment, we see that the red points come closer to the green optimal label line. Although we can recognize that there is a larger deviation, when regarding large parameter values for both p_1 as well as p_2. Again, the precise values are given by $\mu = 0.102$ and $\sigma = 0.131$ for p_1 and $\mu = 0.082$ and $\sigma = 0.131$ for p_2. Consequently, the mean error for the training data is approximately 4% higher compared to the results of Section 3.1.

We can make similar observations, when comparing training and test. Again, the red points seem to approximately fit the label points with high deviations for large parameter values; Figure 3.17 and Figure 3.19. The deviation (Figure 3.18 and Figure 3.20) can be described by $\mu = 0.097$ and $\sigma = 0.127$ for the first parameter and $\mu = 0.076$ and $\sigma = 0.127$ for the second one.

To summarize the results, we observe that the values for μ and σ lie within an comparable range for the training as well as for the test data. Therefore, also minimizing the objective function J_U results in a good generalizability for the neural network. Nevertheless, the prediction quality is slightly worse with mean relative deviation of around 8–10% compared to objective J_L with 4–8%.

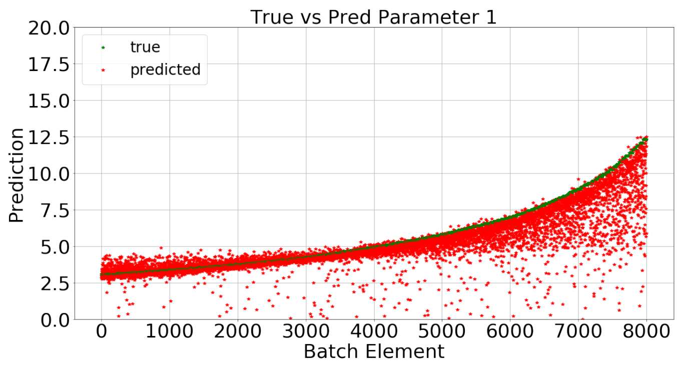

FIG. 3.13. *Index of the training samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_1 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 500\,000$ w.r.t. the objective function J_U.*

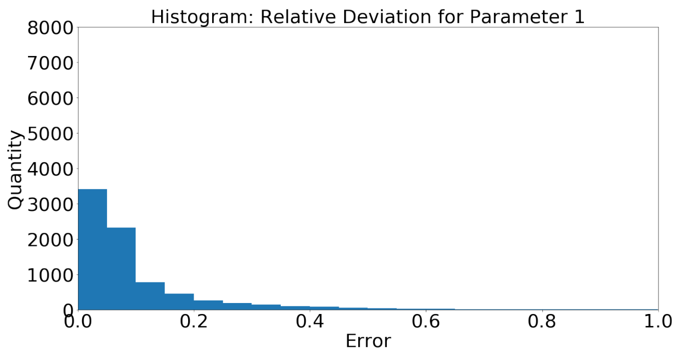

FIG. 3.14. *Relative error (abscissa) between neural network's prediction and true label for p_1 and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all training samples at optimization step $i = 500\,000$ for the objective function J_U.*

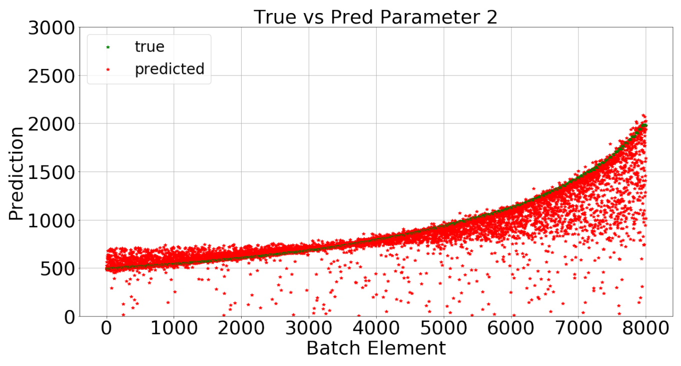

FIG. 3.15. *Index of the training samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_2 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 500\,000$ w.r.t. the objective function J_U.*

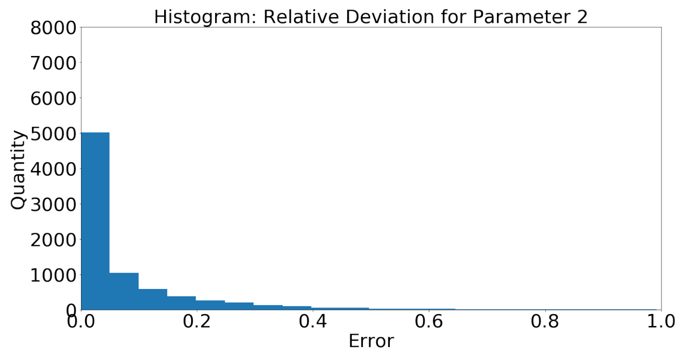

FIG. 3.16. *Relative error (abscissa) between neural network's prediction and true label for p_2 and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all training samples at optimization step $i = 500\,000$ for the objective function J_U.*

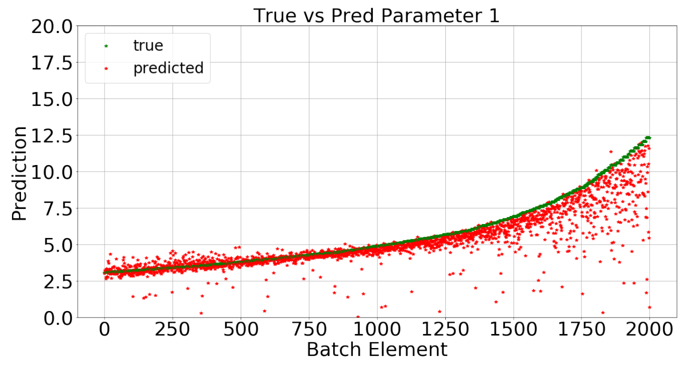

FIG. 3.17. *Index of the test samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_1 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 500\,000$ w.r.t. the objective function J_U.*

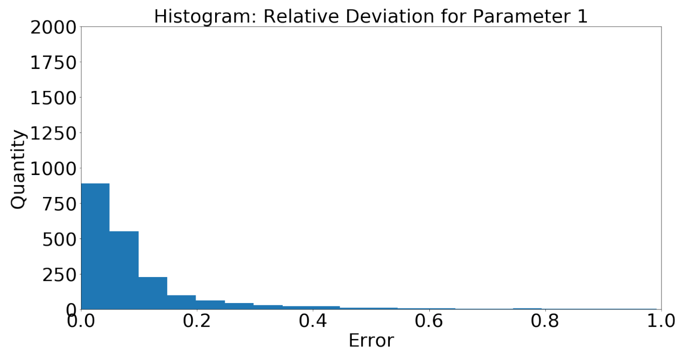

FIG. 3.18. *Relative error (abscissa) between neural network's prediction and true label for p_1 and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all test samples at optimization step $i = 500\,000$ for the objective function J_U.*

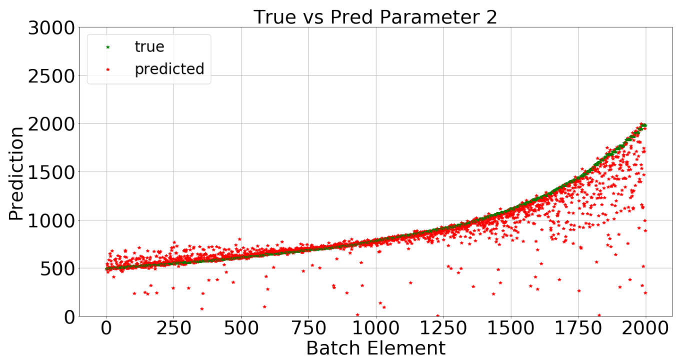

FIG. 3.19. *Index of the test samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter* p_2 *compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step* $i = 500\,000$ *w.r.t. the objective function* J_U.

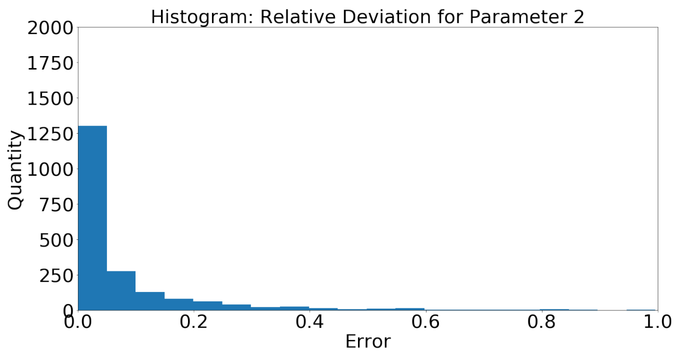

FIG. 3.20. *Relative error (abscissa) between neural network's prediction and true label for* p_2 *and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all test samples at optimization step* $i = 500\,000$ *for the objective function* J_U.

3.3. Robustness for noisy test data. For the experiments of the previous sections, we have always considered the training and test data to be clean or already de-noised. We have also discussed the performance of a convolutional neural network for time series, being trained with a labelled and an unlabelled objective function. For these two experiments, the labelled approach is superior in terms of training and test error compared to the unlabelled approach. We now want to consider the case, where the model, for both—labelled and unlabelled learning—is trained using de-noised acceleration data. Therefore, the data quality for the objective J_U is ensured, using numerical integration to approximate the values of the corresponding velocity and state.

Now assume that the acceleration data of the test dataset is noisy, meaning for each $(\hat{\ddot{z}}, \hat{\ddot{y}}) \in \mathbb{X}_{\text{test}}$ within our test data, there is one corresponding Gaussian noise $\xi \in \mathbb{R}^N$ for $\ddot{z}$ and one corresponding Gaussian noise $\nu \in \mathbb{R}^N$ for $\ddot{y}$ such that the input of the neural network is given by

$$(\bar{\ddot{z}}, \bar{\ddot{y}}) = \left(\hat{\ddot{z}} + \xi, \hat{\ddot{y}} + \nu\right).$$

We compare which approach is more adequate to process noisy test data. Usually, neural networks can react very sensitive to data noise. Therefore, we test both trained networks, at the one hand the labelled and on the other hand the unlabelled approach and compare the performance for the noisy test data with $\xi_k, \nu_k \sim \mathcal{N}(0, 0.01)$ for all $k \in \{0, 1, \dots, N-1\}$.

As can be seen for the objective function J_L and the test dataset (compare Figure 3.21–Figure 3.24), the output of the neural network is quite sensitive to the Gaussian noise added to the test dataset, as expected. We can recognize a decreased performance of the prediction quality. Although the most part of the red point cloud is concentrated around the green labels, we can see that there are more values spread compared to the clean test data in Section 3.1. As mentioned previously, we can recognize that the objective J_L can give highly precise values for the training data: For both parameters p_1 and p_2 we get mean relative deviations of 4–8%. Contrarily, the performance for the noisy test dataset is much worse in this case: For both p_1 and p_2 the test deviation lies between 26–28%. The generalizability for the convolutional neural network is for this specific case not given any more. There are significantly better results in predicting the values of the test data when the unlabelled objective J_U is used for the training procedure; compare Figure 3.25–Figure 3.28. For the training dataset, the unlabelled objective function results in an adjustment of the network's parameters to predict with a relative deviation of 8–10%. The noisy test data results in a relative deviation of around 16%. Consequently, the performance for the training data is relatively close to the performance of the test data and therefore the second objective is more robust than the labelled one.

The robustness of the second objective compared to the first one becomes obvious in Figure 3.29–Figure 3.32. In Figure 3.29, the mean relative deviations, for both parameter p_1 and parameter p_2, are shown at every $i = \lambda \cdot 100\,000$ iteration steps with $\lambda \in \{1, 2, 3, 4, 5\}$ for the training as well as for the test dataset using objective J_L. The straight lines represents the evaluation for the optimization process using the de-noised test data, the dashed line for the noisy test data. In an analogous way, Figure 3.31 displays the results for the mean standard deviation also using objective J_L. As can be seen in Figure 3.29 and Figure 3.31, the performance lines are close to each other in case of the clean test dataset. When considering the noisy test data for parameter p_1 (violet dashed line) and p_2 (cyan dashed line) there is a large gap compared to the training deviation.

Therefore, we again see that the approach using J_L works well as far as generalizability for de-noised test data is concerned but fails for noisy samples.

In contrast to that, we can have a look at the mean relative deviation for J_U in Figure 3.30 and the corresponding mean standard deviation in Figure 3.32. As can be seen, the dashed lines and the straight lines are more close to each other with an absolute smaller error for the noisy test data and the unlabelled approach.

Therefore, we conclude that the second objective is more robust for parameter estimation using noisy test data in a convolutional neural network.

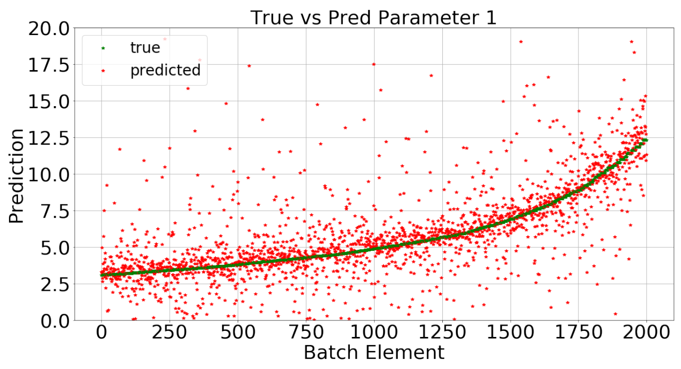

FIG. 3.21. *Index of the noisy test samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_1 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 500\,000$ w.r.t. the objective function J_L.*

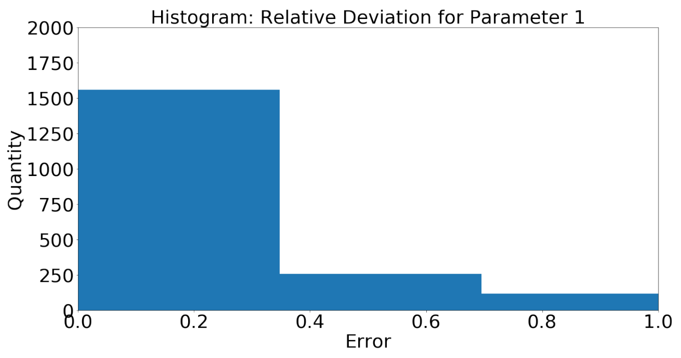

FIG. 3.22. *Relative error (abscissa) between neural network's prediction and true label for p_1 and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all noisy test samples at optimization step $i = 500\,000$ for the objective function J_L.*

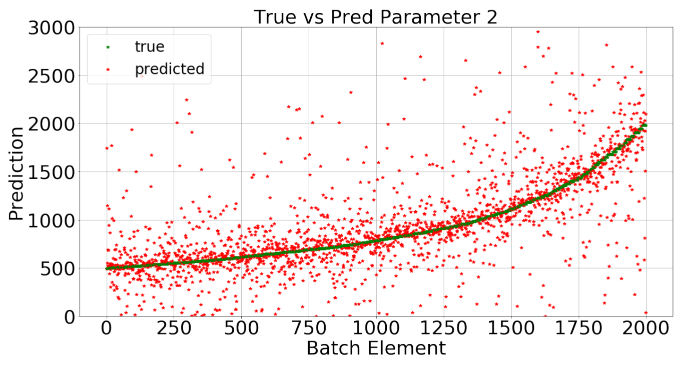

FIG. 3.23. *Index of the noisy test samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_2 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 500\,000$ w.r.t. the objective function J_L.*

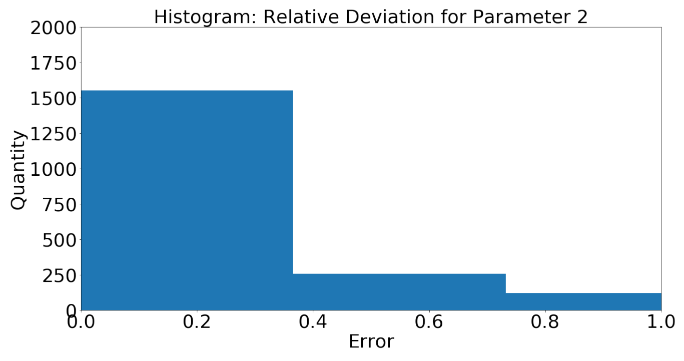

FIG. 3.24. *Relative error (abscissa) between neural network's prediction and true label for p_2 and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all noisy test samples at optimization step $i = 500\,000$ for the objective function J_L.*

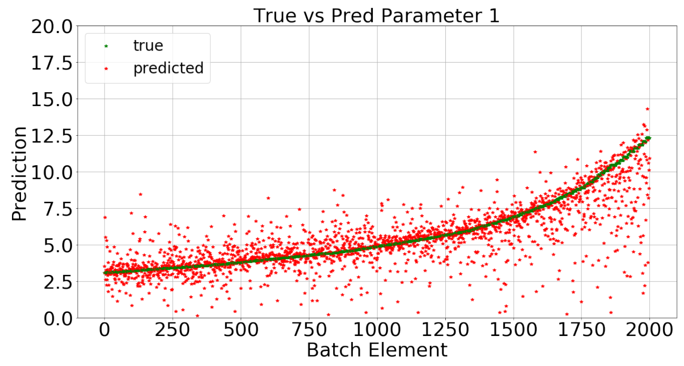

FIG. 3.25. *Index of the noisy test samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_1 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 500\,000$ w.r.t. the objective function J_U.*

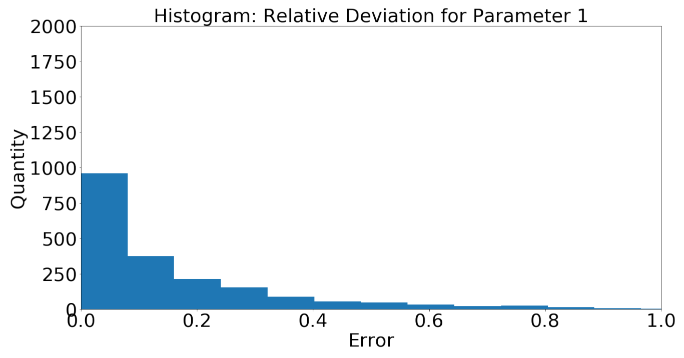

FIG. 3.26. *Relative error (abscissa) between neural network's prediction and true label for p_1 and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all noisy test samples at optimization step $i = 500\,000$ for the objective function J_U.*

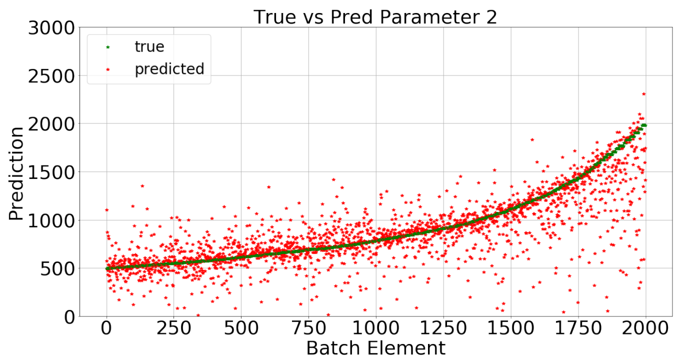

FIG. 3.27. *Index of the noisy test samples (abscissa) sorted according to the value of the label (ordinate). The prediction of the neural network (red) for parameter p_2 compared to the underlying label value (green). The figure shows the prediction for all training samples at optimization step $i = 500\,000$ w.r.t. the objective function J_U.*

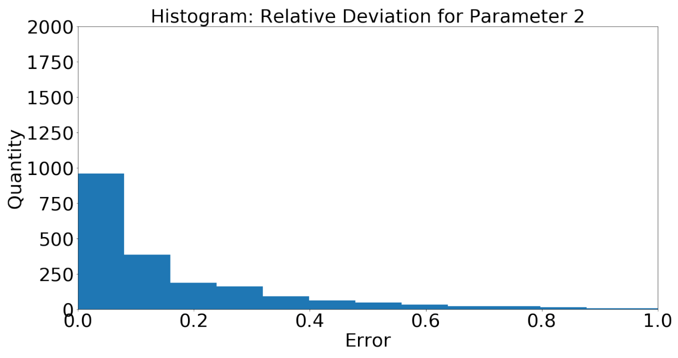

FIG. 3.28. *Relative error (abscissa) between neural network's prediction and true label for p_2 and quantity of samples with same deviation (ordinate) shown as histogram. The plot shows the relative deviation for all noisy test samples at optimization step $i = 500\,000$ for the objective function J_U.*

4. Conclusion.

We have shown that a hybrid objective function, which uses knowledge about the underlying data's dynamics, can be used in a meaningful way to develop a more robust tool for parameter estimation with respect to noisy test data, compared to standard neural network optimization using the data samples' labels.

Therefore, we have discussed the problem of system identification for approximative vehicle models given by a non-homogeneous system of second order ordinary differential equations. Data has been generated in terms of discrete representations of the system components' acceleration, using a symplectic Euler integration scheme to numerically solve the differential equations. Variation of the samples has been guaranteed by randomly generating the road profile and the mass for the uppermost component of the dynamical system. An architecture for a convolutional neural network has been developed as a data driven model to predict the coefficients of the relevant differential equation, based on the generated acceleration data.

The training of the neural network's parameters has been carried out with respect to two different objective functions, one of which uses the true values of the equation's parameters for labelled optimization and the second of which uses reproduction of the input data, comparable to the principle of auto-encoders, for unlabelled optimization.

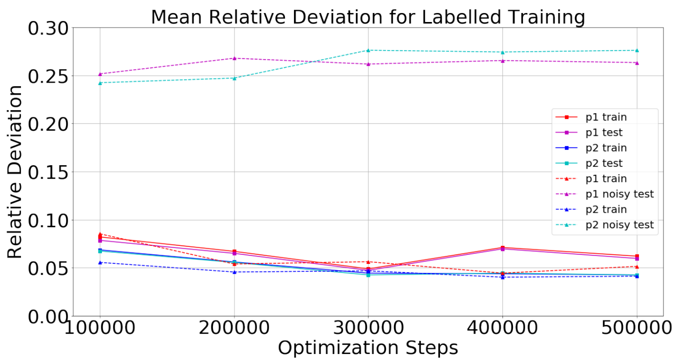

FIG. 3.29. *Mean relative deviation of* p_1 *and* p_2 *for each* 100 000 *steps for two optimization processes using* J_L*: (1) Optimization process using de-noised training and de-noised test data. Mean deviation is shown for* p_1 *training (red line),* p_1 *test (violet line),* p_2 *training (blue line) and* p_2 *test (cyan line). (2) Optimization process using de-noised training and noisy test data. Mean deviation is shown for* p_1 *training (red dashed line),* p_1 *test (violet dashed line),* p_2 *training (blue dashed lined) and* p_2 *(cyan dashed line).*

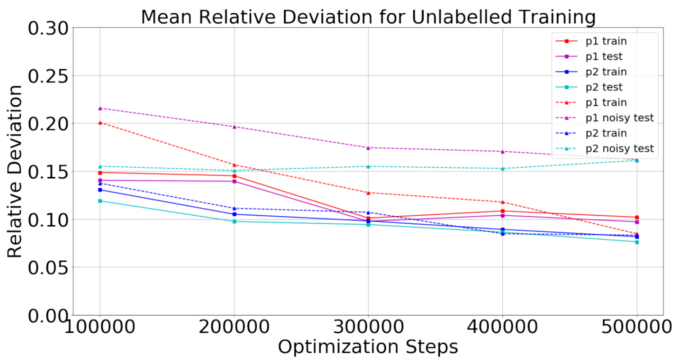

FIG. 3.30. *Mean relative deviation of* p_1 *and* p_2 *for each* 100 000 *steps for two optimization processes using* J_U*: (1) Optimization process using de-noised training and de-noised test data. Mean deviation is shown for* p_1 *training (red line),* p_1 *test (violet line),* p_2 *training (blue line) and* p_2 *test (cyan line). (2) Optimization process using de-noised training and noisy test data. Mean deviation is shown for* p_1 *training (red dashed line),* p_1 *test (violet dashed line),* p_2 *training (blue dashed lined) and* p_2 *(cyan dashed line).*

It has been shown that both objectives can be used to train the neural network's parameters, such that the prediction performance is acceptable for both clean training and clean test data. Here, the labelled approach slightly outperforms the unlabelled approach. Nevertheless, none of the objectives leads to overfitting, when comparing the training and test performance.

In contrast, if test samples are used that have been modified by adding Gaussian noise to the clean data, we can recognize that the unlabelled approach is significantly more robust against noise compared to the labelled one. It is therefore preferable to use prior knowledge about data, when the inner structure is known. As a consequence, it is worth doing further investigations in this field to find the root cause of the results. Simple mathematical models that show similar results following the approaches in this work, could be taken into consideration to get a deeper understanding of this robustness effect.

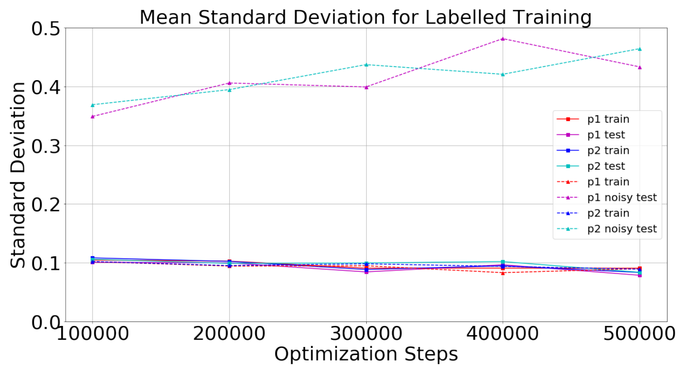

FIG. 3.31. *Mean standard deviation of p_1 and p_2 for each* 100 000 *steps for two optimization processes using J_L: (1) Optimization process using de-noised training and de-noised test data. Mean deviation is shown for p_1 training (red line), p_1 test (violet line), p_2 training (blue line) and p_2 test (cyan line). (2) Optimization process using de-noised training and noisy test data. Mean deviation is shown for p_1 training (red dashed line), p_1 test (violet dashed line), p_2 training (blue dashed lined) and p_2 (cyan dashed line).*

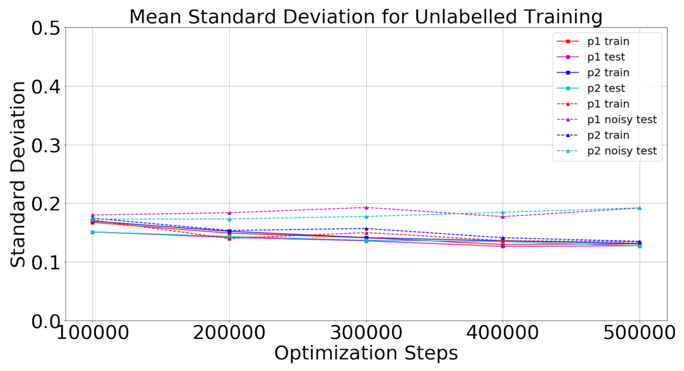

FIG. 3.32. *Mean standard deviation of p_1 and p_2 for each* 100 000 *steps for two optimization processes using J_U: (1) Optimization process using de-noised training and de-noised test data. Mean deviation is shown for p_1 training (red line), p_1 test (violet line), p_2 training (blue line) and p_2 test (cyan line). (2) Optimization process using de-noised training and noisy test data. Mean deviation is shown for p_1 training (red dashed line), p_1 test (violet dashed line), p_2 training (blue dashed lined) and p_2 (cyan dashed line).*

REFERENCES

[1] W. ABBAS, O. B. ABOUELATTA, M. EL-AZAB, M. ELSAIDY, AND A. A. MEGAHED, *Optimization of biodynamic seated human models using genetic algorithms*, Engineering, 2 (2010), pp. 710–719.

[2] W. ABBAS, A. EMAM, S. BADRAN, M. SHEBL, AND O. ABOUELATTA, *Optimal seat and suspension design for a half-car with driver model using genetic algorithm*, Intell. Control Autom., 4 (2013), Art. 31745, 7 pages.

[3] I. AYED, E. DE BÉZENAC, A. PAJOT, J. BRAJARD, AND P. GALLINARI, *Learning dynamical systems from partial observations*, e-print arXiv:1902.11136, February 2019.
https://arxiv.org/abs/1902.11136

[4] S. BAI, J. Z. KOLTER, AND V. KOLTUN, *An empirical evaluation of generic convolutional and recurrent networks for sequence modeling*, e-print arXiv:1803.01271, March 2018.
https://arxiv.org/abs/1803.01271

[5] S. L. BRUNTON, J. L. PROCTOR, AND J. N. KUTZ, *Sparse identification of nonlinear dynamics with control (SINDYc)*, IFAC-PapersOnLine, 49 (2016), pp. 710–715.

[6] F. Chen, N. Chen, H. Mao, and H. Hu, *Assessing four neural networks on handwritten digit recognition dataset (MNIST)*, e-print arXiv:1811.08278, November 2018. https://arxiv.org/abs/1811.08278
[7] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, *Neural ordinary differential equations*, in Advances in Neural Information Processing Systems 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds., Curran Associates, Red Hook, 2018, pp. 6571–6583.
[8] A. Daw, A. Karpatne, W. Watkins, J. Read, and V. Kumar, *Physics-guided neural networks (PGNN): an application in lake temperature modeling*, e-print arXiv:1710.11431, October 2017. https://arxiv.org/abs/1710.11431
[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of deep bidirectional transformers for language understanding*, e-print arXiv:1810.04805v2, May 2019. https://arxiv.org/abs/1810.04805v2
[10] G. Dorffner, *Neural networks for time series processing*, Neural Netw. World, 6 (1996), pp. 447-468.
[11] A. N. Emmanuel, A. David, Y. K. Benjamin, and E. T. Yannick, *A hybrid neural network approach for batch fermentation simulation*, Aust. J. Basic Appl. Sci., 3 (2009), pp. 3930–3936.
[12] A. Faheem, F. Alam, and V. Thomas, *The suspension dynamic analysis for a quarter car model and half car model*, in 3rd BSME-ASME International Conference on Thermal Engineering, 2006.
[13] E. Hairer, C. Lubich, and G. Wanner, *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, Springer, Berlin, 2006.
[14] F. Hamilton, *Parameter estimation in differential equations: A numerical study of shooting methods*, SIAM Undergrad. Res. Online, 4 (2011), Art. 01073, 16 pages.
[15] F. Hamilton, A. L. Lloyd, and K. B. Flores, *Hybrid modeling and prediction of dynamical systems*, PLoS Comput. Biol., 13 (2017), Art. e1005655, 20 pages.
[16] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, IEEE Conference Proceedings, Los Alamitos, 2016, pp. 770–778.
[17] K. J. Keesman, *System Identification: An Introduction*, Springer, London, 2011.
[18] D. P. Kingma and M. Welling, *An introduction to variational autoencoders*, e-print arXiv:1906.02691, June 2019. https://arxiv.org/abs/1906.02691
[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., Curran Associates, Red Hook, 2012, pp. 1097–1105.
[20] A. Kulkarni, S. A. Ranjha, and A. Kapoor, *A quarter-car suspension model for dynamic evaluations of an in-wheel electric vehicle*, Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering, 232 (2018), pp. 1139–1148.
[21] V. Kumar, *Modelling and simulation of a passenger car for comfort evaluation*, Int. J. Res. Appl. Sci. Eng. Technol., 6 (2018), pp. 4013–4018.
[22] C. Li, Z. Ding, D. Zhao, J. Yi, and G. Zhang, *Building energy consumption prediction: An extreme deep learning approach*, Energies, 10 (2017), Art. 1525, 20 pages.
[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, e-print arXiv:1301.3781, January. 2013 https://arxiv.org/abs/1301.3781
[24] A. Mitra, N. Benerjee, H. Khalane, M. Sonawane, D. Joshi, and G. Bagul, *Simulation and analysis of full car model for various road profile on a analytically validated matlab/simulink model*, IOSR J. Mech. Civ. Eng., (2013), pp. 22–33.
[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing atari with deep reinforcement learning*, e-print arXiv:1312.5602, December 2013. https://arxiv.org/abs/1312.5601
[26] M. Peifer and J. Timmer, *Parameter estimation in ordinary differential equations for biochemical processes using the method of multiple shooting*, IET Syst. Biol., 1 (2007), pp. 78–88.
[27] D. C. Psichogios and L. H. Ungar, *A hybrid neural network-first principles approach to process modeling*, AIChE Journal, 38 (1992), pp. 1499–1511.
[28] T. Qin, K. Wu, and D. Xiu, *Data driven governing equations approximation using deep neural networks*, J. Comput. Phys., 395 (2019), pp. 620–635.
[29] L. Rahim, S. Blume, S. Reicherts, P. Sieberg, and D. Schramm, *Zustandsschätzung des Wankverhaltens von Personenkraftwagen mittels künstlicher neuronaler Netze*, in Mobilität in Zeiten der Veränderung, H. Proff, ed., Springer, Wiesbaden, 2019, pp. 229–239.
[30] 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*, J. Comput. Phy., 378 (2019), pp. 686–707.
[31] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations*, e-print arXiv:1711.10566, November 2017. https://arxiv.org/abs/1711.10566

[32] J. R. RAOL AND H. MADHURANATH, *Neural network architectures for parameter estimation of dynamical systems*, IEE Proc.-Control Theory Appl., 143 (1996), pp. 387–394.
[33] D. RAVÌ, C. WONG, F. DELIGIANNI, M. BERTHELOT, J. ANDREU-PEREZ, B. LO, AND G.-Z. YANG, *Deep learning for health informatics*, IEEE J. Biomed. Health Inform., 21 (2016), pp. 4–21.
[34] S. H. RUDY, A. ALLA, S. L. BRUNTON, AND J. N. KUTZ, *Data-driven identification of parametric partial differential equations*, SIAM J. Appl. Dyn. Syst., 18 (2019), pp. 643–660.
[35] S. H. RUDY, J. N. KUTZ, AND S. L. BRUNTON, *Deep learning of dynamics and signal-noise decomposition with time-stepping constraints*, J. Comput. Phys., 396 (2019), pp. 483–506.
[36] C. SZEGEDY, W. LIU, Y. JIA, P. SERMANET, S. REED, D. ANGUELOV, D. ERHAN, V. VANHOUCKE, AND A. RABINOVICH, *Going deeper with convolutions*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015, IEEE Conference Proceedings, Los Alamitos, 2015, pp. 1–9.
[37] F. TYAN, Y.-F. HONG, S.-H. TU, AND W. S. JENG, *Generation of random road profiles*, J. Adv. Eng., 4 (2009), pp. 1373–1378.
[38] J. YANG, J. LI, AND G. LIN, *A simple approach to integration of acceleration data for dynamic soil–structure interaction analysis*, Soil Dyn. Earthq. Eng., 26 (2006), pp. 725–734.
[39] J. YANG, M. N. NGUYEN, P. P. SAN, X. L. LI, AND S. KRISHNASWAMY, *Deep convolutional neural networks on multichannel time series for human activity recognition*, in Proceedings of the 24th International Conference on Artificial Intelligence, 2015, Q. Yang and M. Wooldridge, eds., AAAI Press, Palo Alto, 2015, pp. 3995–4001.
[40] B. YUE, J. FU, AND J. LIANG, *Residual recurrent neural networks for learning sequential representations*, Information, 9 (2018), Art. 56, 14 pages.
[41] M. ZENG, L. T. NGUYEN, B. YU, O. J. MENGSHOEL, J. ZHU, P. WU, AND J. ZHANG, *Convolutional neural networks for human activity recognition using mobile sensors*, in 6th International Conference on Mobile Computing, Applications and Services, IEEE Conference Proceedings, Los Alamitos, 2014, pp. 197–205.