

Neural Networks for Symbolic Regression

Jiří Kubalík¹, Erik Derner², and Robert Babuška³

Abstract—Many real-world systems can be described by mathematical formulas that are human-comprehensible, easy to analyze and can be helpful in explaining the system’s behaviour. Symbolic regression is a method that generates nonlinear models from data in the form of analytic expressions. Historically, symbolic regression has been predominantly realized using genetic programming, a method that iteratively evolves a population of candidate solutions that are sampled by genetic operators crossover and mutation. This gradient-free evolutionary approach suffers from several deficiencies: it does not scale well with the number of variables and samples in the training data, models tend to grow in size and complexity without an adequate accuracy gain, and it is hard to fine-tune the inner model coefficients using just genetic operators. Recently, neural networks have been applied to learn the whole analytic formula, i.e., its structure as well as the coefficients, by means of gradient-based optimization algorithms. In this paper, we propose a novel neural network-based symbolic regression method that constructs physically plausible models based on limited training data and prior knowledge about the system. The method employs an adaptive weighting scheme to effectively deal with multiple loss function terms and an epoch-wise learning process to reduce the chance of getting stuck in poor local optima. Furthermore, we propose a parameter-free method for choosing the model with the best interpolation and extrapolation performance out of all models generated through the whole learning process. We experimentally evaluate the approach on four test systems: the TurtleBot 2 mobile robot, the magnetic manipulation system, the equivalent resistance of two resistors in parallel, and the longitudinal force of the anti-lock braking system. The results clearly show the potential of the method to find sparse and accurate models that comply with the prior knowledge provided.

Index Terms—Symbolic regression, neural networks, physics-aware modeling.

I. INTRODUCTION

Symbolic regression (SR) is a data-driven method that generates models in the form of analytic formulas. It has been successfully used in many nonlinear data-driven modeling tasks with quite impressive results [1], [2], [3], [4]. Historically, SR has been predominantly realized using genetic programming (GP) [1], [5], [6], [7], [8], a method that evolves a population of candidate solutions through a number of generations. This

gradient-free learning process is driven by a selection strategy that prefers high-quality solutions to poor ones, and new candidate solutions are created by genetic operators of crossover and mutation. Some GP-based approaches make use of the loss function gradient to fine-tune the inner coefficients of the model [9], [10], [11].

SR has several advantages over other data-driven modeling methods. For example, contrary to (deep) neural networks, which belong to data-hungry approaches, SR can construct good models even from very small training data sets [12]. SR is also suitable for dealing with prior knowledge [8], [13], [14], [15]. This is very important especially when the data set does not sufficiently cover the input space or when some parts of the input space are completely omitted from the data set. Even when a sufficiently large and informative set of training data is available, methods that minimize only the training error tend to yield partially incorrect models, for instance, in terms of their steady-state characteristics, local, and even global behavior. Using prior knowledge about the desired properties of the modeled system within the learning process allows for learning models that, in addition to a small training error, also exhibit high compliance with the physical properties of the given system.

Despite their high popularity, GP-based SR methods suffer from several deficiencies. The models tend to increase in size and complexity without an adequate increase in the model’s performance. The phenomenon is known as code bloat [16]. Furthermore, GP-based approaches do not scale well with the number of variables and samples in the training data set, since an entire population of formulas has to be evolved and evaluated repeatedly through many generations. Last but not least, it is hard to tune the coefficients of the models using just genetic operators.

Recently, several approaches using neural networks (NNs) to learn analytic formulas by means of gradient-based optimization algorithms have been proposed [17], [18], [19], [20], [21], [22]. They all share the idea that analytic models are represented by a heterogeneous NN with units implementing mathematical operators and functions, such as $\{+, -, *, /, \sin, \exp, \text{etc.}\}$. The weights of the network are adjusted using standard gradient-based methods with the ultimate goal of minimizing the training error while maximally reducing the number of active units, i.e., only those units that have above-threshold input/output weights and are involved in the computation of the final NN output. Thus the final NN represents a simple analytic formula. The individual approaches differ in how the learning process is driven towards these sparse analytic models.

In this paper, we propose a novel NN-based SR approach, N4SR (pronounced as “enfɔ:sə”), that allows for using prior

¹Jiří Kubalík is with the Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, 16636 Prague, Czech Republic, jiri.kubalik@cvut.cz.

²Erik Derner is with the Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, 16636 Prague, Czech Republic, erik.derner@cvut.cz.

³Robert Babuška is with Cognitive Robotics, Delft University of Technology, 2628 CD Delft, The Netherlands and with the Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, 16636 Prague, Czech Republic, r.babuska@tudelft.nl.

This work was supported by the European Regional Development Fund under the project Robotics for Industry 4.0 (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000470).

knowledge represented by and evaluated on constraint samples as introduced in [14], [15]. The NN uses an EQL-like architecture [17] with skip connections [20]. The learning process is divided into epochs. During the learning process, models of varying sizes, measured by the number of active weights, are generated. Then, the final model is chosen as the best-performing model among the least complex ones. The model’s performance is judged based on its validation root-mean-square error and compliance with the validity constraints and prior knowledge.

Note that the problem of seeking a sparse model that has a low training error and exhibits desired characteristics is a multi-objective optimization problem. This implies that there is a strong interplay among the respective terms of the loss function. It may happen that some terms become dominant in the loss function while suppressing the effect of some other terms. To remedy this problem, we propose a self-adaptive strategy to keep the pair-wise ratios between the terms around the required values during the whole optimization process.

The main contributions of this paper are:

- We introduce an NN-based SR approach that uses training data and prior knowledge to generate precise and physically plausible models.
- We introduce a self-adaptive strategy to control the contributions of the training error term, regularization term, and constraint error term during the whole optimization process. We show that this method is effective compared to the static one. Moreover, it reduces the number of parameters to be tuned for each single SR instance.
- We propose the final model selection method based on the model’s complexity and the constraint violation error. The method does not require any extrapolation test set. We show that the proposed method is competitive with the one based on the extrapolation error where data sampled from the extrapolation domain are needed.
- The proposed N4SR is thoroughly evaluated on four test problems, including the validation of our design choices, and compared to relevant methods.

The paper is organized as follows. The related work is surveyed in Section II. Then, the particular SR problem considered in this work is defined in Section III. The proposed method is described in Section IV. Section V describes the experiments set up and presents and discusses the results obtained on the four test problems. Finally, Section VI concludes the paper.

II. RELATED WORK

One of the first works on using NN for symbolic regression is [17], where the original Equation Learner (EQL) was introduced. It works with a simple feed-forward multi-layer architecture with several unit types – sigmoid, sine, cosine, identity, and multiplication. The network is trained using a stochastic gradient descent algorithm with mini-batches, Adam [23], and a Lasso-like objective combining the L_2 training loss and L_1 regularization. Moreover, it uses a hybrid regularization strategy that starts with a certain number of update steps without regularization, followed by a regularization phase to

enforce a sparse network structure to emerge, and ends with the final phase with disabled regularization but enforced the L_0 norm of the weights (i.e., keeping all weights that are close to 0 at 0). An important question is how to choose the right network instance (i.e., the final model) among all the network instances generated during the learning process. In [17], they solve it by ranking the network instances w.r.t. validation error and sparsity and selecting the one with the smallest L_2 norm (in rank-space). However, it was shown that in some cases, this does not select a network instance with the best performance metrics.

In [18], an extended version of EQL, denoted as EQL^\dagger , was proposed. In addition to the original EQL, it allows for modelling divisions using a modified architecture that places the division units in the output layer. The objective function is a linear combination of L_2 training loss and L_1 regularization extended by a penalty term for invalid denominator values P^θ . Furthermore, special *penalty epochs* are injected at regular intervals into the training process to prevent output values on data from extrapolation regions having a very different magnitude than the outputs observed on training data. During the penalty epochs, only the penalty function $P^\theta + P^{\text{bound}}$ is minimized, where P^{bound} penalizes for the outputs larger than the maximal desired value observed on all data points (including the extrapolation ones). This way, a reasonable but not necessarily correct behaviour of the model on the extrapolation region is enforced. Moreover, one must estimate the maximal desired output value in advance, which cannot be done reliably in general. Here, the model selection method chooses the network instance that minimizes the sum of normalized interpolation and extrapolation validation errors, where the extrapolation error is calculated on few measured extrapolation points. On the one hand, this method was shown to work better than the one used in EQL. On the other hand, it still relies on known extrapolation points, though just few of them. Another extension of EQL is the informed EQL (iEQL) proposed in [19]. It uses expert knowledge about what are permitted or prohibited equation components and a domain-dependent structured sparsity prior. The authors demonstrated in artificial as well as real-world experiments that iEQL can learn interpretable models of high predictive power. However, the expert knowledge might be hard to define reliably for some problems as even nontrivial nested structures may be beneficial in some cases.

In [21], the EQL architecture with other deep learning architectures and $L_{0.5}$ regularization was proposed. Its power was demonstrated on a simple arithmetic task where a convolutional network is used to extract handwritten MNIST¹ digits and a set of experiments, where the EQL network was applied to analyze physical time-varying systems. Partially inspired by the EQL network, a new multi-layer NN architecture, OccamNet, that represents a probability distribution over functions was proposed in [20]. It uses skip-connections similar to those in DenseNet [24] and a temperature-controlled connectivity scheme, which uses the probabilistic interpretation of the softmax function by sampling sparse paths through

¹<http://yann.lecun.com/exdb/mnist/>

a network, to maximize sparsity. The Mathematical Operation Network (MathONet) proposed in [22] also uses EQL-like NN architecture. The sparse sub-graph of the NN is sought using the proposed Bayesian learning approach that incorporates structural and non-structural sparsity priors. The system was shown to be able to discover ordinary and partial differential equations from the observations.

A different NN-based SR approach class uses deep neural network transformers such as the GPT-2 [25]. They learn the transformer model using a large amount of training data where each sample is typically a tuple of the form (formula, data sampled from the formula). During inference, the transformer model constructs the formula based on a particular data set query. This approach has its own advantages and disadvantages, but this is out of the scope of this work. We refer interested readers to [26], [27], [28], [29].

III. PROBLEM DEFINITION

In this work, we consider a regression problem where a neural network model representing the function $\phi: \mathbb{R}^n \rightarrow \mathbb{R}$ is sought, with n the number of input variables. In particular, a maximally sparse neural network model representing a desirably concise analytic expression is trained so that it simultaneously minimizes the error observed on the training data set, maximizes its validity, and maximizes its compliance with the constraints imposed on the model (defining the model’s desired physical properties). The model’s validity reflects the fact that the neural network may contain units with singularities, such as the division one, which calculates a/b and exhibits a singularity at $b = 0$. Multiple types of singularity units can be considered here. We denote the set of singularity types used in the network as T^s . The constraint satisfaction measure is calculated on a set of constraint samples generated specifically for each constraint type as proposed in [14], [15]. The set of all constraints defined for the particular SR problem is denoted as T^c . The sparsity of the model is measured by the number of active weights and units in the neural network. Note that the ultimate goal is to generate models that not only work well on the training data, i.e., exhibit good interpolation performance, but also work well on the data sampled outside the training data domain, i.e., exhibit good extrapolation performance as well. Thus, while the model is trained on the training data plus the constraint samples representing general model’s properties, the final model’s performance is evaluated on the test data set that comes from different regions of the input space than the training data set.

IV. METHOD

This section describes the main components of N4SR, namely the architecture of the neural network, the forms of the loss function used in different stages of the run, the self-adaptive loss terms weighting scheme, the learning process procedure, and the final model selection rule.

A. Data sets used for learning

Before we describe the method itself, we introduce the following data sets used to train the neural network model:

- Training data, D_t – contains data samples of the form $\mathbf{d}_i = (\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \mathbb{R}^n$ is sampled from the training domain \mathbb{D}_t .
- Validation data, D_v – contains data samples of the same form as D_t sampled from \mathbb{D}_t such that $D_v \cap D_t = \emptyset$. This data set is used to choose intermediate models within the learning process, see Section IV-F, and to choose the final output model according to the proposed model selection method, see Section IV-E.
- Constraint data, D_c – we adopt the constraint representation and evaluation scheme as proposed in [14]. It assumes that any type of prior knowledge can be written as nonlinear inequality and equality constraints that the system must obey. Synthetic constraint samples (i.e., samples not measured on the system) are generated specifically for each constraint in T^c and the desired inequality or equality relation is defined on them. Then, the constraint violation error measures how much the model violates the desired inequality or equality relations over the constraint samples.

B. Architecture

The approach we propose in this paper uses an EQL-like architecture similar to the ones introduced in [19] and [20]. It takes advantage of the skip connections so that simple structures present in shallow layers can be efficiently learned due to a direct propagation of the error gradients. Moreover, these shallow structures can be refined and reused in the subsequent layers. We also allow for using units with singularities. Contrary to the EQL^\pm , singularity units can be used at any layer of the network.

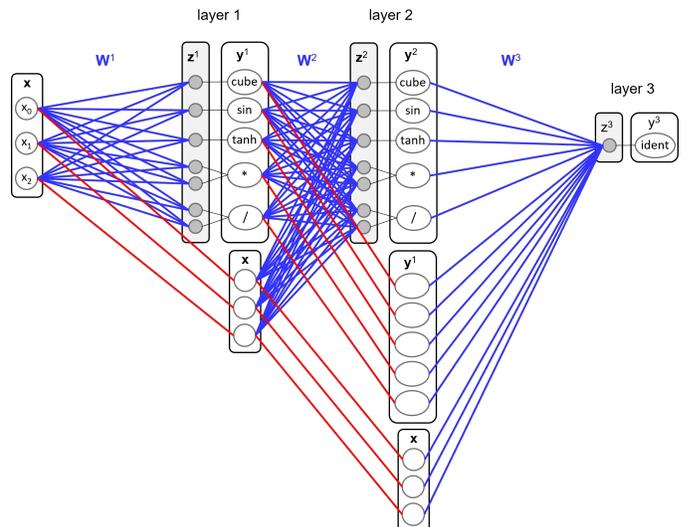


Fig. 1. Network architecture with two hidden layers and one output layer unit. The blue lines mark links with learnable weights. The red lines are the skip connections leading from the source units of layer $k - 1$ to the copy units in layer k . These links are permanently set to 1. For simplicity, this scheme does not show bias links leading to every z node.

Figure 1 shows the core architecture components on an example of the neural network with two hidden layers and one output layer unit. Each hidden layer contains “learnable

units” and the “copy units” (the term introduced in [19]). The learnable units are units whose input weights can be tuned within the learning process (i.e., the links shown in blue). The *learnable weights* of all learnable units are collected in the set W_l . The copy units in layer k are copies of all units from the previous layer $k - 1$. Their weights are permanently set to 1 and are not subject to the learning process (these are the links shown in red). Units represent elementary unary functions, e.g., sin, cube, tanh, and binary functions such as multiplication $*$ and division $/$. Each unit i in layer l calculates its output as

$$y_i^l = g(z_{i,0}^l), \text{ for unary elementary function } g \quad (1)$$

and

$$y_i^l = h(z_{i,0}^l, z_{i,1}^l), \text{ for binary elementary function } h \quad (2)$$

where $z_{i,-}^l$ is an affine transformation of the whole previous layer’s output calculated as

$$z_{i,-}^l = \mathbf{W}_i^l \mathbf{y}^{l-1} + b_i^l. \quad (3)$$

In Figure 1, each hidden layer contains learnable units composed of a single instance of each unary and binary unit type. In general, there may be multiple instances of each unit type. Similarly, the output layer may have multiple output units, not just a single one, and they can be of any type out of the unary and binary unit types, not necessarily the identity one as used here. This depends on the SR problem solved and on the expert knowledge about the formula form sought.

C. Loss functions

Throughout the learning process, the following three loss functions are used in the different stages of the learning process, described in Section IV-F.

$$\begin{aligned} \mathcal{L}_1 &= \mathcal{L}^t + \mathcal{L}^s \\ \mathcal{L}_2 &= \mathcal{L}^t + \mathcal{L}^s + \mathcal{L}^c \\ \mathcal{L}_3 &= \mathcal{L}^t + \mathcal{L}^s + \mathcal{L}^c + \mathcal{L}^r \end{aligned} \quad (4)$$

The loss functions are composed of the following terms:

- Training RMSE, \mathcal{L}^t – this is the root-mean-square error (RMSE) observed on the training data set D_t .
- Singularity units loss, \mathcal{L}^s – this term is defined as the weighted sum of scaled RMSE values calculated for each singularity unit type $sterm_j$ over the aggregated data set $D = D_t \cup D_v \cup D_c$. Note that all these data samples can be used to check the validity of the model as we do not need to know the required output value y . Instead, just the values of the respective z node of each singularity unit are checked whether they take on values greater than or equal to a user-defined threshold θ_j^s for the given singularity type j . Each $sterm_j$ value is divided by its $shist_j$, which is a scaling coefficient reflecting the current trend over the

last $sterm_j$ values as described in Section IV-D. The \mathcal{L}^s is formally defined as

$$\begin{aligned} \mathcal{L}^s &= \alpha \sum_{j \in T^s} \frac{sterm_j}{shist_j}, \\ sterm_j &= \sqrt{\frac{1}{|S_j||D|} \sum_{u \in S_j} \sum_{d \in D} (m_j(\theta_j^s, z_u))^2} \end{aligned} \quad (5)$$

where S_j is the set of all singularity units of the given type j in the model, $m_j(\theta_j^s, z_u)$ is a function defining a suitable error metric for the given singularity type, and z_u is the critical z node of the respective singularity unit u (e.g., the denominator in case of the division unit). The coefficient α is determined using a self-adaptive scheme described below.

We adopt the implementation of the division operation originally proposed in [18] and further extended in [19] that uses the following error metric

$$m_j(\theta_j^s, z_u) = \max(\theta_j^s - z_u, 0).$$

It makes use of the fact that real systems do not operate at the pole $b \rightarrow 0$; thus, only the positive branch of the hyperbola, $1/b$ where $b > 0$, is sufficient to represent the division while the numerator $a \in \mathbb{R}$ determines the sign of the division output value. Here, $b = z_u$ and the threshold $\theta_j^s = 0.0001$ is used to prevent the z_u values from converging very close to the pole value. This representation can also be used for other units exhibiting a singularity such as \log that is defined on the interval $(0, \infty)$ with the singularity at 0.

- Constraint loss, \mathcal{L}^c – this term accumulates the error of the model in terms of the prior knowledge violation. Like the \mathcal{L}^s , this term is calculated as the weighted sum of scaled RMSE values calculated for each constraint on its own specific constraint data set. Formally, the \mathcal{L}^c is defined as

$$\mathcal{L}^c = \beta \sum_{j \in T^c} \frac{cterm_j}{chist_j} \quad (6)$$

where $cterm_j$ is the root-mean-square error calculated for the constraint j on its constraint data set and $chist_j$ is the current trend of the $cterm_j$ values, see below. Again, the coefficient β is determined using the self-adaptive scheme.

- Regularization, \mathcal{L}^r – this term drives the learning process towards a sparse neural network representing a concise analytic expression. Here, we adopt a smoothed $L_{0.5}$ regularization, $L_{0.5}^*$, as proposed in [21]. It exhibits several good properties. It is a trade-off between L_0 and L_1 regularization as it represents an optimization problem that can be solved using gradient descent, contrary to L_0 , and at the same time, it enforces sparsity more strongly than L_1 while penalizing less the magnitude of the weights. Contrary to the original $L_{0.5}$ regularization, $L_{0.5}^*$ does not suffer from the singularity in the gradient as the weights converge to 0 and uses a piece-wise function to smooth out the function at small magnitudes. For a detailed setup of $L_{0.5}^*$, see [21]. The \mathcal{L}^r term is calculated

as the sum of $L_{0.5}^*$ contributions of all *active weights*, i.e., the weights that have the absolute value no less than a user-defined threshold θ^a

$$\begin{aligned} \mathcal{L}^r &= \gamma \frac{rterm}{rhist}, \\ rterm &= \sum_{w \in W_a} L_{0.5}^*(w) \end{aligned} \quad (7)$$

where $W_a \subseteq W_l$ such that $W_a = \{w : w \in W_l \wedge \text{abs}(w) \geq \theta^a\}$, $rhist$ is the current trend of the $rterm$ values, and the coefficient γ is determined using the self-adaptive scheme. The number of active weights is used as the *complexity measure* of the NN model during the learning process.

D. Self-adaptive loss terms weighting scheme

All three forms of the loss function involve multiple terms, which leads to a multi-objective optimization problem. Besides the fact the terms may be competing with each other (e.g., \mathcal{L}^t vs. \mathcal{L}^r , the model's precision and complexity), they may also differ substantially in the scale of the values they attain. Typically, a weighted sum of the individual terms is used as the final loss to be minimized, which may lead to an unwanted dominance of certain terms.

To remedy this situation, we propose a self-adaptive method that adapts the coefficients α , β , and γ involved in the loss terms \mathcal{L}^s , \mathcal{L}^c , and \mathcal{L}^r throughout the whole learning process in order to keep the desired ratios $r_{s/t} = \mathcal{L}^s : \mathcal{L}^t$, $r_{c/t} = \mathcal{L}^c : \mathcal{L}^t$, and $r_{r/t} = \mathcal{L}^r : \mathcal{L}^t$. It uses a sliding window strategy that works with lists of values of \mathcal{L}^t , $sterm_j$, $cterm_j$, and $rterm$, respectively, observed in the last N_w iterations of the learning process. The weights α , β , and γ are updated in each iteration according to Algorithm 1.

Algorithm 1: Function `getTermWeight(Bt, B, type, r)`.

Input: B^t ... set of N_w last \mathcal{L}^t values
 $type$... type of the loss term to be processed;
 $type \in \{S, C, R\}$
 Depending on the $type$, B is a set of N_w last
 $\frac{sterm_j}{shist_j}$ values for each singularity unit type $j \in T^s$
 or
 $\frac{cterm_j}{chist_j}$ values for each constraint type $j \in T^c$
 or
 $\frac{rterm}{rhist}$ values
 r ... desired ratio of the given loss term to \mathcal{L}^t

Output: cf ... given loss term's weighting coefficient

```

if  $type == S$  then
   $acc \leftarrow \sum_{j \in T^s} \text{mean}(\frac{sterm_{j,1}}{shist_j}, \dots, \frac{sterm_{j,N_w}}{shist_j})$ 
if  $type == C$  then
   $acc \leftarrow \sum_{j \in T^c} \text{mean}(\frac{cterm_{j,1}}{chist_j}, \dots, \frac{cterm_{j,N_w}}{chist_j})$ 
if  $type == R$  then  $acc \leftarrow \text{mean}(\frac{rterm_1}{rhist}, \dots, \frac{rterm_{N_w}}{rhist})$ 
if  $acc == 0$  then
  |  $cf \leftarrow 1$ 
else
  |  $cf \leftarrow r \frac{\text{mean}(B^t)}{acc}$ 
return  $cf$ 

```

First, the mean value of the respective N_w terms (i.e., $\frac{sterm_j}{shist_j}$, $\frac{cterm_j}{chist_j}$, or $\frac{rterm}{rhist}$) is calculated, denoted as acc . If the acc is zero, then the respective loss term weight is set to 1. Otherwise, the weight is set to reflect the corresponding desired ratio. The $shist_j$, $chist_j$, and $rhist$ values are calculated as the mean of the last $sterm_j$, $cterm_j$, and $rterm$ values observed during the last N_w iterations. They are used to normalize the raw constraint, singularity, and regularization terms so that they all contribute to acc with values of the same magnitude. Moreover, each singularity and constraint type contributes to acc relatively to its current trend and independently to the other types. Thus, if the $cterm_j$ equals the $chist_j$, then the constraint j contributes to acc with the value of 1. If $cterm_j$ is less than $chist_j$, then the constraint j contributes to acc with a value less than one and vice versa. It works the same for the singularity terms. Without this normalization, some singularity or constraint type may dominate within the respective loss term if its values are by orders of magnitude higher than the others.

The weights α , β , and γ are used to scale the raw loss terms values so that the actual ratios $r_{s/t}^*$, $r_{c/t}^*$, and $r_{r/t}^*$ follow the desired ratios. After the scaling is applied, i.e., the current values of \mathcal{L}^s , \mathcal{L}^c , and \mathcal{L}^r have been calculated, it is further checked whether the obtained loss terms do not exceed the maximum value for which the actual ratio is less than or equal to the desired one. If this condition is violated, the respective loss term value is set to the value that implies the actual ratio r_{-}^*/t is equal to the desired one r_{-}/t .

Note that only $sterm_j$, $cterm_j$, and $rterm$ are scaled in each iteration. The \mathcal{L}^t serves as a baseline relative to which the other terms are adjusted. Since the primary goal is to fit well the training data, each of the $r_{s/t}$, $r_{c/t}$, and $r_{r/t}$ should be set to a value less than 1.

E. Final model selection

During the learning process, many NN models are generated. It is important to select the best one in the end, while the pursued criteria are, in general, the model complexity, and its interpolation and extrapolation performance. In [17], [18], and [19] as well as in [20] and [21], the final model selection method always builds on the fact that ‘‘few’’ extrapolation points are known, i.e., both the input variable values as well as the target value of the points are known. Despite the fact that the extrapolation points are not used within the learning process, the assumption that such a concrete piece of information about the desired model's performance outside the domain of the training data set is available for the final model selection makes the whole approach dependent on such type of data. This renders such approaches limited as they can not be used, for example, in the situation when no extrapolation points can be measured on the system by the time the system is to be modeled.

Here, we propose a final model section strategy that does not require any set of extrapolation points. Instead, it uses just the model's complexity, its validation RMSE, and the measures of its compliance with the prior knowledge and the singularity units' constraints. Note that in practice, it is much easier

to define the desired “high-level” model’s properties than to obtain particular measurements on the system. The proposed method uses an acceptance rule where one of the following two conditions must hold in order for the new model, $model$, to be accepted as the best-so-far model, $model^*$:

- 1) The $model$ ’s complexity is lower than the complexity of $model^*$.
- 2) The $model$ ’s complexity is equal to the complexity of $model^*$

and

for all $j \in T^s$, the $model$ ’s $sterm_j$ is not worse than that of $model^*$

and

for all $j \in T^c$, the $model$ ’s $cterm_j$ is not worse than that of $model^*$

and

the $model$ ’s validation RMSE is not worse than that of $model^*$.

Thus, the final $model^*$ is the least complex model found with the best values of all of the $sterm_j$, $cterm_j$, and the validation RMSE objectives among the models of the same minimum complexity.

F. Epoch-wise learning process

The whole learning process, see Algorithm 2, is divided into three stages – initial, exploration-focus, and final stage.

The goal of the initial stage is to evolve the NN such that it exhibits at least partial capabilities (1) to fit the training data, (2) to satisfy the constraints imposed on the singularity units, and (3) to satisfy the constraints imposed on the desired model’s properties. In the first half of this stage, the \mathcal{L}_1 loss function is optimized, while in the rest of this stage the \mathcal{L}_2 is optimized. The complexity of the NN does not matter in this stage. The NN then passes to the exploration-focus stage where it is further trained in an epoch-wise manner. Each epoch starts with the exploration phase, where all learnable weights W_l are considered, followed by the focus phase, where only active weights W_a are further refined. The active weights are collected in the `maskWeights()` function at line 18 as the weights with the absolute value no less than the threshold θ^a . This is the only phase of the learning process where the \mathcal{L}^r term is used to drive the search towards a simpler model. The final stage performs the fine-tuning of the active weights of the NN. All weights that become inactive at any iteration of this stage, line 25, are set to zero for the rest of the run and only active weights are updated in each `learningStep()` using \mathcal{L}_2 . Thus, the complexity of the model can only decrease in this stage. Finally, the $model^*$ in the form of an analytic expression represented by the best-of-run NN model is returned.

As described above, a different loss function is used in different phases of the learning process. Also, either all learnable weights or just the active ones are considered for being adjusted within the gradient descent learning step. These two things are passed as input parameters to the `learningStep()` function in each iteration. Depending on the loss function used, respective weighting coefficients

Algorithm 2: N4SR algorithm

Input: Neural network with the set of learnable weights W_l
 $N_{init}, N_{final} \dots$ number of iterations of the initial and final stage
 $N_w \dots$ size of the loss term weights’ adaptation window
 $N_e, N_f \dots$ number of iterations of the exploration and focus phase
 $E \dots$ number of the exploration-focus stage epochs

Output: Model in the form of an analytic expression represented by the final sparse network

```

1  init ( $W_l$ )
   /* initial stage */
2  for  $0 \leq i < N_{init}/2$  do
3     $B^t, B^s, model^* \leftarrow learningStep(W_l, \mathcal{L}_1)$ 
4     $\alpha \leftarrow getTermWeight(B^t, B^s, S, r_s/t)$ 
5  for  $0 \leq i < N_{init}/2$  do
6     $B^t, B^s, B^c, m^*, model^* \leftarrow learningStep(W_l, \mathcal{L}_2)$ 
7     $\alpha \leftarrow getTermWeight(B^t, B^s, S, r_s/t)$ 
8     $\beta \leftarrow getTermWeight(B^t, B^c, C, r_c/t)$ 
9   $M^* \leftarrow \{m^*\}$ 
   /* exploration-focus stage */
10 for  $0 \leq e < E$  do
11    $W_l \leftarrow getWeights(m^*)$ 
12    $\theta^v \leftarrow (1 + \epsilon) \cdot getMeanValidRMSE(M^*)$ 
13   for  $0 \leq i < N_e$  do
14      $B^t, B^s, B^c, model^* \leftarrow learningStep(W_l, \mathcal{L}_2)$ 
15      $\alpha \leftarrow getTermWeight(B^t, B^s, S, r_s/t)$ 
16      $\beta \leftarrow getTermWeight(B^t, B^c, C, r_c/t)$ 
17   for  $0 \leq i < N_f$  do
18      $W_a \leftarrow maskWeights()$ 
19      $B^t, B^s, B^c, B^r, m^*, model^* \leftarrow learningStep(W_a,$ 
20        $\mathcal{L}_3)$ 
21      $\alpha \leftarrow getTermWeight(B^t, B^s, S, r_s/t)$ 
22      $\beta \leftarrow getTermWeight(B^t, B^c, C, r_c/t)$ 
23      $\gamma \leftarrow getTermWeight(B^t, B^r, R, r_r/t)$ 
24      $M^* \leftarrow update(M^*, m^*)$ 
   /* final stage */
25 for  $0 \leq i < N_{final}$  do
26    $W_a \leftarrow maskWeights()$ 
27    $B^t, B^s, B^c, model^* \leftarrow learningStep(W_a, \mathcal{L}_2)$ 
28    $\alpha \leftarrow getTermWeight(B^t, B^s, S, r_s/t)$ 
29    $\beta \leftarrow getTermWeight(B^t, B^c, C, r_c/t)$ 
29 return  $model^*$ 

```

are updated after each learning step (e.g., lines 7–8 executed after line 6). Besides the weight update itself, the `learningStep()` function updates several other objects. Firstly, it updates structures storing the history of the last N_w values of the relevant loss terms. It also updates, when applicable, the model m^* which serves as the seed for each epoch of the exploration-focus stage, see line 19 (and line 6 where m^* is initialized for the first time). Lastly, the function returns the current version of the $model^*$, which is the best-so-far model with respect to the defined final model selection strategy, see Section IV-E.

The core of the learning process is the exploration-focus stage. This stage implements a restarted optimization strategy to avoid premature convergence to a potentially poor suboptimal model. It runs several epochs, each executes the exploration and focus phase one by one. At the beginning of each epoch, all learnable weights are set to the weights of the

current m^* , line 11. Then, the maximum acceptable RMSE observed on the validation set D_v , θ^v , is calculated, line 12. It is defined as $(1 + \epsilon)$ times the mean validation RMSE over all models in M^* , which is a variable storing the final m^* of the last k epochs, see lines 9 and 23. The threshold θ^v is used in the acceptance rule that determines whether the new NN model updated within `learningStep()` will be accepted as the m^* of the current epoch. It gets accepted iff its complexity is not higher than the current m^* complexity and its validation RMSE is not higher than the threshold θ^v . The parameter ϵ defines a tolerance margin, i.e., how much the current m^* can be worse in terms of the validation RMSE than the mean of k last m^* models. Here we use $\epsilon = 0.5$.

During the whole exploration phase, all weights in W_l are optimized with respect to the \mathcal{L}_2 loss function, line 14. After the exploration phase, the focus phase is carried out. Each iteration of this phase starts with extracting the set of active weights W_a , line 18, which are then optimized using \mathcal{L}_3 . Once the focus phase has been completed, M^* is updated using the current m^* .

V. EXPERIMENTS

A. Algorithms compared

In this study, we experiment with three different methods. We compare multiple variants of N4SR, an alternative neural network-based method from the literature, and a genetic programming-based algorithm.

The N4SR algorithms are divided into variants denoted as N4SR-W_SCL with:

- Weighting $W \in \{A, S\}$ – the adaptive (A) or static (S) loss terms weighting scheme. In the static variant, the weighting coefficients α , β , and γ do not adapt. Instead, they are determined at the moment when they are used for the first time and stay constant for the rest of the run. In particular, coefficient α is calculated at the very first iteration of the run using the current \mathcal{L}^t and $sterm_j$ values as $\alpha = r_{s/t} \cdot \mathcal{L}^t / \sum_{j \in T^s} sterms_j$. Similarly, coefficient β is calculated at the first iteration of the second phase of the initial stage, line 8 of Algorithm 2, using the current \mathcal{L}^t and $cterm_j$ values as $\beta = r_{c/t} \cdot \mathcal{L}^t / \sum_{j \in T^c} cterms_j$. Coefficient γ is calculated at the first iteration of the first pass through the focus phase, line 22 of Algorithm 2, using the current \mathcal{L}^t and $rterm$ values as $\gamma = r_{r/t} \cdot \mathcal{L}^t / rterm$. Then, the loss terms \mathcal{L}^s , \mathcal{L}^c , and \mathcal{L}^r are calculated using modified (5), (6), and (7) without the normalization, so that $shist_j = 1$, $chist_j = 1$, and $rhist = 1$.
- Selection $S \in \{C, E\}$ – the constraint satisfaction-based (C) final model selection rule, defined in Section IV-E, or an extrapolation-based (E) final model selection strategy. The extrapolation-based strategy is a modification of the one defined in Section IV-E such that only the *model*'s complexity and its RMSE observed on a few extrapolation points are considered.
- Constraints $C \in \{Y, N\}$ – denoting whether the constraints are used (Y) or not (N) within the learning process.

The variant without constraints works according to Algorithm 2 with the modification that the loss functions \mathcal{L}_2 and \mathcal{L}_3 do not involve the \mathcal{L}^c term. It also implies that this variant can work only with the extrapolation-based final model selection rule.

- Learning strategy $L \in \{E, S\}$ – the epoch-wise (E) or a single-epoch (S) learning used within the exploration-focus stage. The single-epoch variant goes through a single epoch of the exploration-focus stage with N_f set so that the total number of iterations spent in this stage is the same as for the epoch-wise variant.

EQL^\dagger – the EQL algorithm with division units and the improved model selection method working with the extrapolation-validation dataset, proposed in [18]. We used the publicly available implementation².

mSNGP-LS – the multi-objective SNGP using the local search procedure to estimate coefficients of the evolved linear models, proposed in [14].

B. Evaluation data sets

The following data sets were used to evaluate models obtained with the compared algorithms:

- Extrapolation data set, \bar{D}_e – contains data samples of the same form as D_t sampled from the extrapolation domain. We use the extrapolation domain \mathbb{D}_e to denote the parts of the problem's input space whose samples are either very sparsely present in D_t and D_v or are entirely omitted in these data sets. This data set is not used for learning. It is used just to select the final output model according to the methods that rely on a few known samples measured in the extrapolation domain. This is the case of the EQL^\dagger algorithm and the N4SR variants with $S = E$.
- Interpolation test data set, D_i – contains data samples of the same form as D_t sampled from \mathbb{D}_t such that $D_i \cap (D_t \cup D_v) = \emptyset$. This data set is used to evaluate models' interpolation performance.
- Extrapolation test data set, D_e – contains data samples of the same form as D_t sampled from the extrapolation domain \mathbb{D}_e . This data set is used to evaluate models' extrapolation performance.

C. Test problems

The proposed method was experimentally evaluated on the following four problems. We chose these problems since we possess detailed knowledge of the data and the desired properties of the models sought. Moreover, we can illustrate interesting application scenarios for these problems, such as using very sparse or unevenly distributed training data. Standard data-driven modeling approaches fail to generate physically plausible models when only such insufficient data sets are available.

1) *TurtleBot*: This problem is to find a discrete-time model of a real physical system, the two-wheel TurtleBot 2 mobile robot (Figure 2).

²<https://github.com/martius-lab/EQL>

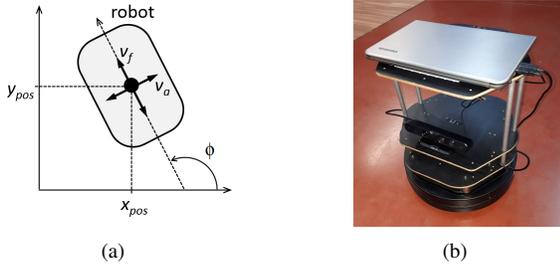


Fig. 2. TurtleBot mobile robot. A schematic (a) and a photo of the system (b).

The robot's state is captured by the state vector $\mathbf{x} = (x_{pos}, y_{pos}, \phi)^\top$, with x_{pos} and y_{pos} the robot's position coordinates and ϕ the robot's heading. The control input is $\mathbf{u} = (v_f, v_a)^\top$, with v_f and v_a the desired forward and angular velocity, respectively. In this work, we model only the x_{pos} component of the robot's motion model since (1) the y_{pos} component is analogous and (2) it is more illustrative than the ϕ component as there are more types of prior knowledge defined for x_{pos} . The model has the form of the following nonlinear difference equation

$$x_{pos,k+1} = f^{x_{pos}}(x_{pos,k}, y_{pos,k}, \phi_k, v_{f,k}, v_{a,k}),$$

where k denotes the discrete time step.

Data sets. We used the data sets introduced in [15], which were collected during the operation of the real robot. Five sequences of samples starting from the initial state $\mathbf{x}_0 = (0, 0, 0)^\top$ were generated with a sampling period $T_s = 0.2$ s. In each sequence, we steered the robot by random inputs drawn from the domain $v_f \in [0, 0.3] \text{ m} \cdot \text{s}^{-1}$, $v_a \in [-1, 1] \text{ rad} \cdot \text{s}^{-1}$. Of these five sequences, a randomly chosen one was used to create the training data set, another one was used for the validation data set, and the remaining three sequences were used for the test data sets.

Prior knowledge. We use the prior knowledge defined for the TurtleBot in [15]. All three prior knowledge types that the x_{pos} variable should comply with are of the invariant type. This means that when the model for the x_{pos} variable is evaluated on the relevant constraint sample, it should always output the value equal to its original value. The following three types of prior knowledge about the x_{pos} were used:

- Steady-state behavior: If the control inputs, v_f and v_a , are set to zero, then the robot may change neither its position nor its heading. This is represented by the following equality constraint:

$$x_{pos} = f^{x_{pos}}(x_{pos}, y_{pos}, \phi, 0, 0).$$

- Axis-parallel moves: If the robot moves parallel to the y -axis, then its x_{pos} does not change. This is represented by the following equality constraints:

$$x_{pos} = f^{x_{pos}}(x_{pos}, y_{pos}, -\pi/2, v_f, 0),$$

$$x_{pos} = f^{x_{pos}}(x_{pos}, y_{pos}, \pi/2, v_f, 0).$$

- Turning on the spot: If the forward velocity is zero, the robot may not change its position. This is represented by the following equality constraint:

$$x_{pos} = f^{x_{pos}}(x_{pos}, y_{pos}, \phi, 0, v_a).$$

The values of the state variables x_{pos} , y_{pos} , ϕ , and of the control inputs v_f and v_a were randomly sampled within the same limits as for the training data. We generated 50 constraint samples for each prior knowledge type, so 150 samples in total.

2) *Magnetic manipulation:* The magnetic manipulation system, *magman*, consists of an iron ball moving on a rail and an electromagnet placed at a fixed position under the rail (Figure 3).

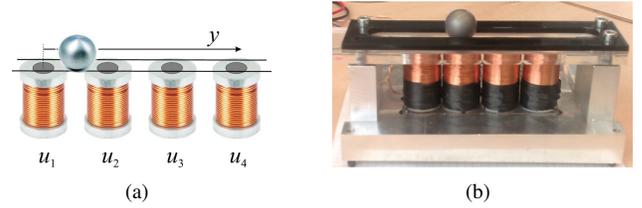


Fig. 3. Magman: A schematic (a) and a photo of the system (b).

The goal is to find a model of the nonlinear magnetic force affecting the ball, $f(x)$, as a function of the horizontal distance, x , between the iron ball and the electromagnet, given a constant current i through the coil. We use data measured on a real system and an empirical model $\hat{f}(x) = -ic_1x/(x^2 + c_2)^3$ proposed in the literature [30] as the *reference model*, see Figure 4. Parameters c_1 and c_2 were found empirically for the given system.

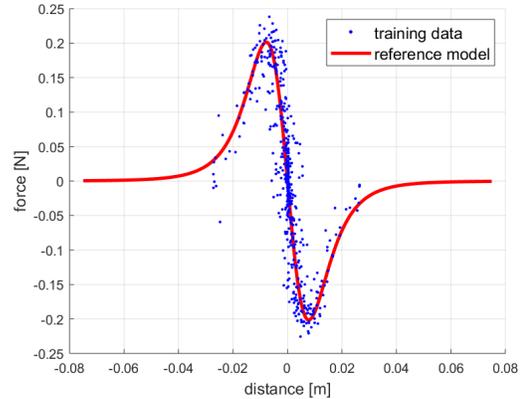


Fig. 4. Training data and the reference model for the *magman* problem.

Data sets. The region of operation considered for the *magman* spans over the interval $-0.075 \text{ m} \leq x \leq 0.075 \text{ m}$. However, only its small part, $\mathbb{D}_t = [-0.027, 0.027] \text{ m}$, is covered by the data measured on the real system [31], see Figure 4. A proper form of the model outside the sampled interval is governed by the constraints imposed on the model, see below. The whole data set of 858 measured samples was split into two sets in the ratio of 7:3. The larger one was further split into the D_t and D_v sets so that $|D_t| = 400$

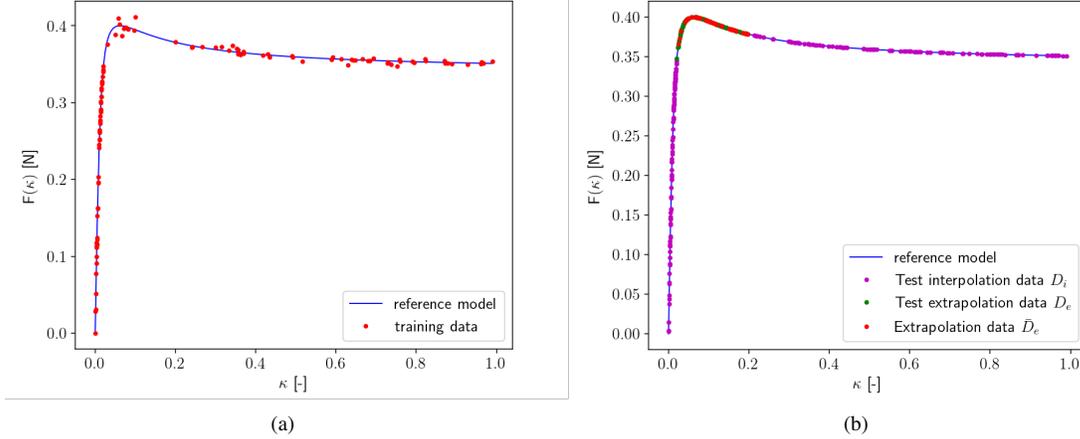


Fig. 5. Magic formula reference model and the data sets. (a) Training and validation data. (b) Interpolation test data and extrapolation data.

and $|D_v| = 201$. The smaller one was used for the test interpolation data set D_i . Additionally, two data sets, \bar{D}_e and D_e , with samples from the extrapolation domain $\mathbb{D}_e = [-0.075, -0.027] \cup [0.027, 0.075]$ m were generated with the following sizes $|\bar{D}_e| = 40$ and $|D_e| = 200$. The target values of \bar{D}_e and D_e samples were determined using the reference model.

Prior knowledge. Five types of prior knowledge were defined. The model sought returns positive values on the interval $[-0.075, 0]$ m and negative ones on the interval $[0, 0.075]$ m. It is monotonically increasing on the intervals $[-0.075, -0.02]$ m and $[0.02, 0.075]$ m and monotonically decreasing on the interval $[-0.006, 0.006]$ m. Finally, the model's function goes through the origin, i.e., $f(0) = 0$, and approaches zero at negative and positive infinity, respectively, which is represented by the constraints $f(-0.075) = 10^{-3}$ and $f(0.075) = -10^{-3}$, respectively. The constraint set contains 50 samples for the positive values, negative values, increasing monotonicity, and decreasing monotonicity plus 3 samples for the desired exact values, resulting in 203 constraint samples.

3) *Resistors*: This problem was proposed in [8] to test SR method based on genetic programming with formal constraints. Originally, it considers a sparse set of noisy samples derived using the equivalent resistance of two resistors in parallel, $r = r_1 r_2 / (r_1 + r_2)$, used here as a *reference model*. The goal is to find such a model $f(r_1, r_2)$ that fits the training data and exhibits the same properties as the *reference model*, see below.

Data sets. Here, we use two variants of the data set used within the learning process. One with only 10 samples and the other one with 500 samples. The values of r_1 and r_2 are sampled uniformly from the interval $[0.0001, 20] \Omega$. The target values are disturbed with a noise randomly generated with a normal distribution $\mathcal{N}(0, 0.05 \sigma_y)$, where σ_y is a standard deviation of the original output values obtained with the *reference model*. When using the large set, it is split into D_t and D_v in the ratio of 7:3. When the smaller one is used then eight samples go for D_t , and the remaining two samples are in D_v . The interpolation test set D_i is sampled from the

training domain as well. Two data sets are sampled from the extrapolation domain $\mathbb{D}_e = [20, 40]^2$, \bar{D}_e with 40 samples and D_e with 500 samples.

Prior knowledge. We used the following three prior knowledge types as defined in [8] and used as well in [14]:

- symmetry with respect to arguments:

$$f(r_1, r_2) = f(r_2, r_1),$$

- domain-specific constraint:

$$r_1 = r_2 \implies f(r_1, r_2) = \frac{r_1}{2},$$

- domain-specific constraint:

$$f(r_1, r_2) \leq r_1, f(r_1, r_2) \leq r_2.$$

The constraint set contains a total of 150 constraint samples, 50 for each constraint.

4) *Anti-lock braking system – magic formula*: This problem considers the control of an anti-lock braking system and particularly the longitudinal force $F(\kappa)$ as the function of the wheel slip κ . The force $F(\kappa)$ is described by the ‘magic’ formula of the following form

$$F(\kappa) = m g d \sin(c \arctan(b(1 - e)\kappa + e \arctan(b\kappa))), \quad (8)$$

where b , c , d and e are road surface-specific constants. The magic formula is an empirical model commonly used to simulate steady-state tire forces and moments. By adjusting the function coefficients, the same special function can be used to describe longitudinal and lateral forces (sine function) and self-aligning moment (cosine function). Here, we consider the *reference model* used in [32] with $m = 407.75$ kg, $g = 9.81$ m \cdot s $^{-1}$, and the slip force parameters $(b, c, d, e) = (55.56, 1.35, 0.4, 0.52)$, see Figure 5, which correspond to a wet asphalt for a water level of 3 mm [33].

Data sets. A data set of 110 samples generated using the reference model (8). The whole set was divided into D_t and D_v in the ratio of 4:1. The data are intentionally sampled unevenly. The steep left and the right flat region κ in the interval $[0, 0.02]$ and $[0.2, 0.99]$, representing the interpolation domain \mathbb{D}_i , are densely sampled with 50 samples each. Target values of these samples are disturbed with a noise randomly

generated with a normal distribution $\mathcal{N}(0, 0.0025)$. Contrary, the peak of the function with κ values in the interval $[0.03, 0.1]$ is covered very sparsely in the data with only 10 samples. Moreover, a larger noise drawn from $\mathcal{N}(0, 0.005)$ is added to the target values. This corresponds to the fact that, in reality, the system is unstable around the peak and it is hard to collect precise data there. Thus, the peak represents the extrapolation domain \mathbb{D}_e in the sense that it is rather poorly defined by the data. Again, the data deficiency is compensated in our method by the use of prior knowledge, see below. Additionally, three data sets for models' performance evaluation were used: D_i of size 200 sampled from \mathbb{D}_i and data sets \bar{D}_e and D_e of sizes 40 and 100, respectively, sampled from \mathbb{D}_e . The target values of the samples were generated as the noiseless output of the reference model.

Prior knowledge. Three types of prior knowledge were defined for this problem, reflecting the key properties of the model sought. The model should return zero for $\kappa = 0$. Further, in the right part of \mathbb{D}_i , the model is monotonically decreasing and approaching from above a certain value in the limit. So, its second derivative in this region should be positive. The last property of the model is that it has a single maximum located within \mathbb{D}_e . This can be described by a constraint that enforces the model to be concave everywhere in \mathbb{D}_e . The constraint set contains a total of 101 constraint samples, 1 sample for the exact value at $\kappa = 0$ and 50 samples for each of the other two constraints.

We briefly illustrate the implementation of constraints on the monotonically decreasing function with a positive second derivative property. As described in Section IV-A, a set of N_j constraint samples is generated for each constraint j . Here, the samples have the form $(\kappa_l, \kappa_c, \kappa_r)$, where κ_c is randomly sampled in \mathbb{D}_e and $\kappa_l = \kappa_c - \delta$, $\kappa_r = \kappa_c + \delta$, and $\delta = 0.001$. For each such constraint sample k , the error value e_k is calculated as

$$e_k = \max((f(\kappa_r) - f(\kappa_c)), 0) + \max((f(\kappa_c) - f(\kappa_l)), 0) \\ + \max((f(\kappa_c) - f(\kappa_r)) - (f(\kappa_l) - f(\kappa_c)), 0),$$

where the first line represents contributions for a non-decreasing property observed on the given constraint data triple $(\kappa_l, \kappa_c, \kappa_r)$ and the second line represents a penalty for a negative second derivative observed on the constraint sample. Then, the corresponding $cterm_j$ is calculated as the root-mean-square error over all e_k observed for the constraint.

D. Experiment set up

1) *Network architecture:* We used a network architecture with three hidden layers, denoted as the 'general' architecture. The first two hidden layers contain four elementary functions $\{\sin, \tanh, \text{ident}, *\}$, each with two copies. The third hidden layer contains, in addition to that, one division unit. The output layer contains a single identity unit ident . Note, the ident unit calculates a weighted sum of its inputs so it can realize both the addition and subtraction units. The same architecture was used for all test problems but the `magic` one. There we used a function set with \arctan instead of \tanh in order to make the set-up consistent with the magic formula reference

model (8). Moreover, for experiments with the `resistors` problem, we also used an architecture with a limited function set $\{\text{ident}, *, /\}$, denoted as the 'informed' architecture, that corresponds to the one used in [14]. Its name reflects the fact that we know the minimum set of elementary functions needed to compose the correct formula. The first two hidden layers contain 4 copies of the multiplication and ident unit each. The third hidden layer contains 3 copies of each of those two unit types. The number of units was chosen so that the number of learnable weights of the two architectures was as close as possible. Thus, the first and the second architecture comprises 396 and 403 learnable weights in total, respectively.

2) *Algorithms' configuration:* The algorithms were tested with the following parameter setting:

- `N4SR`: $N_{init} = 2000$, $N_e = 20$, $N_f = 980$ and $E = 87$ for epoch-wise variants, $N_f = 86980$ and $E = 1$ for single epoch variants, $N_{final} = 1000$, $N_w = 10$, $r_{s/t} = 0.5$, $r_{c/t} = 0.5$, $r_{r/t} = 0.5$. The parameters are chosen so that the total number of iterations is always $T = 90000$.
- `EQL \ddagger` : We adopted the configuration used in [18] with the exception that the total number of iterations T was set to 90000. The topology of the network was set the same as in the corresponding `N4SR` experiments, with the exception that the division unit is not in the third hidden layer. Instead, it is the only unit of the output layer, which is a design feature of `EQL \ddagger` .
- `mSNGP-LS`: We adopted the configuration used in [14] with a few modifications. The set of elementary functions is set to comply with the set of elementary functions used in `N4SR` networks for the given problem. The maximum model's complexity is bounded from above by two parameters, the maximum number of features $n_f = 5$ and the maximum feature's depth $\delta = 3$, used with the same values for all test problems. Note the maximum feature's depth is equal to the number of hidden layers of the `N4SR` networks. Thus, when the features are aggregated in the final model, then its maximum possible depth is the same as the maximum depth of the `N4SR` models.

3) *Experiments evaluation:* Fifty independent runs were carried out for each tested pair of the method and the training data set. The best model is returned at the end of each run according to the model selection strategy used. For the `mSNGP-LS`, we adopt the selection method proposed in [15]. It uses two performance metrics — the RMSE calculated on the validation data set D_v and the constraint violation error observed on the validation constraint data set. The validation constraint data set is generated in the same way as the constraint samples used in `N4SR` methods and the constraint violation error is calculated as the sum of $cterm_j$ values for all constraints in T^c . Then, the `mSNGP-LS` method chooses among all models in the last population of the run the model that has the best validation RMSE out of all models with the constraint violation value less than the population's median.

On the `turtlebot` problem, the models' performance is presented using a simulation RMSE, which is calculated as the root-mean-square-error between $x_{pos,k+1}$ and $\hat{x}_{pos,k+1}$ for all points $k = 1 \dots |D| - 1$ in the data sequence D , where

TABLE I
RESULTS OBTAINED WITH EQL[±], mSNGP-LS, AND N4SR METHODS WITH THE GENERAL ARCHITECTURE ON THE TURTLEBOT PROBLEM. THE COMPLEXITY IS GIVEN AS THE NUMBER OF ACTIVE LINKS / NUMBER OF ACTIVE UNITS.

Method	complexity	N_{nt}	$RMSE_{valid}$	$RMSE_{test1}$	$RMSE_{test2}$	$RMSE_{test3}$	$RMSE_{sum}$
N4SR-ACYE	12 / 4	50	0.20	0.51	0.54	0.42	1.45
N4SR-ACYS	21 / 6	50	0.30	0.41	0.50	0.37	1.20
N4SR-AEYE	12 / 4	50	0.10	0.39	0.44	0.37	1.12
N4SR-AEYS	21 / 6	50	0.13	0.43	0.52	0.39	1.34
N4SR-AENE	16 / 4	50	5.97	2.40	1.01	2.30	5.52
N4SR-SCYE	22 / 5	49	0.51	0.52	0.51	0.45	1.43
N4SR-SCYS	30 / 8	50	0.68	0.61	0.50	0.58	1.70
N4SR-SENE	20 / 4	49	3.72	2.71	1.71	2.74	7.13
EQL [±]	30 / 18	50	1.73	2.61	1.97	2.26	8.68
mSNGP-LS	NA	50	0.03	0.11	0.17	0.15	0.42

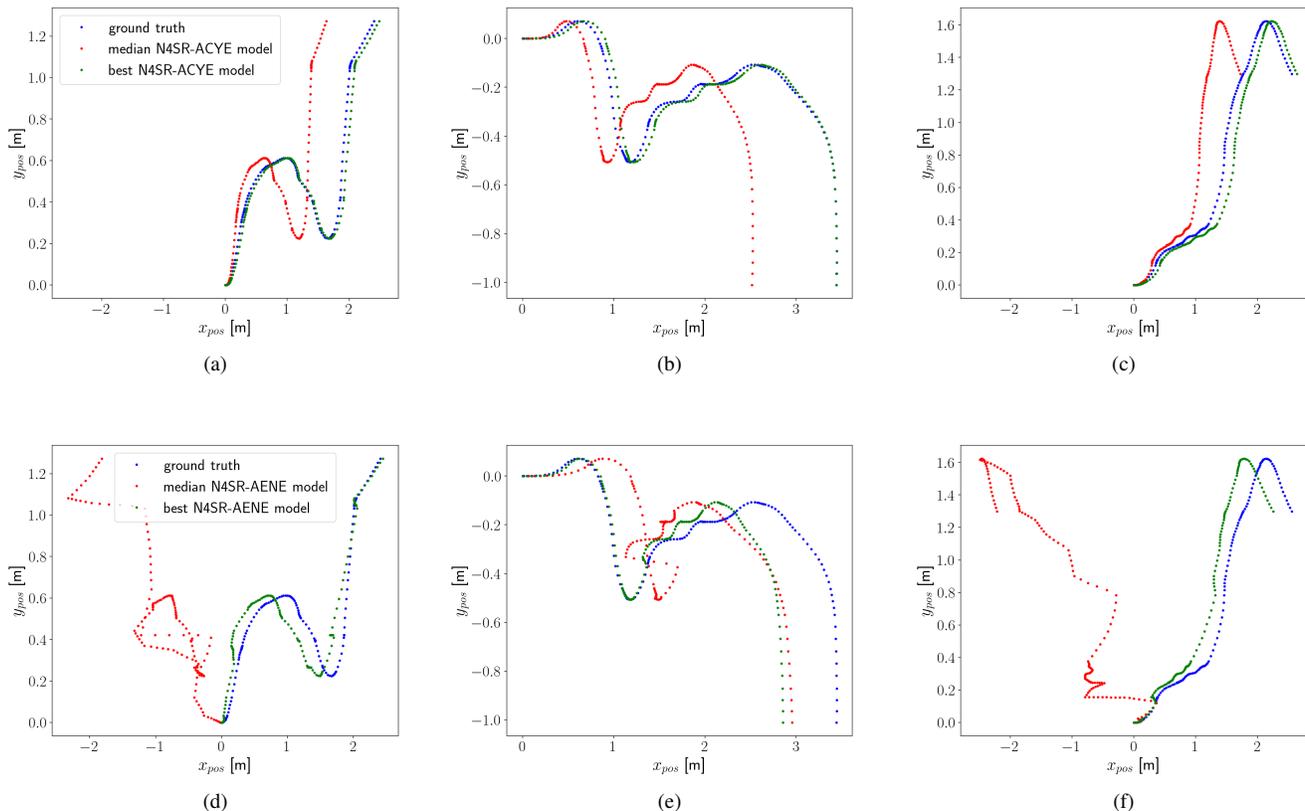


Fig. 6. Examples of the turtlebot simulation trajectories generated using the $f^{x_{pos}}(\cdot)$ models obtained with N4SR-ACYE (a)-(c) and N4SR-AENE (d)-(f), respectively, on the three test data sets. Out of all best-of-run models collected over all runs for each experiment, a trajectory of the median model w.r.t. $RMSE_{sum}$ (shown in red) and a trajectory of the model with the best $RMSE_{sum}$ value (shown in green) are presented. The reference ground truth trajectory is shown in blue.

$\hat{x}_{pos,k+1}$ is the value predicted by the model according to $\hat{x}_{pos,k+1} = f^{x_{pos}}(\hat{x}_{pos,k}, y_{pos,k}, \phi_k, v_{f,k}, v_{a,k})$.

Finally, the median value of the following performance measures over the fifty best-of-run models are presented:

- complexity – model complexity defined as the number of active weights and active units, respectively.
- $RMSE_{int}$, $RMSE_{ext}$, $RMSE_{int+ext}$ – RMSE calculated on test data D_i , D_e , and $D_i \cup D_e$, respectively.
- $RMSE_{valid}$, $RMSE_{test1}$, $RMSE_{test2}$, $RMSE_{test3}$ – simulation RMSE values calculated on the turtlebot problem, and the $RMSE_{sum}$ value calculated as the sum of all $RMSE_{test}$ values.

- N_{nt} – the number of runs in which the method yields a nontrivial model, i.e., a model with more than one active link.

E. Results

In this section, tabular results are presented for all of the compared methods, accompanied by examples of models produced by the N4SR variants. In the tables, the **N4SR-ACYE** is highlighted in bold as this is the N4SR variant that corresponds to the proposed method.

1) *TurtleBot*: Table I shows results obtained with the compared algorithms on the turtlebot problem.

TABLE II

RESULTS OBTAINED WITH EQL[‡], mSNGP-LS, AND N4SR METHODS WITH THE GENERAL ARCHITECTURE ON THE MAGMAN PROBLEM. THE COMPLEXITY IS GIVEN AS THE NUMBER OF ACTIVE LINKS / NUMBER OF ACTIVE UNITS.

Method	complexity	N_{nt}	$RMSE_{int}$	$RMSE_{ext}$	$RMSE_{int+ext}$
N4SR-ACYE	9 / 5	19	$5.78 \cdot 10^{-2}$	$4.84 \cdot 10^{-3}$	$4.34 \cdot 10^{-2}$
N4SR-ACYS	7.5 / 5	18	$5.84 \cdot 10^{-2}$	$5.20 \cdot 10^{-3}$	$4.39 \cdot 10^{-2}$
N4SR-AEYE	9 / 5	19	$5.83 \cdot 10^{-2}$	$1.42 \cdot 10^{-3}$	$4.38 \cdot 10^{-2}$
N4SR-AEYS	7.5 / 5	18	$5.87 \cdot 10^{-2}$	$6.26 \cdot 10^{-4}$	$4.41 \cdot 10^{-2}$
N4SR-AENE	7 / 5	50	$5.70 \cdot 10^{-2}$	$2.46 \cdot 10^{-1}$	$1.68 \cdot 10^{-1}$
N4SR-SCYE	8.5 / 5	20	$5.63 \cdot 10^{-2}$	$5.02 \cdot 10^{-2}$	$5.40 \cdot 10^{-2}$
N4SR-SCYS	9 / 6	20	$5.65 \cdot 10^{-2}$	$4.65 \cdot 10^{-2}$	$5.22 \cdot 10^{-2}$
N4SR-SENE	7.5 / 5	20	$5.65 \cdot 10^{-2}$	$1.38 \cdot 10^{-1}$	$1.01 \cdot 10^{-1}$
EQL [‡]	30 / 18	50	$5.60 \cdot 10^{-2}$	$6.12 \cdot 10^{-2}$	$5.86 \cdot 10^{-2}$
mSNGP-LS	NA	50	$5.59 \cdot 10^{-2}$	$4.61 \cdot 10^{-2}$	$5.18 \cdot 10^{-2}$

We can see that the variants with the adaptive weighting scheme have significantly better validation RMSE than the static ones. However, they all perform comparably on the test sequences. The adaptive variants also generate significantly simpler models in terms of the number of active weights and active units. The proposed constraint satisfaction-based final model selection method works comparably to the extrapolation-based one; see N4SR-ACYE vs. N4SR-AEYE and N4SR-ACYS vs. N4SR-AEYS. This is an important observation that demonstrates the ability of the method to reliably identify the final model even when no extrapolation domain samples are provided. Further, we observe a clear benefit of using prior knowledge for learning. When no prior knowledge is used, poor models are produced, see N4SR-SENE, N4SR-AENE, and the EQL[‡] models. Finally, the best-performing method on this problem is the mSNGP-LS. We discuss a possible reason for what causes the difference between N4SR and mSNGP-LS in Section V-F.

Figure 6 shows examples of trajectories generated with selected models. Particularly, the median and the best models w.r.t. $RMSE_{sum}$ produced by N4SR-ACYE (Figure 6a-c) and N4SR-AENE (Figure 6d-f) are presented. These plots clearly illustrate the benefits of using prior knowledge. One can see that both the median and the best model produced by N4SR-ACYE generate trajectories that accurately imitate the shape of the ground truth one. Moreover, the trajectories generated with the best model have a minimal offset from the ground truth one along the whole trajectory. On the contrary, trajectories generated with the models produced by N4SR-AENE exhibit larger discrepancies in terms of both the shape and the offset.

The raw analytic expression represented by the best N4SR-ACYE model is

$$x_{pos,k+1} = -0.365((-0.470 \sin(0.993 \phi_k + 1.552)) \\ (0.994 v_{f,k})) + 1.00008 x_{pos,k},$$

and can be further simplified to

$$x_{pos,k+1} = 0.171 v_{f,k} \sin(0.993 \phi_k + 1.552) \\ + 1.00008 x_{pos,k}.$$

We can see that the best N4SR-AENE model, represented by the following simplified analytic expression, is much more

complex:

$$x_{pos,k+1} = 0.170 - 0.088 v_{f,k} + 1.0003 x_{pos,k} + 0.169 \\ (0.555 \sin(-0.140 \phi_k^2 + 0.871 \phi_k v_{f,k} + 0.087 \phi_k \\ + 2.587 v_{f,k} + 0.281) + 0.204) (1.309 v_{f,k} + 0.053 y_{pos,k} \\ - 0.223 (0.535 \phi_k + 1.588) (0.271 \phi_k - 1.691 v_{f,k} + 0.489) \\ + 0.096 \sin(-0.140 \phi_k^2 + 0.871 \phi_k v_{f,k} + 0.087 \phi_k \\ + 2.587 v_{f,k} + 0.281)).$$

2) *Magnetic manipulation*: The results obtained on the magman problem are shown in Table II. We can see that N4SR-ACYE and N4SR-ACYS produce models that have the best performance on the extrapolation data as its $RMSE_{ext}$ is, by order of magnitude, better than the EQL[‡], mSNGP-LS, and N4SR variants using the static weighting scheme. Only models obtained with N4SR-AEYE and N4SR-AEYS exhibit better extrapolation performance. But this can be attributed to the fact that these variants have certain knowledge of the models' extrapolation performance already when selecting the final model of each run. N4SR-ACYE performs best, even better than mSNGP-LS, in terms of the overall $RMSE_{int+ext}$ metric. Again, the constraint satisfaction-based final model selection performs equally to the extrapolation-based one.

All N4SR variants but N4SR-AENE exhibit the same behavior in that they are vulnerable to collapsing to a trivial model with only one active weight, particularly the one of the output unit's bias link. Only about 40% of the runs end up with nontrivial models. The N4SR-AENE finds a nontrivial model in all runs, but these fit well only in the interpolation domain. Outside the interpolation domain, the models go wild since the method cannot use any helpful information to direct the search towards better models.

Interestingly, EQL[‡] models achieve comparable performance to N4SR and mSNGP-LS in terms of the RMSE metrics. This can also be attributed to its feature that the models are selected based on the few known extrapolation points \bar{D}_e . Despite its rather good test RMSE values, the models are still not truly useful as they do not comply with the desired properties defined for the extrapolation domain, see Figure 7c. Moreover, the EQL[‡] models are much more complex than the N4SR ones.

Figure 7 also shows models generated by N4SR-SCYE, N4SR-ACYE, and mSNGP-LS. Again, the median and the

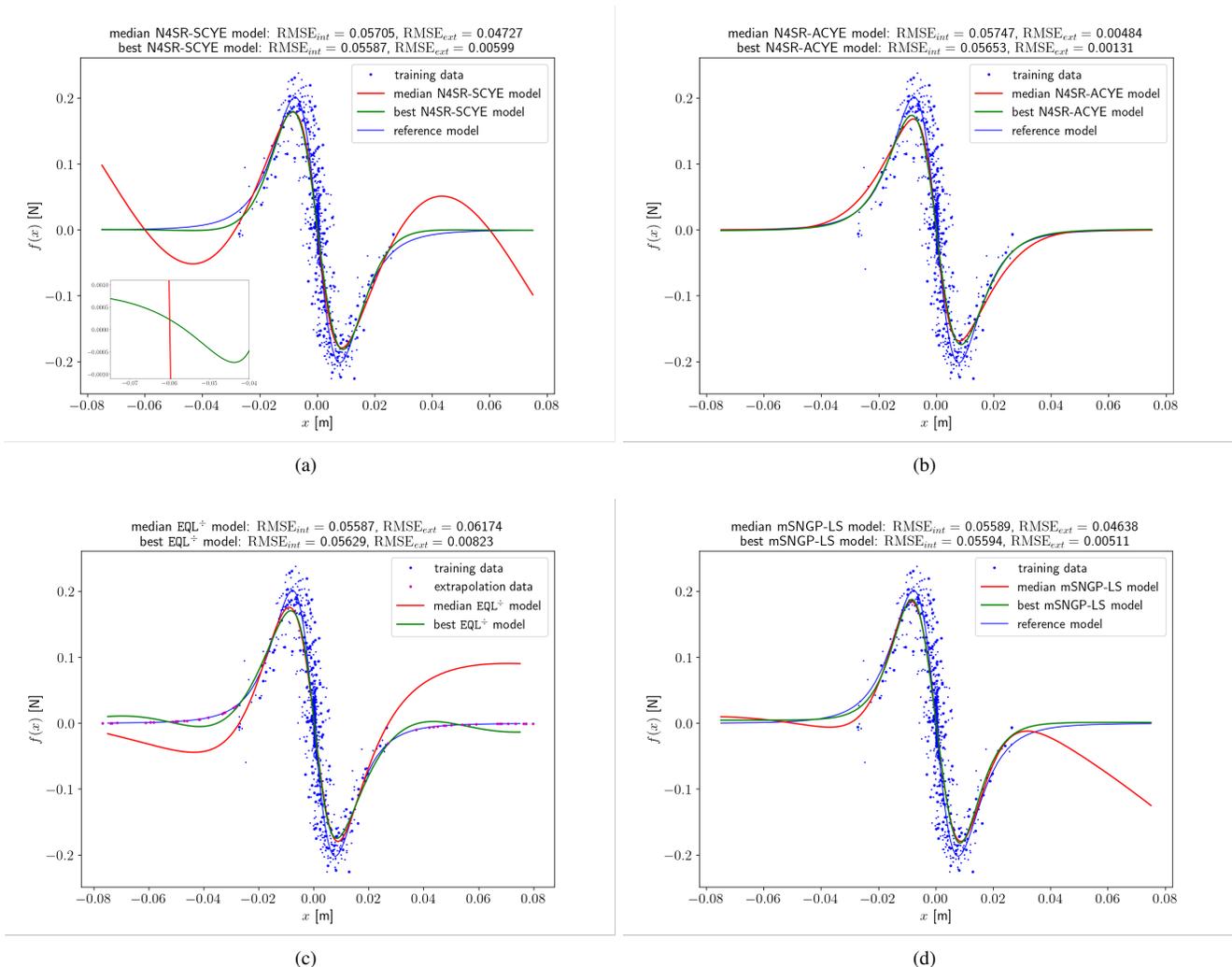


Fig. 7. Examples of models generated for the magman problem with N4SR-SCYE (a), N4SR-ACYE (b), EQL⁺ (c), and mSNGP-LS (d) method. Out of all best-of-run models collected over all runs for each experiment, the median model w.r.t. $RMSE_{int+ext}$ (shown in red) and the model with the best $RMSE_{int+ext}$ value (shown in green) are presented.

best models w.r.t. $RMSE_{int+ext}$ over the set of the non-trivial models are presented. One can see that neither of the N4SR-SCYE and mSNGP-LS can reliably produce nontrivial models that are perfect in terms of the increasing monotonicity constraint. In particular, N4SR-SCYE failed to generate such a model in all runs and mSNGP-LS succeeded only in 3 out of 50 runs. Contrary to that, only one nontrivial model produced by N4SR-ACYE did not comply with this monotonicity constraint. For illustration, we show the best N4SR-ACYE model

$$f(x) = -0.713 \sin(4.729 \tanh(8.035 \tanh(5.835 x))) \\ + 1.149 \tanh(8.035 \tanh(5.835 x)) \\ - 1.863 \tanh(2.850 \tanh(8.035 \tanh(5.835 x)))$$

and the best N4SR-SCYE model

$$f(x) = -1.210 \sin(4.755 \tanh(46.011 x) - 0.027) \\ - 1.448 \tanh(1.220 \tanh(46.011 x)).$$

3) *Resistors*: Results obtained on the resistors problem are shown in Table III for the general architecture and in

Table IV for the informed architecture. The first observation is that the NN architecture matters. If the topology contains only the units that are necessary to compose the desired expression, then the N4SR methods produce significantly better models both in terms of the test accuracy and the model's complexity. N4SR-ACYE is able to find very good models even with only ten training samples measured in the interpolation domain. Moreover, N4SR-ACYE is much better than N4SR-SCYE and its models are simpler. This again demonstrates the advantages of the adaptive weighting strategy over the static one. On this problem, N4SR methods are stable as only a few runs yield the trivial model. Again, the constraint satisfaction-based model selection method is comparable to the extrapolation-based one. Interestingly, N4SR variants without prior knowledge perform much better than EQL⁺, which completely fails to find good models. Note that both methods use the extrapolation data for choosing the final model. Altogether, mSNGP-LS performs best on this problem except in the case when the informed architecture is used with the large training data. There, the best method is the N4SR-ACYE.

TABLE III

RESULTS OBTAINED WITH EQL[±], mSNGP-LS, AND N4SR METHODS WITH THE GENERAL ARCHITECTURE ON THE RESISTORS PROBLEM. THE COMPLEXITY IS GIVEN AS THE NUMBER OF ACTIVE LINKS / NUMBER OF ACTIVE UNITS.

$ D_t \cup D_v $	Method	Complexity	N_{nt}	$RMSE_{int}$	$RMSE_{ext}$	$RMSE_{int+ext}$
500	N4SR-ACYE	11 / 3	50	0.050	0.128	0.097
500	N4SR-ACYS	9 / 4	47	0.297	0.956	0.766
10	N4SR-ACYE	22 / 6	50	0.610	1.290	1.038
10	N4SR-ACYS	21 / 7	50	0.620	1.830	1.386
500	N4SR-AEYE	11 / 3	50	0.061	0.076	0.069
500	N4SR-AEYS	9 / 4	47	0.287	0.887	0.702
500	N4SR-AENE	36.5 / 9	50	0.085	0.702	0.502
10	N4SR-AEYE	22 / 6	50	0.623	1.157	0.948
10	N4SR-AEYS	21 / 7	50	0.657	1.749	1.311
10	N4SR-AENE	24 / 7	50	1.040	2.370	1.850
500	N4SR-SCYE	47 / 10	45	0.074	0.304	0.221
500	N4SR-SCYS	50 / 11	45	0.074	0.360	0.258
500	N4SR-SENE	47 / 11	44	0.091	0.781	0.556
10	N4SR-SCYE	50 / 10	47	0.351	0.467	0.432
10	N4SR-SCYS	48 / 11	47	0.398	0.508	0.469
10	N4SR-SENE	29 / 8	45	1.050	2.450	1.900
500	EQL [±]	30 / 18	50	0.550	8.460	6.010
500	mSNGP-LS	NA	50	0.023	0.032	0.029
10	EQL [±]	30 / 18	50	18.820	17.420	22.810
10	mSNGP-LS	NA	50	0.008	0.009	0.008

TABLE IV

RESULTS OBTAINED WITH EQL[±], mSNGP-LS, AND N4SR METHODS WITH THE INFORMED ARCHITECTURE ON THE RESISTORS PROBLEM. THE COMPLEXITY IS GIVEN AS THE NUMBER OF ACTIVE LINKS / NUMBER OF ACTIVE UNITS.

$ D_t \cup D_v $	Method	complexity	N_{nt}	$RMSE_{int}$	$RMSE_{ext}$	$RMSE_{int+ext}$
500	N4SR-ACYE	10 / 3	50	0.012	0.021	0.017
500	N4SR-ACYS	8 / 3	50	0.011	0.023	0.018
10	N4SR-ACYE	12 / 3	50	0.109	0.125	0.118
10	N4SR-ACYS	11 / 3	50	0.107	0.119	0.113
500	N4SR-AEYE	10 / 3	50	0.009	0.015	0.012
500	N4SR-AEYS	8 / 3	50	0.009	0.016	0.014
500	N4SR-AENE	12.5 / 3	50	0.050	0.040	0.063
10	N4SR-AEYE	12 / 3	50	0.091	0.096	0.094
10	N4SR-AEYS	11 / 3	50	0.099	0.101	0.099
10	N4SR-AENE	22 / 4	50	0.966	2.320	2.496
500	N4SR-SCYE	19 / 4	50	0.041	0.099	0.072
500	N4SR-SCYS	19 / 5	49	0.042	0.137	0.101
500	N4SR-SENE	19 / 4	49	0.069	0.082	0.079
10	N4SR-SCYE	30 / 5	49	0.481	0.728	0.818
10	N4SR-SCYS	30.5 / 6	49	0.495	0.849	0.819
10	N4SR-SENE	29 / 5	45	2.360	2.920	3.250
500	EQL [±]	45 / 15	50	0.550	6.090	4.340
500	mSNGP-LS	NA	50	0.012	0.036	0.027
10	EQL [±]	45 / 15	50	5.680	14.060	12.990
10	mSNGP-LS	NA	50	0.008	0.009	0.008

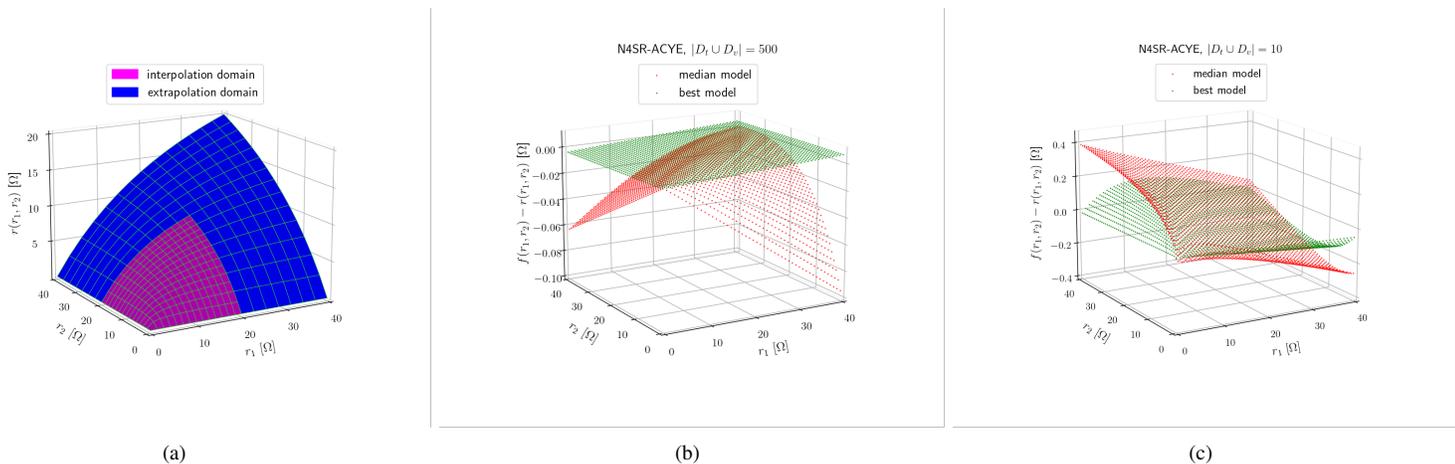


Fig. 8. Examples of models generated for the resistors problem with N4SR-ACYE method. (a) is the reference model, (b) shows models attained when using the learning data set of size $|D_t \cup D_v| = 500$, and (c) shows models attained with the learning data set of size 10. Out of all best-of-run models collected over all runs for each experiment, the median model w.r.t. $RMSE_{int+ext}$ (shown in red) and the model with the best $RMSE_{int+ext}$ value (shown in green) are presented.

Figure 8 demonstrates the performance of the median and the best w.r.t. the $RMSE_{int+ext}$ models generated by the N4SR-ACYE method using small and large training data sets, respectively. It shows the difference between the N4SR model and the reference model on the whole domain. One can see that the models are doing well on the interpolation as well as on the extrapolation domain, even when trained on the small data set. The raw analytic expression represented by the best N4SR-ACYE model is

$$f(r_1, r_2) = 0.767((-1.012((-0.745 r_1 + 1.080 r_2) \\ (-0.962 r_1 + 0.695 r_2)))/(2.346 r_1 + 2.346 r_2)) \\ + 0.237 r_1 + 0.248 r_2,$$

which can be rewritten as

$$f(r_1, r_2) = (0.0004 r_1^2 + 2.347 r_1 r_2 - 0.0002 r_2^2) \\ /(2.346 r_1 + 2.346 r_2).$$

Note that this is not the right expression but it is a close approximation of the reference model. Please, see more on this in Section V-F.

4) *Anti-lock braking system – magic formula*: Results obtained on the `magic` problem are summarized in Table V. The overall best method is again mSNGP-LS. As for the N4SR, one can observe that the variants using the static weighting method slightly outperform the ones using the adaptive scheme in terms of the RMSE measures. Also, the number of nontrivial models is about twice as high for the variants with the static weighting method. This indicates that setting up the loss terms weights just once for the whole run was sufficient in this case. However, the models generated with the static method are larger than the ones produced with the adaptive scheme.

On the one hand, the single-epoch learning strategy works equally to the epoch-wise one when used together with the static weighting scheme, compare N4SR-SCYS to N4SR-SCYE. On the other hand, it gets much worse than the epoch-wise learning strategy when used with the adaptive weighting scheme, compare N4SR-ACYS to N4SR-ACYE. Then its median performance becomes even slightly worse than the EQL⁺ method. Still, its above-average models are much better than the EQL⁺ ones, see the green curves in Figures 9c and 9d.

Another observation is that even the N4SR-AENE method performs comparably to the N4SR-ACYE one in terms of the RMSE measures. However, a detailed visual inspection reveals imperfections of the N4SR-AENE models. In Figure 9a, one can see that neither the median nor the best model fully complies with the constraints imposed on the model. On the contrary, N4SR-ACYE provides the most reliable models, see Figure 9b. For illustration, we show the raw analytic expression of the best N4SR-ACYE model

$$f(\kappa) = (-0.429 \sin(-0.593((-1.470 \arctan(-31.132 \kappa \\ -0.126) - 0.741)(1.462 \arctan(-31.132 \kappa - 0.126) \\ + 0.753)) + 1.5 \arctan(-31.132 \kappa - 0.126)))$$

and its equivalent obtained by a symbolic simplification

$$f(\kappa) = -0.429 \sin(1.275 \arctan(-31.132 \kappa - 0.126))^2 \\ + 2.799 \arctan(-31.132 \kappa - 0.126) + 0.331).$$

F. Discussion

The results obtained with the proposed N4SR-ACYE method are very promising. We observed that the method is capable of finding sparse and accurate models that exhibit desired properties defined as the prior knowledge for the given problem. Nevertheless, we also observe that the GP-based mSNGP-LS approach often outperforms N4SR-ACYE. Here, we analyze the results and propose our hypothesis on why it is so.

A detailed inspection of the final models obtained with mSNGP-LS on `turtlebot` problem revealed that the analytic expressions of the best-performing models are composed of simple elementary structures. By ‘simple’, we mean that the arithmetic operators and the \sin and \tanh functions, which are at the lower levels of the expression tree, operate on ‘raw’ variables (i.e., the variables are weighted by the coefficient of 1). An example is the best mSNGP-LS model

$$x_{pos,k+1} = 1.0011 x_{pos,k} - 0.0833 \sin(\phi_k - v_{f,k}) \\ + 0.0815 \sin(\phi_k + v_{f,k}) \\ + 0.0018 \sin(\phi_k + \sin(v_{a,k})) \\ + 0.0029 \tanh(v_{a,k} + 1.0) - 0.003$$

composed of simple elementary structures $\sin(\phi_k - v_{f,k})$, $\sin(\phi_k + v_{f,k})$, $\sin(\phi_k + \sin(v_{a,k}))$, and $\tanh(v_{a,k} + 1.0)$. Note that the mSNGP-LS works in a ‘bottom-up’ manner. It starts the search process with many of the elementary structures easily available and tries to combine them in the final expression optimally. Thus, it works with building blocks that are already there and ‘just’ searches for their best combination.

On the contrary, for NN-based SR approaches of the N4SR type, it is hard to converge to expressions like this. The method can be seen as a ‘top-down’ approach. It starts with the complete NN topology where each function unit has its z node(s) realized as the random affine transformation of *all* previous layer’s units outputs. Moreover, the learned weights are randomly initialized to rather small values, thus far from the desired value of 1. Then, it has to carefully eliminate all the useless units and all useless inputs to each active function unit through many iterations of the gradient-based optimization process to get a sparse model composed of simple elementary structures. This leads to models like this:

$$x_{pos,k+1} = -0.3654(-0.4699 \sin(\mathbf{0.9933} \phi_k \\ + 1.5518)) \mathbf{0.9935} v_{f,k} + \mathbf{1.00008} x_{pos,k}.$$

This is the best N4SR-ACYE model on the `turtlebot` problem. We can see that the variables’ multiplication coefficients are set to values very close to but not exactly one.

Another analysis revealed that on the `resistors` problem, N4SR-ACYE could hardly find the models that perfectly fit the expression of the reference model. In particular, only 3 out of 50 runs of N4SR-ACYS converged to the model

TABLE V
RESULTS OBTAINED WITH EQL^{\ddagger} , $mSNGP$ -LS, AND N4SR METHODS WITH THE GENERAL ARCHITECTURE ON THE $MAGIC$ PROBLEM. THE COMPLEXITY IS GIVEN AS THE NUMBER OF ACTIVE LINKS / NUMBER OF ACTIVE UNITS.

Method	complexity	N_{nt}	$RMSE_{int}$	$RMSE_{ext}$	$RMSE_{int+ext}$
N4SR-ACYE	11 / 4	22	$9.76 \cdot 10^{-3}$	$1.16 \cdot 10^{-2}$	$1.06 \cdot 10^{-2}$
N4SR-ACYE	10 / 4	24	$5.42 \cdot 10^{-2}$	$6.36 \cdot 10^{-2}$	$6.04 \cdot 10^{-2}$
N4SR-AEYE	11 / 4	24	$1.31 \cdot 10^{-2}$	$1.06 \cdot 10^{-2}$	$1.35 \cdot 10^{-2}$
N4SR-AEYS	10 / 4	24	$5.63 \cdot 10^{-2}$	$4.96 \cdot 10^{-2}$	$5.76 \cdot 10^{-2}$
N4SR-AENE	18 / 5	50	$1.04 \cdot 10^{-2}$	$7.50 \cdot 10^{-3}$	$1.08 \cdot 10^{-2}$
N4SR-SCYE	19 / 5	45	$5.29 \cdot 10^{-3}$	$1.01 \cdot 10^{-2}$	$8.36 \cdot 10^{-3}$
N4SR-SCYS	16 / 5	45	$4.52 \cdot 10^{-3}$	$1.07 \cdot 10^{-2}$	$7.14 \cdot 10^{-3}$
N4SR-SENE	17 / 4	47	$1.21 \cdot 10^{-2}$	$3.30 \cdot 10^{-2}$	$2.89 \cdot 10^{-2}$
EQL^{\ddagger}	29 / 17	50	$5.27 \cdot 10^{-2}$	$4.87 \cdot 10^{-2}$	$5.26 \cdot 10^{-2}$
$mSNGP$ -LS	NA	50	$7.26 \cdot 10^{-4}$	$3.36 \cdot 10^{-3}$	$2.01 \cdot 10^{-3}$

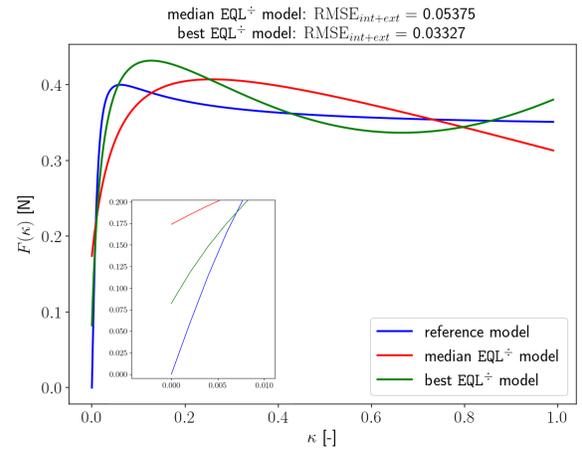
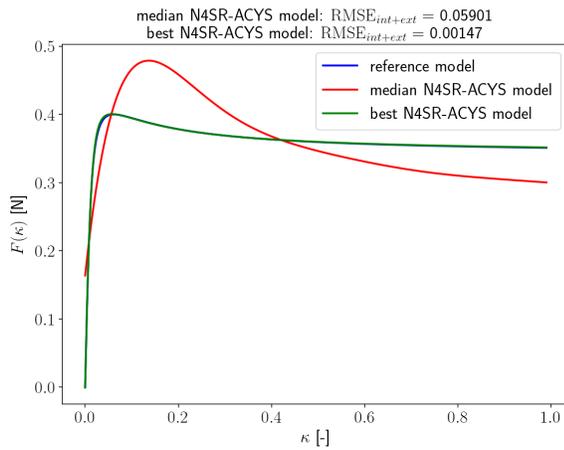
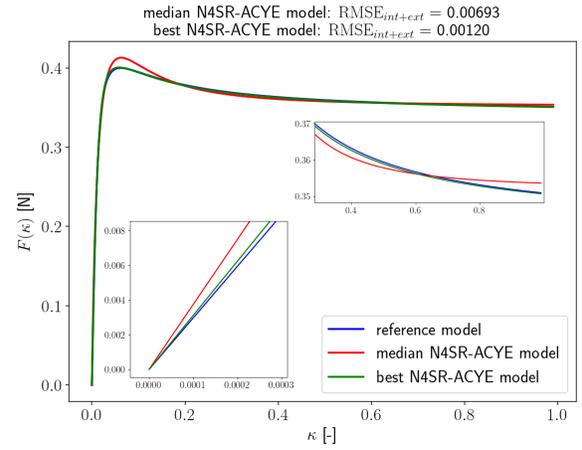
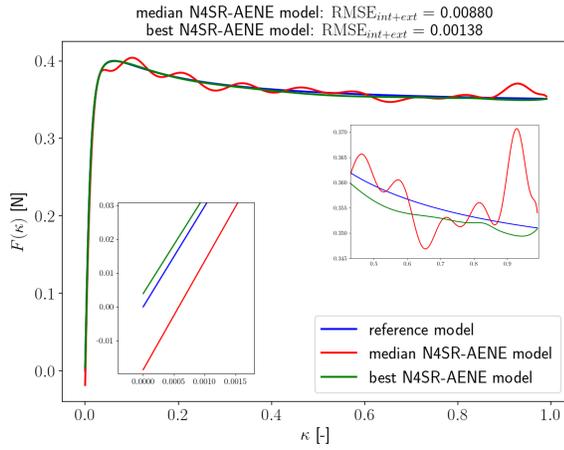


Fig. 9. Examples of models generated for the $MAGIC$ problem with N4SR-AENE (a), N4SR-ACYE (b), N4SR-ACYS (c), and EQL^{\ddagger} (d) method. Out of all best-of-run models collected over all runs for each experiment, the median model w.r.t. $RMSE_{int+ext}$ value (shown in red) and the model with the best $RMSE_{int+ext}$ value (shown in green) are presented.

that after simplification represents the expression $f(r_1, r_2) = \frac{c_1 r_1 r_2}{(c_2 r_1 + c_3 r_2)}$ where $c_1 \doteq c_2 \doteq c_3$. N4SR-ACYE failed to find this expression at all. Most of the time, expressions of the form $f(r_1, r_2) = \frac{c_1 r_1^2 + c_2 r_1 r_2 - c_3 r_2^2}{c_4 r_1 + c_5 r_2}$ with $c_1 \doteq c_3$, $c_2 \doteq c_4 \doteq c_5$, and $c_1 \ll c_2$ were found, which represent a close approximation of the reference model. This can also be caused by the top-down nature of the N4SR-ACYE approach. It may be that the sub-optimal model is more easily attainable since there are many more ways leading from the complete initial NN architecture to the sub-optimal model than to the optimal one.

Based on these analyses and observations, we hypothesize that when solving problems with the optimal solution composed of simple elementary structures, the mSNGP-LS method working in the bottom-up manner can much more efficiently arrive at the desired model than the N4SR-ACYE method.

VI. CONCLUSIONS AND FUTURE WORK

We proposed a new neural network-based symbolic regression method, N4SR, that generates models in the form of sparse analytic expressions. The novelty of this approach is that it allows the incorporation of prior knowledge describing desired properties of the system to be incorporated into the learning process. It also involves components to facilitate convergence to a sparse model and to eliminate the probability of getting stuck in some poor local optimum, namely the adaptive loss terms weighting scheme and the epoch-wise learning process. Also important is the proposed method for selecting the final model of the run.

We experimentally tested the approach on four test systems and compared it to another neural network-based algorithm and to a genetic programming-based algorithm. The results clearly demonstrate the potential of the proposed method to find sparse, accurate, and physically plausible models also in cases when only a very small training data set is given. Moreover, we demonstrated that the proposed parameter-less method for the final model selection is good at identifying models that perform well in the interpolation domain (i.e., the domain from which the training data were sampled) as well as in the extrapolation domain (i.e., the domain that was completely omitted in the training data).

We also identified weaknesses of the approach, particularly its inefficiency in pruning the initial NN architecture towards the sparse one representing the final simple analytic expression. We showed that on some problems, it tends to converge to suboptimal expressions due to its top-down search strategy.

In our future research, we will primarily investigate new strategies for better exploration of the space of sparse model architectures. We will focus on approaches that learn the most significant structures on the fly and use the information to guide the learning process. Other approaches will consider alternative representations of the NN units that would facilitate the pruning of the NN architecture towards a sparse final model.

REFERENCES

[1] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[2] G. Richmond-Navarro, W. R. Calderón-Muñoz, R. LeBoeuf, and P. Castillo, "A magnus wind turbine power model based on direct solutions using the blade element momentum theory and symbolic regression," *IEEE Transactions on Sustainable Energy*, vol. 8, no. 1, pp. 425–430, 2017.

[3] S.-M. Udrescu and M. Tegmark, "AI Feynman: A physics-inspired method for symbolic regression," *Science Advances*, vol. 6, p. eaay2631, 04 2020.

[4] E. Derner, J. Kubalík, N. Ancona, and R. Babuska, "Constructing parsimonious analytic models for dynamic systems via symbolic regression," *Appl. Soft Comput.*, vol. 94, p. 106432, 2020.

[5] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[6] N. Staelens, D. Deschrijver, E. Vladislavleva, B. Vermeulen, T. Dhaene, and P. Demeester, "Constructing a no-reference H.264/AVC bitstream-based video quality metric using genetic programming-based symbolic regression," *IEEE Trans. Cir. and Sys. for Video Technol.*, vol. 23, no. 8, p. 1322–1333, Aug. 2013.

[7] I. Arnaldo, U.-M. O'Reilly, and K. Veeramachaneni, "Building predictive models via feature synthesis," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 983–990.

[8] I. Bladek and K. Krawiec, "Solving symbolic regression problems with formal constraints," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '19. New York, NY, USA: ACM, 2019, pp. 977–984.

[9] A. Topchy, W. F. Punch *et al.*, "Faster genetic programming based on local gradient search of numeric leaf values," in *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)*, vol. 155162. Morgan Kaufmann San Francisco, CA, 2001.

[10] J. Žegklitz and P. Pošík, "Symbolic regression in dynamic scenarios with gradually changing targets," *Applied Soft Computing*, vol. 83, p. 105621, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1568494619304016>

[11] M. Kommenda, B. Burlacu, G. Kronberger, and M. Affenzeller, "Parameter identification for symbolic regression using nonlinear least squares," *Genetic Programming and Evolvable Machines*, vol. 21, no. 3, p. 471–501, sep 2020. [Online]. Available: <https://doi.org/10.1007/s10710-019-09371-3>

[12] E. Derner, J. Kubalík, and R. Babuska, "Selecting informative data samples for model learning through symbolic regression," *IEEE Access*, vol. 9, pp. 14 148–14 158, 2021. [Online]. Available: <https://doi.org/10.1109/ACCESS.2021.3052130>

[13] K. Krawiec, I. Bladek, and J. Swan, "Counterexample-driven genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '17. New York, NY, USA: ACM, 2017, pp. 953–960.

[14] J. Kubalík, E. Derner, and R. Babuška, "Symbolic regression driven by training data and prior knowledge," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, ser. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 958–966.

[15] J. Kubalík, E. Derner, and R. Babuska, "Multi-objective symbolic regression for physics-aware dynamic modeling," *Expert Syst. Appl.*, vol. 182, p. 115210, 2021. [Online]. Available: <https://doi.org/10.1016/j.eswa.2021.115210>

[16] L. Trujillo, L. Munoz, E. Galvan-Lopez, and S. Silva, "neat genetic programming: Controlling bloat naturally," *Information Sciences*, vol. 333, pp. 21–43, 10 Mar. 2016. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0020025515008038>

[17] G. Martius and C. H. Lampert, "Extrapolation and learning equations," *CoRR*, vol. abs/1610.02995, 2016. [Online]. Available: <http://arxiv.org/abs/1610.02995>

[18] S. S. Sahoo, C. H. Lampert, and G. Martius, "Learning equations for extrapolation and control," *CoRR*, vol. abs/1806.07259, 2018. [Online]. Available: <http://arxiv.org/abs/1806.07259>

[19] M. Werner, A. Junginger, P. Hennig, and G. Martius, "Informed equation learning," *CoRR*, vol. abs/2105.06331, 2021. [Online]. Available: <https://arxiv.org/abs/2105.06331>

[20] A. Costa, R. Dangovski, O. Dugan, S. Kim, P. Goyal, M. Soljačić, and J. Jacobson, "Fast neural models for symbolic regression at scale," 2021. [Online]. Available: <https://arxiv.org/abs/2007.10784>

[21] S. Kim, P. Y. Lu, S. Mukherjee, M. Gilbert, L. Jing, V. Ceperic, and M. Soljagic, "Integration of neural network-based symbolic regression in deep learning for scientific discovery," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 9, pp. 4166–4177, sep 2021.

- [22] H. Zhou and W. Pan, "Bayesian learning to discover mathematical operations in governing equations of dynamic systems," 2022. [Online]. Available: <https://arxiv.org/abs/2206.00669>
- [23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.
- [24] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.
- [25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.
- [26] M. Valipour, B. You, M. Panju, and A. Ghodsi, "SymbolicGPT: A generative transformer model for symbolic regression," *CoRR*, vol. abs/2106.14131, 2021. [Online]. Available: <https://arxiv.org/abs/2106.14131>
- [27] L. Biggio, T. Bendinelli, A. Neitz, A. Lucchi, and G. Parascandolo, "Neural symbolic regression that scales," *CoRR*, vol. abs/2106.06427, 2021. [Online]. Available: <https://arxiv.org/abs/2106.06427>
- [28] S. d'Ascoli, P. Kamienny, G. Lample, and F. Charton, "Deep symbolic regression for recurrent sequences," *CoRR*, vol. abs/2201.04600, 2022. [Online]. Available: <https://arxiv.org/abs/2201.04600>
- [29] M. Vastl, J. Kulhánek, J. Kubalík, E. Derner, and R. Babuška, "SymFormer: End-to-end symbolic regression using transformer-based architecture," 2022. [Online]. Available: <https://arxiv.org/abs/2205.15764>
- [30] Z. Hurak and J. Zemanek, "Feedback linearization approach to distributed feedback manipulation," in *American control conference*, Montreal, Canada, 2012, pp. 991–996.
- [31] J. Damsteeg, S. Nagesh Rao, and R. Babuška, "Model-based real-time control of a magnetic manipulator system," in *Proceedings 56th IEEE Conference on Decision and Control (CDC)*, Melbourne, Australia, Dec. 2017, pp. 3277–3282.
- [32] C. F. Verdier, R. Babuška, B. Shyrokau, and M. Mazo, "Near optimal control with reachability and safety guarantees," *IFAC-PapersOnLine*, vol. 52, no. 11, pp. 230–235, 2019, 5th IFAC Conference on Intelligent Control and Automation Sciences ICONS 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2405896319307761>
- [33] R. Gnadler, "Ermittlung von [my]-Schlupf-Kurven an Pkw-Reifen," Frankfurt [Main], 1995. [Online]. Available: [http://slubdd.de/katalog-TN_libero_mab2\)1000534083](http://slubdd.de/katalog-TN_libero_mab2)1000534083)