Learning Equations for Extrapolation and Control

Subham S. Sahoo [1] **Christoph H. Lampert** [2] **Georg Martius** [3]

Abstract

We present an approach to identify concise equations from data using a shallow neural network approach. In contrast to ordinary black-box regression, this approach allows understanding functional relations and generalizing them from observed data to unseen parts of the parameter space. We show how to extend the class of learnable equations for a recently proposed equation learning network to include divisions, and we improve the learning and model selection strategy to be useful for challenging real-world data. For systems governed by analytical expressions, our method can in many cases identify the true underlying equation and extrapolate to unseen domains. We demonstrate its effectiveness by experiments on a cart-pendulum system, where only 2 random rollouts are required to learn the forward dynamics and successfully achieve the swing-up task.

1. Introduction

In machine learning, models are typically treated as black-box function approximators that are only judged by their ability to predict correctly for unseen data (from the same distribution). In contrast, in the natural sciences, one searches for interpretable models that provide a deeper understanding of the system of interest and allow formulating hypotheses about unseen situations. The latter is only possible if the true underlying functional relationship behind the data has been identified. Therefore, when scientists construct models, they do not only minimize a training error but also impose constraints based on prior knowledge: models should be *plausible*, i.e. consist of components that have physical expressions in the real world, and they should be *interpretable*, which typically means that they consist only of a small number of interacting units.

[1]Indian Institute of Technology, Kharagpur, India [2]IST Austria, Klosterneuburg, Austria [3]Max Planck Institute for Intelligent Systems, Tübingen, Germany. Correspondence to: Georg Martius <georg.martius@tuebingen.mpg.de>.

Machine learning research has only very recently started to look into related techniques. As a first work, Martius & Lampert (2016) recently proposed EQL, a neural network architecture for identifying functional relations between observed inputs and outputs. Their networks represent only plausible functions through a specific choice of activation functions and it prefers simple over complex solutions by imposing sparsity regularization. However, EQL has two significant shortcomings: first, it is not able to represent divisions, thereby severely limiting to which physical systems it can be applied, and second, its model selection procedure is unreliable in identifying the true functional relation out of multiple plausible candidates.

In this paper, we propose an improved network for equation learning, EQL$\div$, that overcomes the limitation of the earlier works. In particular, our main contributions are

1. we propose a network architecture that can handle divisions as well as techniques to keep training stable,

2. we improve model/instance selection to be more effective in identifying the right network/equation,

3. we demonstrate how to reliably control a dynamical robotic system by learning its forward dynamics equations from very few random tryouts/tails.

The following section describes the equation learning method by Martius & Lampert (2016) and introduces our improvements. Afterwards, we discuss its relation to other prior work. In Section 4 we present experimental results on identifying equations and in Section 5 we show its application to robot control. We close with a discussion and outlook.

2. Identifying equation with a network

We consider a regression problem, where the data originates from a system that can be described by an (unknown) analytical function, $\phi : \mathbb{R}^n \to \mathbb{R}^m$. A typical example could be a system of ordinary differential equations that describes the dynamics of a robot, or the predator-prey equations of an ecosystem. The observed data, $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ is assumed to originate from $y = \phi(x) + \xi$ with additive zero-mean noise ξ. Since ϕ is unknown, we model the input-output relationship with a function $\psi : \mathbb{R}^n \to \mathbb{R}^m$ and aim to find an instance that minimizes the empirical error on
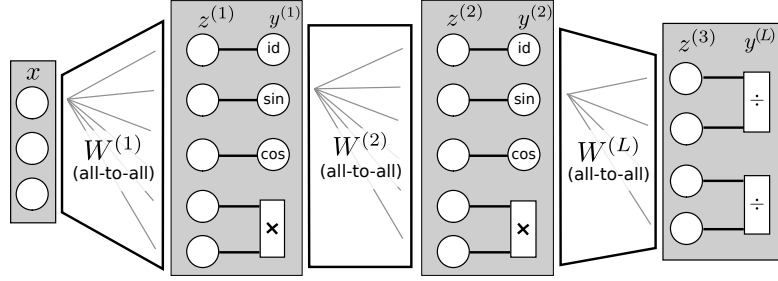
Figure 1. Network architecture of the proposed improved Equation Learner EQL$^{\div}$ for 3 layers ($L = 3$) and one neuron per type ($u = 3, v = 1$). The new division operations are places in the final layer, see Martius & Lampert (2016) for the original model.

the training set as well as on future data, potentially from a different part of the feature space. For example, we might want to learn the robot dynamics only in a part of the feature space where we know it is safe to operate, while later it should be possible also to make predictions for movements into unvisited areas.

2.1. Equation Learner

Before introducing EQL$^{\div}$, we first recapitulate the working principles of the previously proposed Equation Learner (EQL) network. It uses a multi-layer feed-forward network with units representing the building blocks of algebraic expressions. Instead of homogeneous hidden units, each unit has a specific function, e. g. identity, cosine or multiplication, see Fig. 1. Complex functions are implemented by alternating linear transformations, $z^{(l)} = W^{(l)}y^{(l-1)} + w_o^{(l)}$ in layer l, with the application of the base-functions. There are u unary functions $f_1, \ldots, f_u$, $f_i \in \{\text{identity}, \sin, \cos\}$, which receive the respective component, $z_1, \ldots, z_u$. The v binary functions, $g_1, \ldots, g_v$ receive the remaining component, $z_{u+1}, \ldots, z_{u+2v}$, as input in pairs of two. In EQL these are *multiplication units* that compute the product of their two input values: $g_j(a, b) := a \cdot b$. The output of the unary and binary units are concatenated to form the output $y^{(l)}$ of layer l. The last layer computes the regression values by a linear read-out

$$y^{(L)} := W^{(L)}y^{(L-1)} + w_o^{(L)}. \tag{1}$$

For a more detailed discussion of the architecture, see (Martius & Lampert, 2016).

2.2. Introducing division units

The EQL architecture has some immediate shortcomings. In particular, it cannot model divisions, which are, however, common in the equations governing physical systems. We, therefore, propose a new architecture, EQL$^{\div}$, that includes *division units*, which calculate a/b. Note that this is a non-trivial step because any division creates a pole at $b \to 0$ with an abrupt change in convexity and diverging function value and its derivative. Such a divergence is a serious problem

for gradient based optimization methods.

To overcome the divergence problem, we first notice that from any real system we cannot encounter data at the pole itself because natural quantities do not diverge. This implies that a single branch of the hyperbola $1/b$ with $b > 0$ suffices as a basis function. As a further simplification we use divisions only in the ouput layer.

Finally, in order to prevent problems during optimization we introduce a curriculum approach for optimization, progressing from a strongly regularized version of division to the unregularized one.

Regularized Division: Instead of EQL's Eq. (1), the last layer of the EQL$^{\div}$ is

$$y^{(L)} := \left(h_1^\theta(z_1^{(L)}, z_2^{(L)}), \ldots, h_m^\theta(z_{2m}^{(L)}, z_{2m+1}^{(L)}) \right), \tag{2}$$

where $h^\theta(a, b)$ is the division-activation function given by

$$h^\theta(a, b) := \begin{cases} \frac{a}{b} & \text{if } b > \theta \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where $\theta \geq 0$ is a threshold, see Fig. 2. Note that using $h^\theta = 0$ as the value when the denominator is below θ (forbidden values of b) sets the gradient to zero, avoiding misleading parameter updates. So the discontinuity plays no role in practice.

Penalty term: To steer the network away from negative values of the denominator, we add a cost term to our objective that penalizes "forbidden" inputs to each division unit:

$$p^\theta(b) := \max(\theta - b, 0), \tag{4}$$

where θ is the threshold used in Eq. (3) and b is the denominator, see Fig. 2. The global penalty term is then

$$P^\theta = \sum_{i=1}^N \sum_{j=1}^n p^\theta(z_{2j}^{(L)}(x_i)) \tag{5}$$

where $z_{2j}^{(L)}(x_i)$ is the denominator of division unit j for input x_i, see Eq. (2).
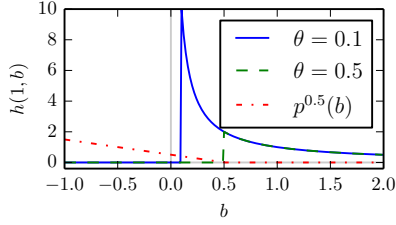
Figure 2. Regularized division function $h^\theta(a, b)$ and the associated penalty term $p^\theta(b)$. The penalty is linearly increasing for function values $b < \theta$ outside the desired input values.

Penalty Epochs: While Eq. (5) prevents negative values in the denominator at training time, the right equation should not have negative denominators even for potential extrapolation data. Similarly, we would like to prevent that output values on future data have a very different magnitude than the observed outputs, as this could be a sign of overfitting (e. g. learning a polynomial of too high-degree). To enforce this we introduce particular *penalty epochs*, which are injected at regular intervals (every 50 epochs) into the training process.

During a penalty epoch we randomly sample N input data points in the expected test range (including extrapolation region) without labels and the network is trained using the cost $\mathcal{L}^{\text{Penalty}} = P^\theta + P^{\text{bound}}$, where the latter is given by:

$$P^{\text{bound}} := \sum_{i=1}^{N} \sum_{j=1}^{n} \max(y_j^{(L)}(x_i) - B, 0) \quad (6)$$
$$+ \max(-y_j^{(L)}(x_i) - B, 0).$$

The value B reflects the maximal desired output value. It is, of course, problem dependent but can easily be estimated from the observed training data. The system is insensitive to the exact choice. In our experiments, expected outputs are $[-3, 3]$ and we use $B = 10$.

Curriculum: The threshold θ in the division function Eq. (3) plays the role of avoiding overly large gradients. However, ultimately we want to learn the precise equation so we introduce a curriculum during training in which regularization is reduced continuously. More precisely, θ decreases with epoch t as

$$\theta(t) = 1/\sqrt{t+1}. \quad (7)$$

For validation and testing, we use $\theta = 10^{-4}$.

2.3. Network training

EQL$^{\div}$ is fully differentiable in its free parameters $\Theta = \{W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)}\}$, which allows us to train it in an end-to-end fashion using back-propagation. The

objective is Lasso-like (Tibshirani, 1996),

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \|\psi(x_i) - y_i\|^2 + \lambda \sum_{l=1}^{L} \left|W^{(l)}\right|_1 + P^\theta \quad (8)$$

that is, a linear combination of L_2 loss and L_1 regularization extended by the penalty term for small and negative denominators, see Eq. (4). Note that P^{bound} (Eq. 6) is only used in the penalty epochs. For training, we apply a stochastic gradient descent algorithm with mini-batches and Adam (Kingma & Ba, 2015) for calculating the updates. The choice of Adam is not critical, as we observed that standard stochastic gradient descent also works, though it might take longer.

Regularization Phases: We follow the same regularization scheme as proposed in Martius & Lampert (2016). The role of the regularization is to create a network with sparse connections, corresponding to the few terms appearing in a typical formula describing a physical system. Ideally, we would like to minimize the L_0 norm directly, but this would make the loss not differentiable. As commonly done, we use the L_1 norm as a substitute. However, as detailed below, we add a step for emulating a trade-off free L_0 minimization.

The regularization scheme is as follows: we start with an un-regularized phase ($\lambda = 0$) because starting with L_1 regularization sometimes causes weights not to change sign during training but getting "trapped" at zero. We continue by a phase, where regularization is normally enabled by setting λ to a nonzero value, which has the effect that a sparse network structure emerges. Such a phasing was also suggest by Carpenter (2008). As a side remark, subgradient methods or clipping (Schmidt et al., 2009; Carpenter, 2008) was not required to deal with the L_1 regularizer because potentially slight weight fluctuations around zero are not important due to the next phase which is motivated by the following fact. A non-zero L_1 regularization term leads to a trade-off between minimizing the loss and the regularizer. Since our aim is to learn the right functional form with the correct coefficients, we add a third phase where we disable L_1 regularization ($\lambda = 0$) but enforce the same L_0 norm of the weights. This is achieved by keeping all weights $w \in W^{1\ldots L}$ that are close to 0 at 0, i. e. if $|w| < 0.001$ then $w = 0$ during the remaining epochs. In this way, the model complexity is fixed and we ensure that function values fit the observed values as closely as possible, allowing for a correct parameter estimation. It also eliminates any potential fluctuations of small weights that might occur during the L_1 phase. The effect of the regularization phases is schematically illustrated in Fig. 3.

We use $t_1 = \frac{1}{4}T$ and $t_2 = \frac{19}{20}T$, where T is the total number of epochs, large enough to ensure convergence, i. e. $T = (L-1) \cdot 10000$. Note, that early stopping will be disadvantageous.
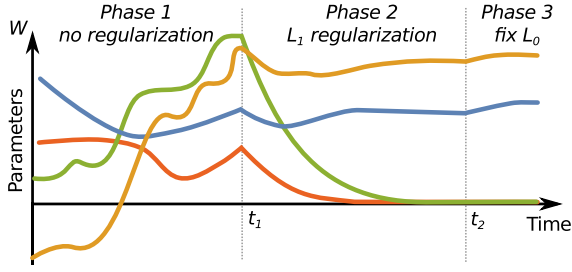
Figure 3. Regularization phases: there is no regularization in the first phase ($t < t_1$) where the weights can move freely, followed by a normal L_1 phase ($t_1 \leq t < t_2$) where many weights go to zero, followed by a phase ($t_2 \leq t$) that fixes the L_0 norm by keeping small weights at zero and allowing all other weights to go to their correct value. Figure adapted from Martius & Lampert (2016).

2.4. Model selection for extrapolation

The model selection is a critical component of the architecture. Only if the "right" formula is selected good extrapolation capabilities can be expected. A set of different hypothesis equations can be obtained by choosing a range of hyperparameters in particular for the regularization strength.

As suggested in Martius & Lampert (2016), the "right" formula is singled out by being the simplest one that still predicts well, according to the Occam's razor principle. We found that this is in principle correct, but for complicated cases we were not able to select the right network instance. In this paper, we distinguish between two cases, where we have access to a few labeled points in the extrapolation domain and where we do not.

Without extrapolation data, the selection process has to be based on the validation and sparsity of the instance. Sparsity is measured in terms of the number of connected units and is denoted by s, where a smaller value means simpler/sparser. In Martius & Lampert (2016), the best model was selected based on the distance in the space of ranked validation error and ranked sparsity. We found that this method is often disadvantageous because it is insensitive to numeric differences in validation error and practically identically performing instances get different ranks. In this paper, we instead propose to normalize the quantities instead. The criterion for the best model is:

$$\arg\min_{\psi} \left[\alpha \tilde{v}^{\text{int}}(\psi)^2 + \beta \tilde{s}(\psi)^2 \right], \qquad (9)$$

where ψ stands for an instance (trained network), $\tilde{v}^{\text{int}}(\psi)$ and $\tilde{s}(\psi)$ are the validation error and sparsity of network ψ normalized to $[0, 1]$ w.r.t. over all instances. The weighting factors $\alpha = 0.5$ and $\beta = 0.5$ are empirically determined, see Fig. 4. We call this the $\mathbf{V^{int}}$-$\mathbf{S}$ selection method, because it relies on interpolation-validation and sparsity.
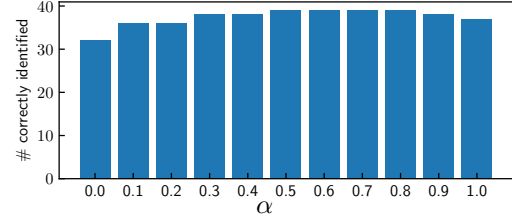

Figure 4. Model selection: trade-off between sparsity and validation error, see Eq. (9) with $\beta = 1 - \alpha$, evaluated empirically in terms of the number of correctly identified equations on all the datasets considered in Section 4. We choose the canonical $\alpha = 0.5$ as the differences between 0.3 and 0.9 are not significant.

With few extrapolation points, we form an additional extrapolation-validation dataset and denote the validation error as v^{ex}. In the same manner as above we can select now based on

$$\arg\min_{\psi} \left[\alpha \tilde{v}^{\text{int}}(\psi)^2 + \beta \tilde{s}(\psi)^2 + \gamma \tilde{v}^{\text{ex}}(\psi)^2 \right]. \qquad (10)$$

A grid search on the weighting factors α, β, γ on all datasets revealed that the sparsity is counterproductive when having some points (40 in our case) from the extrapolation domain. Thus we introduce the $\mathbf{V^{int\&ex}}$ selection method using $\alpha = 0.5, \beta = 0, \gamma = 0.5$, i.e. same weight on interpolation and extrapolation validation error. Note that using the normalized values is important because both error terms might be of different scale.

It might be surprising that the sparsity term loses its importance. We speculate that simply counting nodes can be a misleading measure of simplicity. When instead looking at the extrapolation-validation error, the network can use trigonometric identities with a larger number of terms but which are easier to find.

How many points do we need from the extrapolation domain? Since the extrapolation errors vary largely between correct and incorrect formulas, only relatively few points are needed. Empirically, around 40 points were sufficient to identify the right instance from over a hundred candidates.

Because of the strong non-convexity of the problem, the optimization process may get stuck in a local minimum or not select the correct formula. Therefore, to quantify the expected performance deviations, we use 10 independent runs with random initialization conditions.

3. Relation to prior work

In this work, we are departing from the classical path in machine learning of finding any function that yields a small expected error on future data of the same distribution as the training data. Instead, we aim at discovering the underlying relationship between input and output, much like it is done in the natural sciences. In this way an interpretable function

with concise form is obtained.

In machine learning, this task has received little attention but is studied in the field of *system identification*. The methods from system identification assume that the functional form of the system is known and only the parameters have to be identified. Recently, this was shown to be effective for identifiying partial differential equations in a range of systems (Rudy et al., 2017). In our work, we also learn the parameters of the base functions and, most importantly, their composition.

The task of finding equations for observations is also known as symbolic regression. For a given function space, a heuristic search is performed, typically with evolutionary computation. With these techniques, it is possible to discover physical laws such as invariants and conserved quantities (Schmidt & Lipson, 2009). Unfortunately, due to the exponential search space, the computational complexity becomes prohibitive for larger expressions and high-dimensional problems. We attempt to circumvent this by modeling it as a gradient-based optimization problem. Related to symbolic regression is finding mathematical identities for instance to find computationally more efficient expressions. In (Zaremba et al., 2014), this was done using machine learning to overcome the potentially exponential search space.

Another relation to our work is in *causal learning*, which aims at identifying a causal relationship between multiple observables, originating from some underlying functional mechanism. This was pioneered by Pearl (2000) who reduces this to finding a minimal graphical model based only on tests of conditional independence. Although it provides a factorization of the problem and reveals causes and effects, it leaves the exact functional dependency unexplained. In order to reveal causal relationships from fewer observables (e. g. just two), in Peters et al. (2014) a functional view was taken. However, the causal inference is based on the expected noise distribution instead of the simplicity/plausibility of regression functions.

Our work has some similarity to domain adaptation, because data from the extrapolation domain differs from the data distribution at training time. As we assume a shared labeling function between both, our situation fits the covariate shift (Quionero-Candela et al., 2009) setting. However, existing domain adaptation techniques, such as sample reweighting (Sugiyama et al., 2007) are not applicable, because training and extrapolation domain are disjoint for us. Where existing approaches rely on an assumption of distribution similarity, we instead make use of the fact that the target function has an analytic, and therefore global characterization. For the same reasons, existing theoretic results, such as Ben-David et al. (2010) are not applicable (or vacuous) in our setting.

For a discussion on the architectural relation to prior work we refer to Martius & Lampert (2016). To summarize, the individual components, such as product units or sine units were introduced before. However, the combination as introduced here and the restriction to pairwise multiplication terms are new. To the best of our knowledge, we are the first to use division units in a neural network.

4. Experimental evaluation

We first demonstrate the ability of EQL$^{\div}$ to learn physically inspired models with and without divisions with good extrapolation quality. Technically, we implemented the network training and evaluation procedure in *python* based on the *theano* framework (Theano Development Team, 2016). The code and some data is available at `https://github.com/martius-lab/EQL`.

For all experiments, we have training data in a restricted domain, usually $[-1, 1]^d$ corrupted with noise which is split into training and validation with $90\% - 10\%$ split. For testing, we have two sets, one from the training domain and one from an extrapolation domain, for instance $[-2, 2]^n \setminus [-1, 1]^n$. The error is measured in root mean squares error (RMS) $\sqrt{\frac{1}{N} \sum_i \|\psi(x_i) - y_i\|^2}$. The following hyperparameters were fixed: learningrate (Adam) $\alpha = 0.001$, regularization (Adam) of $\epsilon = 0.0001$, minibatch size of 20, number of units $\frac{1}{3}u = v = 10$, i. e. 10 units per type in each layer.

4.1. Learning formulas with divisions

We start with a small experiment to check whether simple formulas with divisions can be learned. We sample data from the following formula:

$$y = \frac{\sin(\pi x_1)}{(x_2^2 + 1)} \tag{11}$$

As training data, we sample 10000 points uniformly in the hypercube $[-1, 1]^2$ and add noise ($\sim \mathcal{N}(0, 0.01)$). The *extrapolation test set* contains 5000 uniformly sampled points from the data domain $[-2, 2]^2 \setminus [-1, 1]^2$. Note, that the extrapolation domain is 3 times larger than the training domain. We perform model selection among the following hyper-parameters: the regularization strength $\lambda \in 10^{\{-6, -5.9, \dots, -3.6, -3.5\}}$, and $L \in \{2, 3, 4\}$. We use the same parameters for all experiments. We compare our algorithm to a standard multilayer perceptron (MLP) with $\tanh$ activation functions and possible hyperparameters: $\lambda \in 10^{\{-6.3, -6.0, -5.3, -5.0, -4.3, -4.0\}}$, number of layers $L \in \{2, 3, 4\}$, and number of neurons $k \in \{5, 10, 20\}$. A second baseline is given by epsilon support vector regression (SVR) (Basak et al., 2007) with two hyperparameters $C \in 10^{\{-3, -2, -1, 0, 1, 2, 3, 3.5\}}$ and $\epsilon \in 10^{\{-3, -2, -1, 0\}}$ using radial
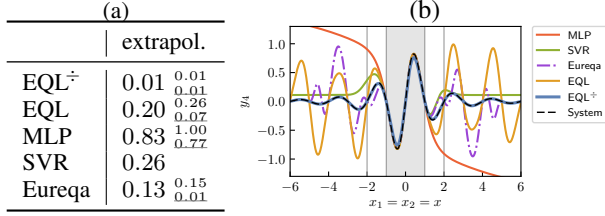
Figure 5. Results on the *division* task, Eq. (11). (a) Reported are median, minimum and maximum (in sub and superscript) of the root mean squares error (RMS) for 10 random initializations. (b) Output y for a slice of the inputs $x_1 = x_2 = x$ for the true system equation (Eq. 11), and best model-selected instances (for the run with the median performance). Note that we plot a much larger domain than that of the extrapolation test-set.

basis function kernel with width $\gamma \in \{0.05, 0.1, 0.2, 0.5, 1.0\}$. We also compare to Eureqa (Nutonian, 2018), a symbolic regression algorithm using evolutionary search introduced in (Schmidt & Lipson, 2009). The termination condition was when the software reported 100% convergence and no better solution was found in the last 10% of the search time.

In Fig. 5 the numerical results and also an illustrative output of EQL$^{\div}$ and the baselines are presented. Only EQL$^{\div}$ can extrapolate to the unseen domain because it has the capacity to identify the underlying expression, which was achieved in all of the 10 independent runs. Note, that the original EQL method cannot extrapolate because it lacks the division units. Interestingly, Eureqa is not finding the right equations in $4/5$ cases. The interpolation performances are at the noise level of 0.01 for all methods (not shown).

4.2. Learning complex formulas.

Following Martius & Lampert (2016) we test EQL$^{\div}$ on a set of complicated formulas and compare it to the original EQL, MLP, SVR and Eureqa as baselines. These do not contain divisions and we want to test whether this poses any problems as our architecture always contains divisions. We consider the following formulas with four-dimensional input and one-dimensional output:

$$y = \tfrac{1}{3}\left(\sin(\pi x_1) + \sin\left(2\pi x_2 + \tfrac{\pi}{8}\right) + x_2 - x_3 x_4\right) \quad \text{(F-1)}$$

$$y = \tfrac{1}{3}\left(\sin(\pi x_1) + x_2\cos\left(2\pi x_1 + \tfrac{\pi}{4}\right) + x_3 - x_4^2\right) \quad \text{(F-2)}$$

$$y = \tfrac{1}{3}\left((1 + x_2)\sin(\pi x_1) + x_2 x_3 x_4\right) \quad \text{(F-3)}$$

$$y = \tfrac{1}{2}\left(\sin(\pi x_1) + \cos(2 x_2 \sin(\pi x_1)) + x_2 x_3 x_4\right) \quad \text{(F-4)}$$

For a correct identification, the equations F-1 requires one hidden layer, F-2 and F-3 require two, and F-4 requires three ($L = 4$). Nevertheless, the right number of layers is automatically detected by model selection. Data generation and training procedure is the same as for the Section 4.1, except that the data is now from a 4-dimensional hypercube, which makes the extrapolation domain 15 times larger than the training domain.
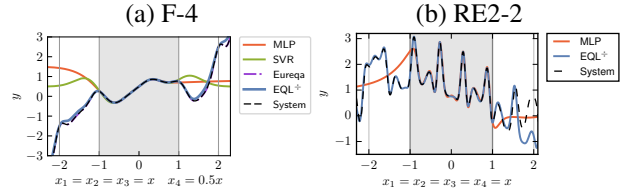

Figure 6. Formula learning: (a) for F-4, (b) for RE3-4. Shown is y for a single cut through the input space for the true system equation and for the best model-selected instances.

dataset	method	extrapolation error	
		$V^{\text{int}}\text{-S}/V^{\text{int}}$	$V^{\text{int\&ex}}$
F-2	EQL$^{\div}$	$0.01\ ^{0.01}_{0.01}$	$0.01\ ^{0.01}_{0.01}$
	EQL	$0.03\ ^{0.06}_{0.01}$	—
	MLP	$0.58\ ^{0.64}_{0.50}$	$0.46\ ^{0.50}_{0.43}$
	SVR	0.91	0.45
	Eureqa	$0.01\ ^{0.01}_{0.01}$	$0.01\ ^{0.01}_{0.01}$
F-3	EQL$^{\div}$	$0.01\ ^{0.01}_{0.01}$	$0.01\ ^{0.01}_{0.01}$
	EQL	$0.35\ ^{0.51}_{0.15}$	—
	MLP	$0.47\ ^{0.51}_{0.43}$	$0.46\ ^{0.50}_{0.39}$
	SVR	0.34	0.39
	Eureqa	$0.01\ ^{0.33}_{0.01}$	$0.01\ ^{0.46}_{0.01}$
F-4	EQL$^{\div}$	$0.23\ ^{1.04}_{0.07}$	$0.21\ ^{0.71}_{0.07}$
	EQL	$0.37\ ^{0.46}_{0.25}$	—
	MLP	$0.86\ ^{0.95}_{0.72}$	$0.86\ ^{0.95}_{0.72}$
	SVR	0.91	1.28
	Eureqa	$0.85\ ^{0.95}_{0.54}$	$0.18\ ^{1.10}_{0.01}$

Table 1. Extrapolation performance and model selection for *formula learning*. See Fig. 5 for details. For EQL the original model selection was used. For F-1 (not shown), both EQL and EQL$^{\div}$, have an error of 0.01.

Table 1 shows the numerical results. All methods are able to interpolate, but only EQL$^{\div}$ and Eureqa achieve good extrapolation results in all cases. The original EQL (with the original model selection) was able to find the right answer only for F-2. With the model selection $V^{\text{int\&ex}}$ (using 40 extrapolation points), EQL$^{\div}$ and Eureqa are able to find the right formula for F-3 every time and for F-4 in about half of the cases (manual inspection of the learned equations).

Figure 6(a) illustrates the performance of the learned models for F-4 visually. It shows the output for a slice through the input for one of the model-selected instances for each method. It is remarkable how well the extrapolation works for EQL$^{\div}$ and Eureqa even far outside training region.

4.3. Random expressions

In order to avoid a bias through hand-crafted formulas, we generated random functional expressions. The expressions were generated with our architecture with 2 and 3 hidden

layers and random sparse connections, 4 instances each, named as RE$\{2,3\}$-$\{1,2,3,4\}$. The weights and biases were sampled from a uniform distribution in $[0.5, 2]$ and were multiplied with $[1, -1]$ with equal probability. Further, the input weights into the sin and cos nodes were multiplied by π, to avoid including only the linear regime of the trigonometric functions. Some of the expressions are very hard to learn, even in the interpolation region. Table 2 shows the experimental results. We also compare the two model-selection strategies (V^{int}-S and $V^{\text{int\&ex}}$). It becomes evident that without a few points from the extrapolation region (here 40) the system is not able to identify the right formula in the majority of cases. For the baselines (MLP, SVR) the model selection based on extrapolation only reduced the strong outliers but did not yield acceptable performance. This is expected because these methods have no chance to identify the right functional relationship. Figure 6(b) illustrates the complicated structure at the example of RE3-4. Again, note that the extrapolation domain is 15 times larger than the training domain.

4.4. Cart-pendulum system

Let us now consider a non-trivial physical system: a pendulum attached to a cart that can move horizontally along a rail but that is attached to a spring damper system, see Fig. 7(a). The system is parametrized by 4 unknowns: the position of the cart, the velocity of the cart, the angle of the pendulum and the angular velocity of the pendulum. We combine these into a four-dimensional vector $x = (x_1, \dots, x_4)$. We set up a regression problem with four outputs from the corresponding system of ordinary differential equations where $y_1 = \dot{x}_1 = x_3$, $y_2 = \dot{x}_2 = x_4$ and

$$y_3 = \frac{-x_1 - 0.01x_3 + x_4^2 \sin(x_2) + 0.1x_4 \cos(x_2) + 9.81 \sin(x_2) \cos(x_2)}{\sin^2(x_2) + 1}$$

(12)

$$y_4 = \frac{-0.2x_4 - 19.62 \sin(x_2) + x_1 \cos(x_2) + 0.01x_3 \cos(x_2) - x_4^2 \sin(x_2)\cos(x_2)}{\sin^2(x_2) + 1}.$$

The task is to learn the function without controlling the system. The formulas contain divisions which are now included in the EQL$^{\div}$ architecture. In Fig. 7(b) the extrapolation performance is illustrated by slicing through the input space. Near the training region all methods fit the data well, but a bit further away only EQL$^{\div}$ can predict well. For all other methods, even the best instances differ considerably from the true values, see also the numerical results in Tab. 3. In 1 out of the 10 independent runs also EQL$^{\div}$ performed poorly. This is less likely for a finer scan of λ values.

5. Control using learned dynamics

In this section, we will demonstrate the effectiveness of the equation learning for robot control. For the cart-pole we know that the system can learn correctly the differential
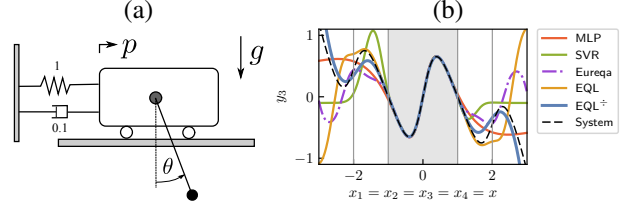

Figure 7. Cart-pendulum system. (a) sketch of the system. The lengths and masses are set to 1, the gravitation constant is 9.81 and the friction constant is 0.01. (b) slices of outputs y_3 for inputs $x_1 = x_2 = x_3 = x_4 = x$ for the true system equation (Eq. 12), and learned EQL, EQL$^{\div}$, and MLP instances.

equations from randomly sampled data, see above. Now, we are challenging the system to learn the dynamics from actual interactions and subsequently use it to control the robot, namely to learn how to perform the swing-up task with the cart-pole. We use the OpenAI Gym cart-pole environment (OpenAI Gym, 2018) that we modified for the swing-up task, i.e. to start at the bottom. The state of the system $s = (x, \dot{x}, \theta, \dot{\theta})$ contains the position and velocity of the cart and the angle and angular velocity of the pole. The learner should model the forward dynamics $f(s, a) \to \dot{s}$, predicting state changes from the current state and action.

At the beginning no knowledge about the system is available, such that we perform $K - 1$ random rollouts with 1000 steps (20secs) with random actions $a \sim \mathcal{N}(0, 0.15)$. The resulting pole angle distribution had mean π and standard deviation of approximately $\pi/4$, so only a small part of the angle range was visited. In order to obtain data for model selection we perform one additional rollout with $a \sim \mathcal{N}(0, 0.25)$.

After training we use the resulting models for optimal control. We define a cost which defines the desired position, i.e. vertical upright pole with small velocities and cart at the center:

$$C = 0.1x^2 + 0.1\dot{x}^2 - \cos(\theta) + 0.02\dot{\theta}^2. \quad (13)$$

Note, that the stability point lies far outside the training domain. To achieve the optimal control we use model predictive control (MPC) (García et al., 1989) with a random shooting method. Briefly, every timestep 1000 lookaheads are simulated using the learned dynamics (Euler integration) with random actions. After a fixed horizon the look-ahead with the lowest cost C is chosen and only the first action is applied. This runs in realtime for $100\,\text{Hz}$ update rate. For MPC to work the model should provide with a good enough representation of the system dynamics.

In Fig. 8 the results for different number of rollouts K are presented. The reward from the environment is $R = \sum_t \cos(\theta_t)$. We report statistics over 5 independent experiments each. With EQL$^{\div}$ the cart-pole is able to accomplish

Table 2. Extrapolation performance for *random graphs*. See Fig. 5 for details. Results for different methods and model selection schemes. Const 0 refers to a constant prediction of zero. For some random expressions marked with ✗(RE2-3 and R3-1) we are not able to learn them with satisfactory precision. A visualization of RE3-4 can be found in Fig. 6(b).

	RE2-1	RE2-2	RE2-3 ✗	RE2-4	RE3-1 ✗	RE3-2	RE3-3	RE3-4
EQL$^{\div}$ V$^{int\&ex}$	$0.02\ _{0.02}^{0.02}$	$0.04\ _{0.03}^{0.11}$	$0.52\ _{0.48}^{0.82}$	$0.01\ _{0.01}^{0.01}$	$0.46\ _{0.28}^{0.55}$	$0.02\ _{0.01}^{0.06}$	$0.01\ _{0.01}^{0.01}$	$0.03\ _{0.02}^{0.52}$
EQL$^{\div}$ V^{int}-S	$0.27\ _{0.02}^{0.39}$	$0.14\ _{0.14}^{0.14}$	$0.76\ _{0.55}^{2.05}$	$0.01\ _{0.01}^{0.01}$	$0.51\ _{0.31}^{1.23}$	$0.08\ _{0.04}^{4.65}$	$0.01\ _{0.01}^{0.01}$	$0.03\ _{0.01}^{1.64}$
MLP V$^{int\&ex}$	$1.54\ _{1.43}^{1.66}$	$1.04\ _{0.96}^{1.09}$	$0.90\ _{0.91}^{0.91}$	$0.95\ _{0.87}^{1.12}$	$1.04\ _{0.84}^{1.36}$	$1.85\ _{1.60}^{2.13}$	$0.52\ _{0.40}^{0.58}$	$1.64\ _{1.34}^{1.96}$
MLP V^{int}	$1.60\ _{1.44}^{1.66}$	$1.05\ _{1.01}^{1.10}$	$1.47\ _{1.10}^{1.65}$	$0.99\ _{0.86}^{1.16}$	$1.31\ _{1.07}^{1.59}$	$2.03\ _{1.65}^{2.24}$	$1.16\ _{0.73}^{2.02}$	$1.89\ _{1.61}^{2.12}$
SVR V$^{int\&ex}$	1.15	1.09	0.59	1.51	0.96	1.81	0.37	1.23
SVR V^{int}	1.20	2.12	17.72	13.89	11.79	11.28	0.37	17.67
Const 0	6.73	2.57	0.50	5.36	1.65	72.26	17.67	3.15

Table 3. Interpolation and extrapolation performance for *cart-pendulum dynamics*. See Fig. 5 for details. V$^{int\&ex}$ model selection is used. Note that predicting 0 would yield an error of 0.96.

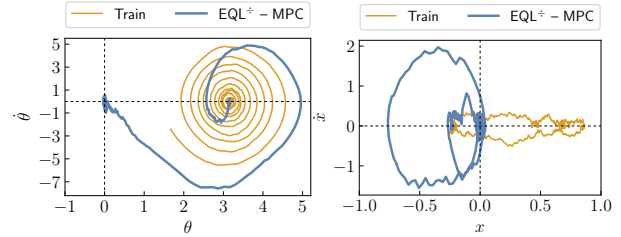|	interpolation	extrapolation
EQL$^{\div}$	$0.010\ _{0.010}^{0.010}$	$0.06\ _{0.04}^{4.20}$
EQL	$0.010\ _{0.010}^{0.011}$	$0.17\ _{0.16}^{0.21}$
MLP	$0.012\ _{0.012}^{0.012}$	$0.18\ _{0.18}^{0.19}$
SVR	0.019	0.36
Eureqa	$0.012\ _{0.011}^{0.012}$	$0.19\ _{0.15}^{17.2}$

Figure 9. Training trajectory and controlled trajectory (EQL$^{\div}$) of cart-pole system in angle-space left and cart positions right, $K = 2$. See also the Video at https://youtu.be/MG9q3gTtBLs.
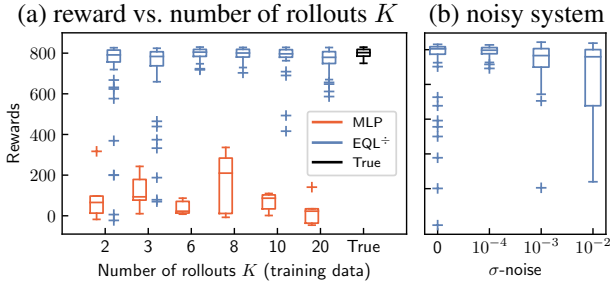

Figure 8. Performance on the cart-pole swing-up task with learned dynamics. (a) reward during test-runs for models trained on K rollouts (5 independent experiments with 10 test runs each). (b) robustness to noise in sensors and actions for $K = 3$.

the task already after $K = 2$ rollouts (one for training, one for validation) in most of the cases. For $K = 3$ we also show its robustness to noise on the states and actions. In comparison the neural network forward model (MLP) does only perform swing around operation but was nowhere close to stabilizing it in the vertical position.

In Fig. 9 a sample training trajectory and a final control trajectory is displayed for $K = 2$.

6. Conclusions

In this paper, we introduce EQL$^{\div}$, a new network architecture for equation learning. It improves over previously suggested work by including the ability to learn di-

vision, thereby substantially increasing the applicability. The network is fully differentiable and trainable with back-propagation, which we achieve by introducing a regularized version of division that is smoothly transformed towards the true division in a curriculum fashion. We also introduce a new model selection technique that identifies the actual functional relation between inputs and outputs more reliably than the purely sparsity-based approach of previous work.

We empirically demonstrate that EQL$^{\div}$ can learn various functional relations, with and without divisions, from noisy data in a confined domain. Furthermore, those can extrapolate to unseen parts of the data space. In a broad set of experiments we show that the approach learns concise functional forms that can be inspected and may provide insights into the relationships within the data.

This ability opens doors for many new applications. As an exemplary task, we show efficient model learning for robot control: EQL$^{\div}$ identifies the forward dynamics for an actuated cart-pendulum system from just 2 random rollouts with limited excitation and under noisy observations. The resulting forward model is good enough to robustly perform a pendulum swing-up task, despite the fact that such a pattern was never observed during training. Also embedding EQL$^{\div}$ as part of deep architecture is conceivable. In future studies, we will consider larger and more complex systems, where we expect good scaling capabilities due to the gradient-based optimization.

References

Basak, D., Pal, S., and Patranabis, D. C. Support vector regression. *Neural Information Processing-Letters and Reviews*, 11(10):203–224, 2007.

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. A theory of learning from different domains. *Machine Learning*, 79(1-2):151–175, 2010.

Carpenter, B. Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. *Alias-i, Inc., Tech. Rep*, pp. 1–20, 2008.

García, C. E., Prett, D. M., and Morari, M. Model predictive control: Theory and practice—a survey. *Automatica*, 25 (3):335 – 348, 1989. ISSN 0005-1098. doi: 10.1016/0005-1098(89)90002-2.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of ICLR*, 2015.

Martius, G. and Lampert, C. H. Extrapolation and learning equations, 2016. arXiv preprint https://arxiv.org/abs/1610.02995.

Nutonian. Eureqa desktop software. https://www.nutonian.com/products/eureqa-desktop, 2018.

OpenAI Gym. Cartpole-v0. https://gym.openai.com/envs/CartPole-v0, 2018.

Pearl, J. *Causality*. Cambridge University Press, 2000.

Peters, J., Mooij, J., Janzing, D., and Schölkopf, B. Causal discovery with continuous additive noise models. *Journal of Machine Learning Research (JMLR)*, 15:2009–2053, 2014.

Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. *Dataset shift in machine learning*. The MIT Press, 2009.

Rudy, S. H., Brunton, S. L., Proctor, J. L., and Kutz, J. N. Data-driven discovery of partial differential equations. *Science Advances*, 3(4), 2017. doi: 10.1126/sciadv.1602614.

Schmidt, M. and Lipson, H. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009. ISSN 0036-8075. doi: 10.1126/science.1165893.

Schmidt, M., Fung, G., and Rosales, R. Optimization methods for L_1-regularization. UBC Technical Report TR-2009-19, 2009.

Sugiyama, M., Krauledat, M., and Müller, K.-R. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8(May): 985–1005, 2007.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL http://arxiv.org/abs/1605.02688.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

Zaremba, W., Kurach, K., and Fergus, R. Learning to discover efficient mathematical identities. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 1278–1286. Curran Associates, Inc., 2014.