Extrapolation and learning equations – Abstract

Georg Martius & Christoph H. Lampert
IST Austria
Am Campus 1, 3400 Klosterneuburg, Austria
{gmartius,chl}@ist.ac.at

In classical machine learning, regression is treated as a black box process of identifying a suitable function from a hypothesis set without attempting to gain insight into the mechanism connecting inputs and outputs. In the natural sciences, however, finding an interpretable function for a phenomenon is the prime goal as it allows to understand and generalize results. This paper proposes a new type of function learning network, called equation learner (EQL), that can learn analytical expressions and is able to extrapolate to unseen domains. It is implemented as an end-to-end differentiable feed-forward network and allows for efficient gradient based training. Due to sparsity regularization concise interpretable expressions can be obtained. We also devise a model selection strategy tailored to the function learning setting, such that often the true underlying source expression can be identified. In this way the behavior of physical systems can be learned and effectively predicted even in new situations. A full description of the method can be found in Martius and Lampert [2].

The following section describes the setting of regression and extrapolation. Afterwards we introduce our method and discuss the architecture, its training, and its relation to prior art. We present our results in the Section *Experimental evaluation* and close with conclusions.

Extrapolation setting and function learning network

We consider a multivariate regression problem with a training set $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$. Because our main interest lies on extrapolation in the context of learning the dynamics of physical systems we assume the data originates from an unknown analytical function (or system of functions), $\phi : \mathbb{R}^n \to \mathbb{R}^m$ with additive zero-mean noise, ξ, i. e. $y = \phi(x) + \xi$ and $\mathbb{E}\xi = 0$. The general task is to learn a function $\psi : \mathbb{R}^n \to \mathbb{R}^m$ that approximates the true functional relation as well as possible in the squared loss sense. In practice, we the empirical error on training or test data D, $E(D) = \frac{1}{N}\sum_{i=1}^{N} \|\psi(x_i) - y_i\|^2$. Ideally, we want to identify the true underlying function ϕ, which we will attempt be searching for the simplest candidate model ψ that explains the data well, as detailed below.

For evaluation, we distinguish between the *interpolation* setting where training and test data are sampled from the same distribution and the *extrapolation* setting, where the test data is outside the training domain. In the example of the robot arm, for instance, the training may be restricted to a certain joint angle range or maximal velocity. For testing we want to make predictions about, for instance, unseen positions and higher velocities.

The main model we propose is a multi-layered feed-forward network with computational units specifically designed for the extrapolation regression tasks. For an L-layer network, there are $L-1$ hidden layers, each consisting of a linear mapping followed by non-linear transformations. For simplicity, here we describe the network as if each hidden layer had the same structure (k' inputs, k outputs). The linear mapping at level l maps the k'-dimensional input $y^{(l-1)}$ to the d-dimensional intermediate representation z given by

$$z^{(l)} = W^{(l)} y^{(l-1)} + b^{(l)}, \tag{1}$$

where $y^{(l-1)}$ is the output of the previous layer, with the convention $y^{(0)} = x$. The weight matrix $W^{(l)} \in \mathbb{R}^{d \times k'}$ and the bias vector $b^{(l)} \in \mathbb{R}^d$ are free parameters that are learned during training. The

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

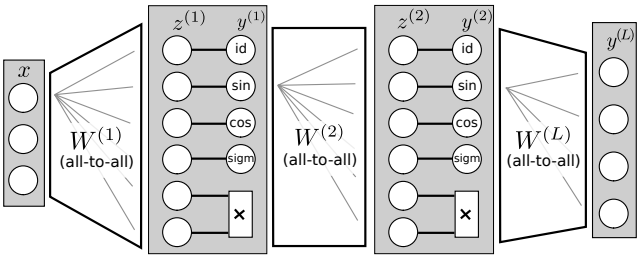


Figure 1: Network architecture of the proposed Equation Learner (EQL) for 3 layers ($L = 3$) and one neuron per type ($u = 4, v = 1$).

non-linear transformation contains u *unary units*, $f_i : \mathbb{R} \to \mathbb{R}$, for $i = 1, \ldots, u$, and v *binary units*, $g_j : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ for $j = 1, \ldots, v$. The latter implement multiplications: $g(a, b) = ab$. Their outputs are concatenated to form the layer output

$$y^{(l)} := \left(f_1(z_1^{(l)}), f_2(z_2^{(l)}), \ldots, f_u(z_u^{(l)}), g_1(z_{u+1}^{(l)}, z_{u+2}^{(l)}), \ldots, g_v(z_{u+2v-1}^{(l)}, z_{u+2v}^{(l)})\right). \tag{2}$$

In total, the nonlinear stage has $k = u + v$ outputs and $d = u + 2v$ inputs. The unary units, $f_1, \ldots, f_u$ receive the respective component, $z_1, \ldots, z_u$ as inputs, and each unit may be one of the following base functions as specified in a fixed type parameter $I_i \in \{0, 1, 2, 3\}$

$$f_i(z_i) := \begin{cases} z_i & \text{if } I_i = 0, \\ \sin(z_i) & \text{if } I_i = 1, \\ \cos(z_i) & \text{if } I_i = 2, \\ \text{sigm}(z_i) & \text{if } I_i = 3, \end{cases} \qquad \text{for } i = 1, \ldots, u, \tag{3}$$

where $\text{sigm}(z) = \frac{1}{1+e^{-z}}$ is the standard sigmoid function. Finally, the last layer computes the regression values by a linear read-out

$$y^{(L)} := W^{(L)} y^{(L-1)} + b^{(L)}. \tag{4}$$

The architecture is depicted in Fig. 1. We call the new architecture Equation Learner (EQL) and denote the function it defines by ψ.

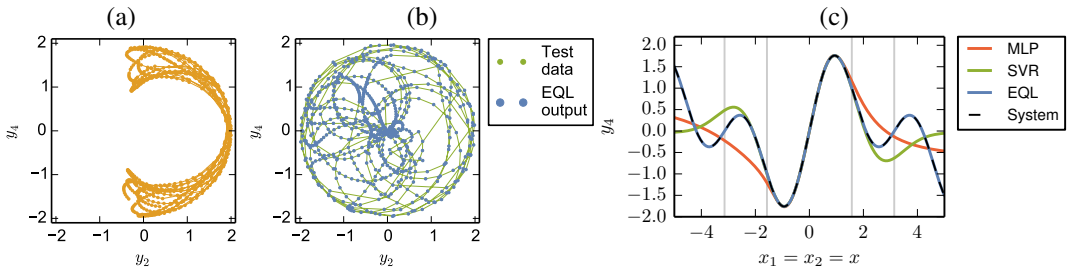
Network training: The EQL is fully differentiable in its free parameters $\theta = \{W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)}\}$, which allows us to train it in an end-to-end fashion using back-propagation. We adopt a Lasso-like objective [4],

$$L(D) = \frac{1}{N} \sum_{i=1}^{|D|} \|\psi(x_i) - y_i\|^2 + \lambda \sum_{l=1}^{L} \left|W^{(l)}\right|_1, \tag{5}$$

that is, a linear combination of L_2 loss and L_1 regularization, and apply a stochastic gradient descent algorithm with mini-batches and Adam [1] for calculating the updates.

The role of the L_1 regularization is to encourage networks with sparse connections, matching the intuition that a typical formula describing a physical system contains only a small number of terms, each operating only on a few variables. However, this type of regularization might lead to an early commitment to a particular sparse configuration, where the optimization gets stuck.

Therefore, we follow a hybrid regularization strategy: at the beginning of the training procedure ($t < t_1$) we use no regularization ($\lambda = 0$), such that parameters can vary freely and reach reasonable starting points. Afterwards, we switch on the regularization by setting λ to a nonzero value, which has the effect that a sparse network structure emerges. Finally, for the last steps of the training ($t > t_2$) 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. This ensures that the learned model finds not only a function of the right parametric form, but also fits the observed values as closely as possible. In practice, we use $t_1 = \frac{1}{4}T$ and $t_2 = \frac{19}{20}T$, where T is total number of update steps, chosen large enough to ensure convergence. We refer to Martius and Lampert [2] for more details.



(d)

	EQL	MLP	SVR
extrapolation error	0.0003 ± 0.00003	0.58 ± 0.03	0.25

Figure 2: Double pendulum kinematics. (a) training trajectory (in y-space). (b) extrapolation test trajectory (in y-space) with output of a learned EQL instance. (c) slices of output y_4 for inputs $x_1 = x_2 = x$ for the true system, one of EQL, MLP, and SVR instances. (d) numeric results: mean and standard deviation of the root mean squares error (RMS) ($\sqrt{E}$) on different test sets for 10 random initializations. Note, that predicting 0 would yield a mean error of 0.84.

Model selection for extrapolation: Model selection serves in our approach two purposes. On the one hand to select the right hyper-parameters, e. g. the number of layers, the number of units and the regularization constant, and on the other hand to select particular instances that are likely to contain the true formula. But how can we tell? Using Occams razor principle: the simplest formula is most likely the right one. Intuitively, if we have the choice between $cos(x)$ and its truncated power series approximation $1 - x^2/2 + x^4/24$, the first one is preferred. We use the number of active hidden units in the network as a proxy for the complexity of the formula. In any case, this argumentation is only correct if the model explains the data well, i. e. it has a low validation error. So we have a dual objective to minimize, which we solve by ranking the instances w. r. t. validation error and sparsity and select the one with the smallest L_2 norm (in rank-space).

Experimental evaluation

Double pendulum kinematics. First we consider real double pendulum where the forward kinematics should be learned. For that we use recorded trajectories of a real double pendulum [3]. The task here is to learn the position of the tips of the double pendulum segments from the given joint angles (x_1, x_2). These positions where not measured such that we supply them by the following formula: $y_1 = \cos(x_1), y_2 = \cos(x_1) + \cos(x_1 + x_2), y_3 = \sin(x_1), y_4 = \sin(x_1) + \sin(x_1 + x_2)$ where (y_1, y_3) and (y_2, y_4) correspond to x-y-coordinates of the first and second end-point respectively. The dataset contains two short trajectories. The first covers only part of the domain (input as well as output) and consists of 819 samples where 10% was used as validation set (randomly sampled), see Fig. 2(a). The second trajectory corresponds to a behavior with several spins of both pendulum segments such that a much larger domain is covered. Nevertheless the angle values are confined to $[-\pi, \pi]$. We use this trajectory as extrapolation test set. The trajectory and the outputs of our method are shown in Fig. 2(b). The prediction for unseen domains is perfect, which is also illustrated in a systematic sweep, see Fig. 2(c). The performance of MLP is off already near the training domain. SVR is a bit better, but still does not give usable predictions for the test data, see also the root means square error in Fig. 2(d).

Robotic arms. A more complicated task is to learn the forward kinematics of multi-segment robotic arms. We consider planar arms with 3, 4, and 5 joints, where each segment is 0.5 units long. For training the arm is controlled by sinusoidal joint target angles with amplitude in $[-\pi/2, \pi/2]$, each joint with a different frequency. The number of data points are: 3000, 6000, and 18000 for the 3, 4, and 5 segment arms respectively, with added noise as above. For testing extrapolation performance the amplitude $[-\pi, \pi]$ was used. Note that the extrapolation space is much larger than the training space. The task is to predict the coordinates of the end-effector of the arms (*kin-3-end*, *kin-4-end*) and the coordinates of all segment positions *kin-5-all*. The numerical results, see Tab. 1, shows that our method is able to extrapolate in these cases.

Table 1: Extrapolation performance for *kinematic of robotic arms* and for *formula learning*. RMS and standard deviations for 5 random initializations. Interpolation error for all methods is around 0.01 (noise level).

	kin-3-end	kin-4-end	kin-5-all	F-1 (near)	F-1 (far)
EQL	0.017 ± 0.000	0.012 ± 0.000	0.011 ± 0.000	0.015 ± 0.005	0.026 ± 0.015
MLP	0.389 ± 0.014	0.415 ± 0.020	0.346 ± 0.013	0.32 ± 0.12	0.920 ± 0.420
SVR	0.235	0.590	0.260	0.28	1.2

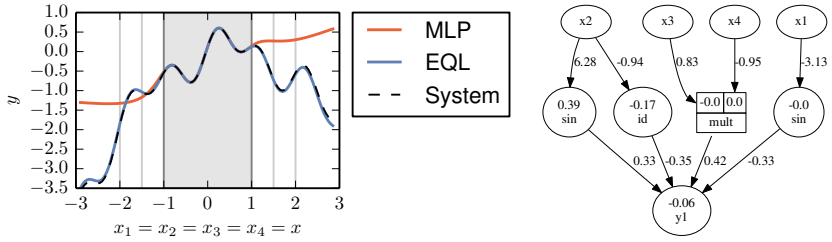


Figure 3: Formula learning analysis for F-1. (left) y for a single cut through the input space for the true system equation and for an instance of EQL and MLP. (right) shows the learned networks correspondingly, where numbers on the edges correspond to weights w and the biases b are shown in the nodes. The corresponding formula extracted from the network is: $-0.33\sin(-3.13x_1) + 0.33\sin(6.28x_2 + 0.39) + 0.33x_2 - 0.056 - 0.33x_3x_4$.

Learning complex formula. In order to find out whether EQL can also learn more complicated formulas, we consider several examples with four-dimensional input and one-dimensional output, where we only present one here (see [2] for more)

$$y = 1/3\left(\sin(\pi x_1) + \sin\left(2\pi x_2 + \pi/8\right) + x_2 - x_3x_4\right) \qquad \text{F-1} \qquad (6)$$

Table 1 shows the numerical results. Again, all methods are able to interpolate, but only EQL achieves good extrapolation results. Figure Fig. 3 illustrates the performance and a learned network visually. It shows one of the model-selected instances and shows that the correct formula was identified, so correct predictions can be made even far outside the training region (much further than illustrated). In the full paper [2] we also present failure cases, which may occur for more complex equations.

Summary

The proposed EQL network architecture can learn analytic expressions that typically occur in equations governing physical, in particular mechanical, systems. With the right regularization and model selection strategy we are able to identify, in many cases, the true underlying formula from the data. Future work is to make the system less likely to get stuck in local minima, which happens as the underlying system becomes more and more complicated and furthermore, to increase function class by adding more base functions.

Acknowledgments: GM received funding from the People Programme (Marie Curie Actions) of the European Union's (FP7/2007-2013) under REA grant agreement no. [291734].

References

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

[2] G. Martius and C. H. Lampert. Extrapolation and learning equations, 2016, arXiv preprint: abs/1610.02995,

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

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