EXTRAPOLATION AND LEARNING EQUATIONS

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

ABSTRACT

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 novel 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. Often the true underlying source expression is identified.

INTRODUCTION

The quality of a model is typically measured by its ability to generalize from a training set to previously unseen data from the same distribution. In regression tasks generalization essentially boils down to interpolation if the training data is sufficiently dense. As long as models are selected correctly, i. e. in a way to not overfit the data, the regression problem is well understood and can – at least conceptually – be considered solved. However, when working with data from real-world devices, e. g. controlling a robotic arm, interpolation might not be sufficient. It could happen that future data lies outside of the training domain, e. g. when the arm is temporarily operated outside of its specifications. For the sake of robustness and safety it is desirable in such a case to have a regression model that continues to make good predictions, or at least does not fail catastrophically. This setting, which we call *extrapolation generalization*, is the topic of the present paper.

We are particularly interested in regression tasks for systems that can be described by real-valued analytic expression, e. g. mechanical systems such as a pendulum or a robotic arm. These are typically governed by a highly nonlinear function but it is nevertheless possible, in principle, to infer their behavior on an extrapolation domain from their behavior elsewhere. We make two main contributions: 1) a new type of network that can learn analytical expressions and is able to extrapolate to unseen domains and 2) a model selection strategy tailored to the extrapolation setting.

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.

REGRESSION AND EXTRAPOLATION

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 function ϕ may, for instance, reflect a system of ordinary differential equations that govern the movements of a robot arm or the like. 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, i. e. achieves minimal expected error $\mathbb{E}\|\psi(x) - \phi(x)\|^2$. In practice, we only have particular examples of the function values available and measure the quality of predicting in terms of the empirical error on training or test data D,

$$E(D) = \frac{1}{N}\sum_{i=1}^{N} \|\psi(x_i) - y_i\|^2 \,. \tag{1}$$

If training and test data are sampled from the same distribution then we speak about an *interpolation* problem. In the *extrapolation* setting the training data is assumed to cover only a limited range of the data 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 the unseen domains, e. g. for higher velocities. To succeed in this task, it is essential to identify the underlying functional relationship instead of just minimizing the empirical error, as detailed below. As usual, we split the data that is available at training time into a part for model training and a part for validation or model selection.

Learning a network for function extrapolation

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 of notation, we explain the network as if each hidden layer had the same structure (k' inputs, k outputs). In practice, each layer can be designed independently of the others, of course, as long as input/output dimensions match.

The linear mapping at level l maps the k'-dimensional input $y^{(l-1)}$ to the d-dimensional intermediate representation z given by

$$z^{(l)} = W^{(l)} y^{(l-1)} + b^{(l)}, \tag{2}$$

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 non-linear transformation contains u *unary units*, $f_i : \mathbb{R} \to \mathbb{R}$, for $i = 1,\dots,u$, and v *binary units*, $g_j : \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ for $j = 1,\dots,v$. Their outputs are concatenated to form the layer output

$$y^{(l)} := \Big(f_1(z_1^{(l)}), f_2(z_2^{(l)}), \dots, f_u(z_u^{(l)}), g_1(z_{u+1}^{(l)}, z_{u+2}^{(l)}), \dots, g_v(z_{u+2v-1}^{(l)}, z_{u+2v}^{(l)}) \Big) \,. \tag{3}$$

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

$$f_i(z_i) := \begin{cases} z_i & \text{if } I_i = 0, \\ \sin(z_i) & \text{if } I_i = 1, \\ \cos(z_i) & \text{if } I_i = 2, \\ \mathrm{sigm}(z_i) & \text{if } I_i = 3, \end{cases} \qquad \text{for } i = 1,\dots,u, \tag{4}$$

where $\mathrm{sigm}(z) = \frac{1}{1+e^{-z}}$ is the standard sigmoid function. The binary units, $g_1,\dots,g_v$ receive the remaining component, $z_{u+1},\dots,z_{u+2v}$, as input in pairs of two. They are *multiplication units* that compute the product of their two input values:

$$g_j(z_{u+2j-1}, z_{u+2j}) := z_{u+2j-1}\cdot z_{u+2j} \qquad \text{for } j = 1,\dots,v. \tag{5}$$

Finally, the L-th and last layer computes the regression values by a linear read-out

$$y^{(L)} := W^{(L)} y^{(L-1)} + b^{(L)}. \tag{6}$$

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

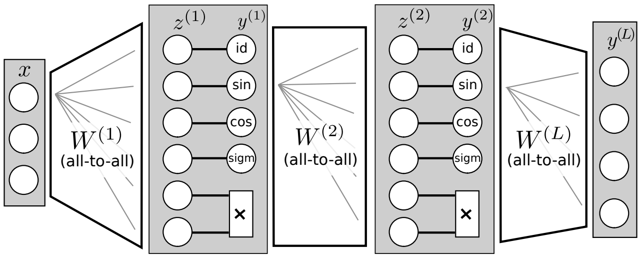


Figure 1: Network architecture of the proposed Equation Learner (EQL) for 3 layers ($L = 3$) and one neuron per type ($u = 4, v = 1$).

Discussion of the architecture

The proposed network architecture differs in two main aspects from typical feed-forward networks: the existence of multiplication units and the possibility of *sine* and *cosine* as nonlinearities for the unary units. Both design choices are motivated by our objective of learning a system of equations that govern a physical system and can extrapolate to new parts of the input space.

Sigmoid nonlinearities are the canonical choice of *activation function* for *artificial neural networks* (ANN) and proved to be successful. In fact, we include sigmoids in our architecture, making it a super class of ANNs. However, they were typically disabled by the training procedure corresponding to their absence in the considered physical equations. Other, predominantly local nonlinearities, in particular *radial basis functions* Broomhead & Lowe (1988) we do not include, since one cannot expect them to extrapolate at all. Further nonlinearities, such as *(square) roots* and *logarithms*, could in principle be useful for learning physical equations, but they pose problems because their domains of definition is restricted to positive inputs. We leave the task of incorporating them in a principled way to future work.

The ability to multiply two values is a second crucial component of our network architecture. Again, it is inspired by the typical form of physical equations, where multiplication of components is arguably second common basic operation after addition (which the linear layers can perform). Multiplication was introduced into neural networks long ago as product-units Durbin & Rumelhart (1989) and Pi-Sigma-unit Shin & Ghosh (1991). The product-units have large fan-in that compute products over all their inputs, potentiated by the respective weights. The result is typically the behavior of a high order polynomial, which are powerful function approximators, but rarely occur in physical equations. Polynomials are also known to require careful fine-tuning in order not to overfit, which makes them a risky choice for the purpose of extrapolation. The Pi-Sigma units are multiplication units with a fixed number of factors and our multiplication units are a special for 2 factors. We find that multiplying just two values at a time is well adjusted to the task we aim at, as it allows to control the maximal degree of the learned polynomial by the depth of the network.

Finally, each layer of the network contains unary units that act as *identity* maps, which in particular gives the network the option to learn functions with smaller number of nonlinearities than the total network depths.

Network training

The EQL is fully differentiable in its free parameters $\theta = \{W^{(1)}, \dots, W^{(L)}, b^{(1)}, \dots, b^{(L)}\}$, which allows us to train it in an end-to-end fashion using back-propagation. We adopt a Lasso-like objective Tibshirani (1996),

$$\mathcal{L}(D) = \frac{1}{N} \sum_{i=1}^{|D|} \|\psi(x_i) - y_i\|^2 + \lambda \sum_{l=1}^{L} \left|W^{(l)}\right|_1 , \tag{7}$$

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 Kingma & Ba (2015) for calculating the updates:

$$\theta_{t+1} = \theta_t + \text{Adam}\left(\frac{\partial \mathcal{L}\left(D_{(t)}\right)}{\partial \theta}, \alpha\right), \tag{8}$$

where $D_{(t)}$ denotes the current mini-batch and α is the stepsize parameter. The choice of Adam is not critical and standard stochastic gradient descent also works. In all numerical experiments we use $\alpha = 0.001$ and a mini-batch size of 20.

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, in a non-convex setting where local minima are likely to occur, this type of regularization can have an undesirable side-effect: during the course of the optimization the weights hardly ever change their sign. The reason is that the regularization leads to a constant rate of weight decay whereas the counteracting derivative with respect to the square loss is proportional to the backpropagated error signal and the input to the unit. The latter contributions are often smaller along paths with small weights, such that many weights go to zero and stay there. Additionally, any non-zero regularization term causes the learned weights to reflect a trade-off between minimizing the loss and the regularizer. Although, this can lead to improved generalization, it also results in a systematic underestimation of the function values.

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. We observed that the exact choice of breakpoints t_1 and t_2 is not critical. 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. T was selected large enough to ensure convergence. Note, that convergence to a sparse structure is important here, so early stopping will be disadvantageous.

Model selection for extrapolation

EQL networks have a number of hyper-parameters, e. g. the number of layers, the number of units and the regularization constant. Unfortunately, standard techniques for model selection, such as evaluation on a hold-out set or cross-validation, will not be optimal for our purpose, since they rely on interpolation quality. In order to extrapolate the network has to find the "right" 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, see Appendix A1 for details. One could also think of differentiating between the unit types. 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), see Eq. (15).

Furthermore, the optimization process may only find a local optimum of the training objective, which depends on the initialization of the parameters. We use independent runs to quantify expected performance deviations.

Related work

In the field of machine learning, regression is often treated as a black box process of identifying a suitable real-valued function from a hypothesis set, e. g. a reproducing kernel Hilbert space for Gaussian Processes Regression (GPR) Williams & Rasmussen (2006) or Support Vector Regression (SVR) Smola & Schölkopf (2004), or a multi-layer network of suitable expressive power Specht (1991). The goal is to find a prediction function that leads to a small expected error on future data, not necessarily to gain insight into the mechanism of how the output values derive from the inputs. The goal of finding an interpretable function is rather common in the natural sciences, such as biology, where high noise levels and strong inter-system variability often make it important to rely on external prior knowledge, and finding a "biologically plausible" model is often preferable over finding one that makes the highest prediction accuracy. As a consequence, model classes are often highly constrained, e. g. allowing only for sparse linear models.

The task of learning a true, nonlinear, functional dependence from observing a physical system, has received little attention in the machine learning literature so far, but forms the basis of the field of *system identification*. There, typically the functional form of the system is known and only the parameters have to be identified. Another approach is to model the time evolution with autoregressive models or higher order convolution integrals (Volterra series) but learning analytic formulas is not common.

Causal learning is an area of recent research that aims at identifying a causal relation between multiple observables, which are typically the result of a physical process. Classically, this tasks reduces to finding a minimal graphical model based only on tests of conditional independence Pearl (2000). Although very successful in some fields, this classical approach only provides a factorization of the problem, separating causes and effects, but it leaves the exact functional dependency unexplained. Recent extensions of causal learning can take a functional view, but typically do not constrain the regression functions to physically plausible ones, but rather constrain the noise distributions Peters et al. (2014). The topic of learning a regression function with emphasis on *extrapolation* performance has not been studied much in the literature so far. Existing work on time series prediction deals with extrapolation in the temporal domain, i. e. predict the next value(s) Wiener (1949). By our nomenclature, this is typically rather an interpolation task, when the prediction is based on the behaviour of the series at earlier time steps but with similar value distribution Müller et al. (1997); Györfi et al. (2013). Extrapolating in the data domain implies that the data distribution at prediction time will differ from the data distribution at training time. This is traditionally called the *domain adaptation* setting. In particular, since we assume a common labeling function, our setting would fall under the *covariate shift* setting Quionero-Candela et al. (2009). Unfortunately, this connection is not particularly useful for our problem. As domain adaptation typically does not make additional assumptions about how the data distribution may change, existing methods need access to some unlabeled data from the test distribution already at training time Ben-David et al. (2010). In our setting this is not possible to obtain.

On the technical level, EQL networks are an instance of general feed-forward networks for function approximation Bishop (1995). In contrast to recent trends towards *deep learning* Bengio (2009); Bengio et al. (2013), our goal is not to learn any data representation, but to learn a function which compactly represents the input-output relation and generalizes between different regions of the data space, like a physical formula. Structurally, EQL networks resemble *sum-product networks (SPNs)* Poon & Domingos (2012) and *Pi-Sigma networks (PSNs)* Shin & Ghosh (1991), in the sense that both are based on directed acyclic graphs with computational units that allows for summation and multiplication. Otherwise, SPNs are different as they act as efficient alternative to probabilistic graphical models for representing probability distributions, whereas EQL networks are meant for the classical task of function approximation. In PSNs each output needs to be passed through multiplicative units, whereas in EQL multiplication is optional.

Finding equations for observations is also known as symbolic regression where a search is performed in a certain function space, typically done with evolutionary computation. With these techniques it is possible to discover physical laws such as invariants and conserved quantities Schmidt & Lipson (2009). Unfortunately, the computational complexity/search time explodes 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.

Experimental evaluation

We demonstrate the ability of EQL to learn physically inspired models with good extrapolation quality by experiments on synthetic and real data. For this, we implemented the network training and

Table 1: Numeric results on *pendulum* dataset. Reported are the mean and standard deviation of the root mean squares error (RMS) ($\sqrt{E}$, Eq. (1)) on different test sets for 10 random initializations.

	interpolation	extrapol. (near)	extrapol. (far)
EQL	0.0102 ± 0.0000	0.012 ± 0.002	0.016 ± 0.007
MLP	0.0138 ± 0.0002	0.150 ± 0.012	0.364 ± 0.036
SVR	0.0105	0.041	0.18

evaluation procedure in *python* based on the *theano* framework Theano Development Team (2016). We will make the code for training and evaluation public after acceptance of the manuscript.

Pendulum. We first present the results of learning the equations of motion for a very simple physical system: a pendulum. The state space of a pendulum is $X = \mathbb{R} \times \mathbb{R}$ where the first value is the angle of the pole in radians and the second value is the angular velocity. In the physics literature, these are usually denoted as (θ, ω), but for our purposes, we call them (x_1, x_2) in order to keep the notation consistent between experiments. The pendulum's dynamic behavior is governed by the following two ordinary differential equations:

$$\dot{x}_1 = x_2 \qquad \text{and} \qquad \dot{x}_2 = -g \sin(x_1)\,, \tag{9}$$

where $g = 9.81$ is the gravitation constant.

We divide each equation by g in order to balance the output scales and form a regression problem with two output values, $y_1 = \frac{1}{g} x_2$ and $y_2 = -\sin(x_1)$.

As training data, we sample 1000 points uniformly in the hypercube $[-h, h] \times [-h, h]$ for $h = 2$. Note that this domain contains more than half of a sine period, so it should be sufficient to identify the analytic expression. The target values are disturbed by Gaussian noise with standard derivation $\sigma = 0.01$. We also define three test sets, each with 1000 points. The *interpolation test set* is sampled from the same data distribution as the training set. The *extrapolation (near) test set* contains data sampled uniformly from the data domain $[-\frac{3}{2}h, \frac{3}{2}h] \times [-\frac{3}{2}h, \frac{3}{2}h] \setminus [-h, h] \times [-h, h]$, which is relatively near the training region and the *extrapolation (far) test set* extends the region to further outside: $[-2h, 2h] \times [-2h, 2h] \setminus [-h, h] \times [-h, h]$. We train a 2-layer EQL and perform model selection among the hyper-parameters: the regularization strength $\lambda \in 10^{\{-7, -6.3, -6, -5.3, -5, -4.3, -4, -3.3, -3\}}$ and the number of nodes $\frac{1}{4}u = v \in \{1, 3, 5\}$. All weights are randomly initialized from a normal distribution with $\sigma = \sqrt{1/(k' + d)}$. The unit selection I is set such that all unit types are equally often. To ensure convergence we chose $T = 10000$ epochs. We compare our algorithm to a standard multilayer perceptron (MLP) with $\tanh$ activation functions and possible hyperparameters: λ as for EQL, number of layers $L \in \{2, 3\}$, 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 basis function kernel with width $\gamma \in \{0.05, 0.1, 0.2, 0.5, 1.0\}$.

Numeric results are reported in Tab. 1. As expected all models are able to interpolate well with a test error on the order of the noise level ($\sigma = 0.01$). For extrapolation however, the performance differ between the approaches. For MLP the prediction quality decreases quickly when leaving the training domain. SVR remains a bit better in the near extrapolation but also fails catastrophically on the far extrapolation data. EQL, on the other hand, extrapolates well, both near and far away from the training domain. The reasons can be seen in Figure 2: while the MLP and SVR simply learns a function that interpolates the training values, EQL finds the correct functional expression and therefore predicts the correct values for any input data.

Double pendulum kinematics. The second system we consider real double pendulum where the forward kinematics should be learned. For that we use recorded trajectories of a real double pendulum Schmidt & Lipson (2009). 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

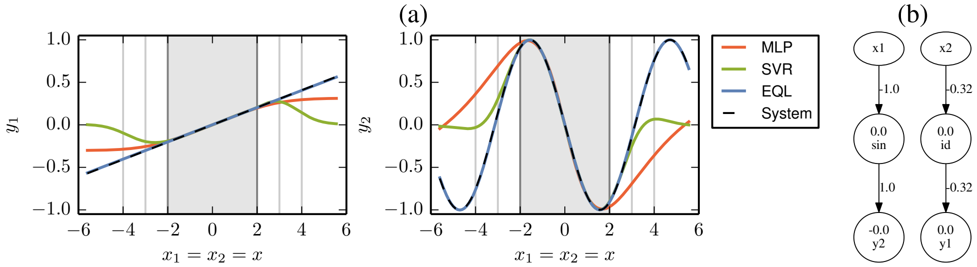
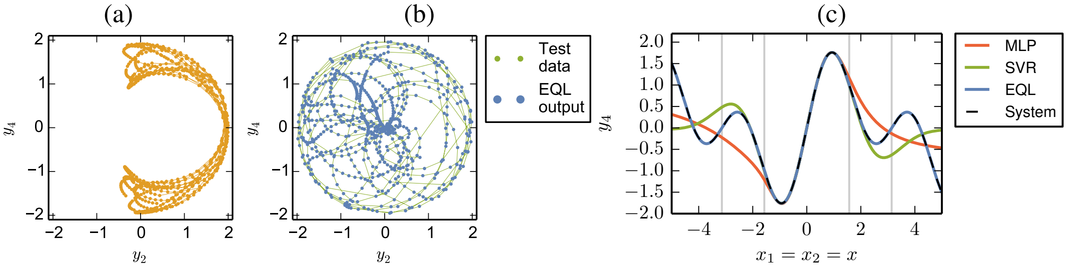


Figure 2: Learning pendulum dynamics. (a) slices of outputs y_1 (left) and y_2 (right) for inputs $x_1 = x_2 = x$ for the true system equation (Eq. 9) and one of EQL, MLP, SVR instances. The shaded area marks the training region and the vertical bars show the size of the *near* and *far* extrapolation domain. (b) one of the learned networks. Numbers on the edges correspond to the entries of W and numbers inside the nodes show the bias values b. All weights with $|w| < 0.01$ and orphan nodes are omitted. Learned formulas: $y_1 = 0.103x_2$, $y_2 = \sin(-x_1)$, which are correct up to symmetry ($^1/_g = 1.01$).



(d)

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

Figure 3: 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, see Tab. 1 for details. Note, that predicting 0 would yield a mean error of 0.84.

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. 3(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. 3(b). The prediction for unseen domains is perfect, which is also illustrated in a systematic sweep, see Fig. 3(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. 3(d).

Model selection is performed to determine λ as above, $u = v \in \{3, 5\}$, (MLP: $k \in \{5, 10, 20\}$) and layer number $L \in \{2, 3\}$.

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. 2, shows that our method is able to extrapolate in these cases. Model selection as above with $u = v \in \{10, 20\}$, (MLP: $k \in \{10, 50\}$) and layer number $L \in \{2, 3, 4\}$. To illustrate the dependence on the amount of

Table 2: Extrapolation performance for *kinematic of robotic arms*. See Tab. 1 for details. Standard deviations for 5 random initializations. Interpolation error for all methods is around 0.012 ± 0.02

	kin-3-end	kin-4-end	kin-5-all
EQL	0.017 ± 0.000	0.012 ± 0.000	0.011 ± 0.000
MLP	0.389 ± 0.014	0.415 ± 0.020	0.346 ± 0.013
SVR	0.235	0.590	0.260

noise and the number of available training points we provide a quantification in Appendix A2. In short, increasing noise can be compensated by increasing amount of data to keep the performance.

Learning complex formula. In order to find out whether EQL can also learn more complicated formulas, we consider three examples with four-dimensional input and one-dimensional output:

$$y = 1/3 \left(\sin(\pi x_1) + \sin\left(2\pi x_2 + \pi/8\right) + x_2 - x_3 x_4\right) \qquad \text{F-1} \qquad (10)$$

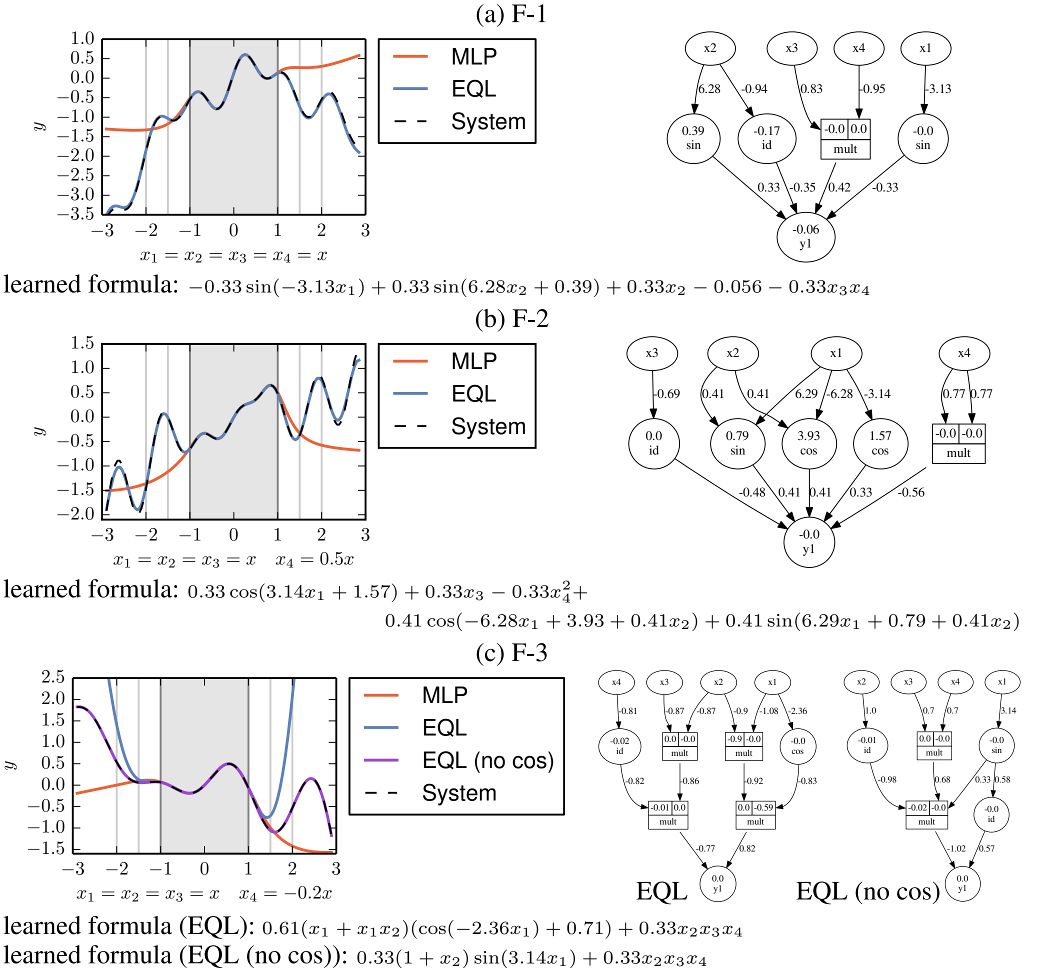
$$y = 1/3 \left(\sin(\pi x_1) + x_2 \cos(2\pi x_1 + \pi/4) + x_3 - x_4^2\right) \qquad \text{F-2} \qquad (11)$$

$$y = 1/3 \left((1 + x_2)\sin(\pi x_1) + x_2 x_3 x_4\right) \qquad \text{F-3} \qquad (12)$$

The first equation requires only one hidden layer to be represented. The second equation and third equation should requires two hidden layers. In particular, F-2 contains a product of x_2 and $\cos$ and F-3 contains a product of three terms, and we use it to test if our restriction to only pairwise product units causes problems for more complex target functions. We follow the same procedure as in the pendulum case for building training and test sets, though with $h = 1$ as input data range. We use 10000 points for training set and validation set (90%-10% split) and 5000 points for each of the test sets. Model selection for EQL is performed as above using the number of layers $L \in 2, 3, 4$. The number of units is set to $\frac{1}{4}u = v = 10$. For the MLP, we select L and λ from the same set as above as well as $k \in \{10, 30\}$.

Table 3 shows the numerical results. Again, all methods are able to interpolate, but only EQL achieves good extrapolation results, except for equation F-3. There it settles in 9 out of 10 cases into a local minimum and finds only an approximating equation that deviates outside the training domain. Interestingly, if we restrict the base functions to not contain cosine, the algorithm finds the right formula. Note, the sparsity of the correct formula is lower than those of the approximation, so it should be selected if found. Figure Fig. 4 illustrates the performance and the learned networks visually. It shows one of the model-selected instances for each case. For F-1 the correct formula was identified, so correct predictions can be made even far outside the training region (much further than illustrated). For F-2 the network provided us with a surprise, because it yields good extrapolation performance with only one hidden layer! How can it implement $x_2 \cos(2\pi x_1 + \pi/4)$? Apparently it uses $1.21(\cos(-2\pi x_1 + \pi + \pi/4 + 0.41x_2) + \sin(2\pi x_1 + \pi/4 + 0.41x_2))$ which is a good approximation for $x_2 \in [-2, 2]$. The sparsity of this solution is 5 whereas the true solution needs at least 6, which explains its selection. For F-3 the suboptimal local minima uses some strange way of approximating $(1 + x_2)\sin(x_1)$ using $(x_1 + x_1 x_2)\cos(\beta x_1)$, which deviates fast, however the true solution would be sparser but was not found. Only if we remove cosine from the base functions we get always the correct formula, see Fig. 4(c).

X-Ray transition energies. As a further example we consider data measured in atomic physics. When shooting electron beams onto atoms one can excite them and they consequently emit x-ray radiation with characteristic peak energies. For each element/isotope these energies are different as they correspond to the potential difference between the electron shells, such that one can identify elements in a probe this way. The data is taken from Deslattes et al. (2003), where we consider one specific transition, called the $K\,\alpha_2$ line, because it was measured for all elements. The true relationship between atomic number Z and transition energies is complicated, as it involves many body interactions and no closed-form solution exists. Nevertheless we can find out which relationships our system proposes. It is known that the main relationship is $K\,\alpha_2 \propto Z^2$ according to Moseley's law. Further correction terms for elements with larger Z are potentially of higher order. We have data for elements with $10 \leq Z \leq 100$, which is split into training/validation sets in the range $[10, 91]$ (70/10 data points) and extrapolation test set in the interval $[92, 100]$ (14 data points because of isotops). Since we have so little data we evaluate the performance for 10 independent training/validation



(a) F-1

learned formula: $-0.33\sin(-3.13x_1) + 0.33\sin(6.28x_2 + 0.39) + 0.33x_2 - 0.056 - 0.33x_3x_4$

(b) F-2

learned formula: $0.33\cos(3.14x_1 + 1.57) + 0.33x_3 - 0.33x_4^2 +$
$0.41\cos(-6.28x_1 + 3.93 + 0.41x_2) + 0.41\sin(6.29x_1 + 0.79 + 0.41x_2)$

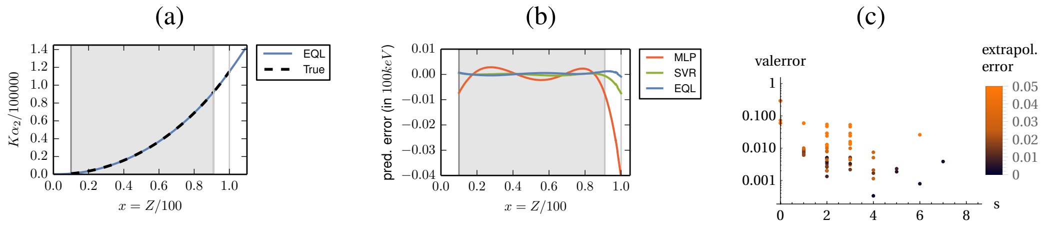
(c) F-3

learned formula (EQL): $0.61(x_1 + x_1x_2)(\cos(-2.36x_1) + 0.71) + 0.33x_2x_3x_4$
learned formula (EQL (no cos)): $0.33(1 + x_2)\sin(3.14x_1) + 0.33x_2x_3x_4$

Figure 4: Formula learning analysis. (a) for F-1, (b) for F-2, and (c) for F-3. (left) y for a single cut through the input space for the true system equation (10–12), and for an instance of EQL, and MLP. (right) shows the learned networks correspondingly, see Fig. 2 for details. The formula representations where extracted from the networks. For F-3 the algorithm fails with the overcomplete base and typically (9/10 times) ends up in a local minima. With less base function (no cosine) the right formula is found. Both results are presented. See text for a discussion.

Table 3: Interpolation and extrapolation performance for *formula learning*. See Tab. 1 for details.

dataset	method	interpolation	extrapol. (near)	extrapol. (far)
F-1	EQL	0.010 ± 0.000	0.015 ± 0.005	0.026 ± 0.015
	MLP	0.011 ± 0.000	0.32 ± 0.12	0.920 ± 0.420
	SVR	0.011	0.28	1.2
F-2	EQL	0.01 ± 0.00	0.013 ± 0.004	0.026 ± 0.019
	MLP	0.01 ± 0.00	0.2 ± 0.014	0.49 ± 0.043
	SVR	0.011	0.3	0.94
F-3	EQL	0.01 ± 0.000	0.047 ± 0.012	0.35 ± 0.11
	EQL (no cos)	0.01 ± 0.000	0.01 ± 0.000	0.011 ± 0.001
	MLP	0.01 ± 0.000	0.084 ± 0.007	0.4 ± 0.021
	SVR	0.01	0.071	0.39



(d)

	interpolation	extrapolation
EQL	0.00042	0.0061 ± 0.0038
MLP	0.002	0.0180 ± 0.0024
SVR	0.00067	0.0057 ± 0.0014

(e)

s	formula
1	$y = 1.28x^2 - 0.183x + 0.026$
2	$y = 1.98x^2 - 1.42x + 0.618 - 1.45\text{sigm}(-3.65x - 0.3)$
3	$y = -0.38z + 2.47\text{sigm}(-2.25z - 2.77) + 0.38$ with $z = \cos(2.32x - 0.08)$
4	$y = 0.221z + 0.42\text{sigm}(0.75z - 3.73)$ with $z = 4.65x^2 - 0.229x$

Figure 5: X-Ray transition energies. (a) Measured data and predicted values by EQL and (b) visualized prediction error for all methods for one train/validation splitting. (c) EQL solutions during model selection in validation error – sparsity space, see Appendix A1 for details. (d) numeric results. Reported are RMS errors with standard deviation for 10 independent train/validation splits. In real units the error is in 100 keV and is well below the difference between neighboring high-Z elements. (e) learned formulas for different sparsities s (lowest dot for each s in (c)).

splits. The data is scaled to lie in $[0, 1]$, i. e. $x = Z/100$ and $y = K\alpha_2/100000$. Model selection is here based on validation error only. The selection for sparsity and validation error only yields the Z^2 relationship. Mini-batch size is 2 here and $T = 50000$ was used. Figure 5 presents the data, the predictions, the learned formulas and the numerical results. EQL and SVR achieve similar performance and MLP is significantly worse. However, EQL also yields interpretable formulas, see Fig. 5(e) that can be used to gain insights into the potential relationship.

Poor extrapolation out of model class — cart-pendulum system

Let us now go beyond our assumptions and consider cases where the true target function is not an element of the hypothesis set.

Consider a pendulum attached to a cart that can move horizontally along a rail but that is attached to a spring damper system, see Fig. 6(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

$$y_3 = \frac{-x_1 - 0.01x_3 + x_4^2 \sin(x_2) + 0.1x_4 \cos(x_2) + 9.81 \sin(x_2) \cos(x_2)}{\sin^2(x_2) + 1}, \tag{13}$$

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

The formulas contain divisions which are not included in our architecture due to their singularities. To incorporate them in a principled manner is left for future work. Thus, the cart-pendulum dynamics is outside the hypothesis class. In this case we **cannot** expect great extrapolation performance and this is confirmed by the experiments. In Fig. 6(b,c) the extrapolation performance is illustrated by slicing through the input space. The near extrapolation performance is still acceptable for both EQL and MLP, but as soon as the training region is left further even the best instances differ considerably from the true values, see also the numeric results in Tab. 4. The SVR is performing poorly also for near extrapolation range. Inspecting the learned expressions we find that the sigmoid functions are rarely used.

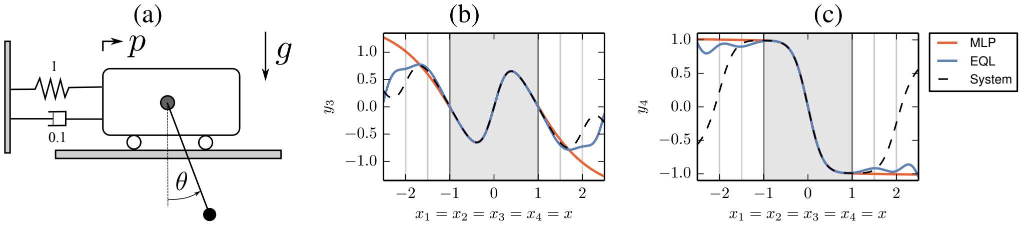


Figure 6: Cart-pendulum system. (a) sketch of the system. The lengths and masses are set to 1, the gravitation constant is 9.81 and the friction constant is 0.01. (b,c) slices of outputs y_3 and y_4 for inputs $x_1 = x_2 = x_3 = x_4 = x$ for the true system equation (Eq. 13), and best EQL, MLP instances.

Table 4: Interpolation and extrapolation performance for *cart-pendulum dynamics*. See Tab. 1 for details. Note that predicting 0 would yield an error of 0.96 on the far test set.

	interpolation	extrapol. (near)	extrapol. (far)
EQL	0.0103 ± 0.0000	0.0621 ± 0.0208	0.180 ± 0.056
MLP	0.0101 ± 0.0000	0.0184 ± 0.0008	0.195 ± 0.006
SVR	0.0118	0.227	0.639

Conclusions

We presented a new network architecture called EQL that can learn analytic expressions that typically occur in equations governing physical, in particular mechanical, systems. The network is fully differentiable, which allows end-to-end training using backpropagation. By sequencing L_1 regularization and fixing L_0 norm we achieve sparse representations with unbiased estimation of factors within the learned equations. We also introduce a model selection procedure specifically designed to select for good extrapolation quality by a multiobjective criterion based on validation error and sparsity. The proposed method is able to learn functional relations and extrapolate them to unseen parts of the data space, as we demonstrate by experiments on synthetic as well as real data. The approach learns concise functional forms that may provide insights into the relationships within the data, as we show on physical measurements of x-ray transition energies.

The optimization problem is nontrivial and has many local minima. We have shown cases where the algorithm is not reliably finding the right equation but instead finds an approximation only, in which case extrapolation may be poor.

If the origin of the data is not in the hypothesis class, i. e. the underlying expression cannot be represented by the network and good extrapolation performance cannot be achieved. Thus it is important to increase the model class by incorporating more base functions which we will address in future work alongside the application to even larger examples. We expect good scaling capabilities to larger systems due to the gradient based optimization. Apart from the extrapolation we also expect improved interpolation results in high-dimensional spaces, where data is less dense.

Acknowledgments

This work was in parts funded by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no. 308036: "Life-long learning of visual scene understanding" (L3ViSU). GM received funding from the People Programme (Marie Curie Actions) in FP7/2007-2013 under REA grant agreement no. 291734.

References

Debasish Basak, Srimanta Pal, and Dipak Chandra Patranabis. Support vector regression. *Neural Information Processing-Letters and Reviews*, 11(10):203–224, 2007.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1-2):151–175, 2010.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2 (1):1–127, 2009.

Yoshua Bengio, Aaron Courville, and Pierre Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(8): 1798–1828, 2013.

Christopher M Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.

David S Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, DTIC Document, 1988.

Richard D Deslattes, Ernest G Kessler Jr, P Indelicato, L De Billy, E Lindroth, and J Anton. X-ray transition energies: new approach to a comprehensive evaluation. *Reviews of Modern Physics*, 75 (1):35, 2003.

Richard Durbin and David E. Rumelhart. Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, 1(1):133–142, March 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.1.133. URL http://dx.doi.org/10.1162/neco.1989.1.1.133.

Lázló Györfi, Wolfgang Härdle, Pascal Sarda, and Philippe Vieu. *Nonparametric curve estimation from time series*, volume 60. Springer, 2013.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *in Proceedings of ICLR*, 2015.

K-R Müller, Alexander J Smola, Gunnar Rätsch, Bernhard Schölkopf, Jens Kohlmorgen, and Vladimir Vapnik. Predicting time series with support vector machines. In *Artificial Neural Networks (ICANN)*, pp. 999–1004. Springer, 1997.

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

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

Hoifung Poon and Pedro M. Domingos. Sum-product networks: A new deep architecture, 2012.

Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009. ISSN 0036-8075. doi: 10.1126/science.1165893. URL http://science.sciencemag.org/content/324/5923/81.

Yoan Shin and Joydeep Ghosh. The pi-sigma network : An efficient higher-order neural network for pattern classification and function approximation. In *in Proceedings of the International Joint Conference on Neural Networks*, pp. 13–18, 1991.

Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

Donald F. Specht. A general regression neural network. *IEEE Transactions on Neural Networks (TNN)*, 2(6):568–576, 1991.

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.

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

Norbert Wiener. *Extrapolation, interpolation, and smoothing of stationary time series*, volume 2. The MIT Press, 1949.

Christopher K. I. Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. The MIT Press, 2006.

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

Appendix

A1: Model selection details

Quantifying sparsity

We actually want a measure of complexity of the formula, however, since it is not clear what is the right choice of a measure, we use the sparsity instead, by counting the number of active/used hidden units denoted by s. For a given network phi we get

$$s(\phi) = \sum_{l=1}^{L} \sum_{i=1}^{k} \Theta(|W_{i,\cdot}^{(l)}| * |W_{\cdot,i}^{(l+1)}| - 0.01)\,, \tag{14}$$

where Θ is the heavyside function and 0.01 is an arbitrary threshold. For the multiplication units the norm of the incoming weights for both inputs are added (omitted to avoid clutter in the formula).

Selection criteria

As stated in the main text, we strive to choose the model that is both simple and has good performance in terms of the validation set. Since both quantities have different scales, we proposed to choose them based on their ranking. Let $r^v(\phi)$ and $r^s(\phi)$ be the ranks of the network ϕ w. r. t. the validation error and sparsity $s(\phi)$respectively, then the network with minimal squared rank norm is selected:

$$\arg\min_{\phi} \left[r^v(\phi)^2 + r^s(\phi)^2 \right] \tag{15}$$

In Fig. 7 the extrapolation performance of all considered networks for the *kin2D-4-end* dataset is visualized in dependence of validation error and the sparsity. It becomes evident that the best performing networks are both sparse and have a low validation error.

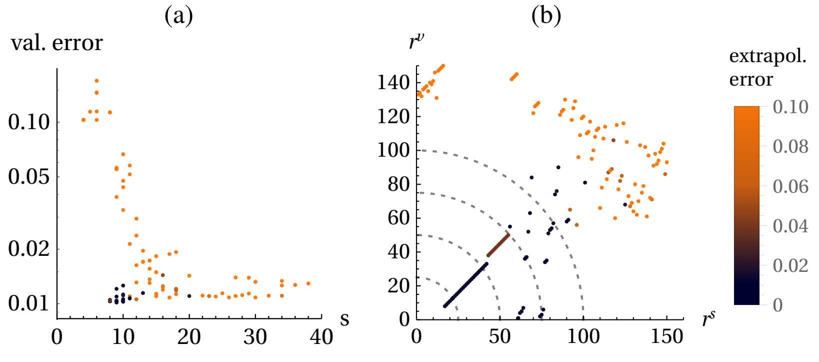


Figure 7: Model selection criteria. (a) extrapolation performance depending on validation error and sparsity (s) for the *kin-4-end* dataset as an illustration. (b) the same as in (a) but in rank-space. Circle arcs indicate the L_2 norm iso-lines.

A2: Dependence on noise and number of data points

In order to understand how the method depends on the amount of noise and the number of datapoints we scan through the two parameters and present the empirical results in Fig. 8. In general the method is robust to noise and as expected, more noise can be compensated by more data.

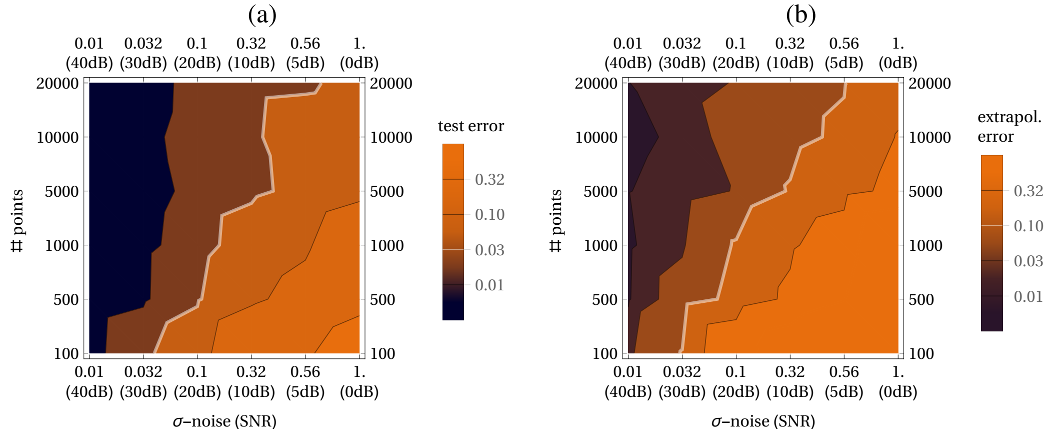


Figure 8: Interpolation performance (a) and extrapolation performance (b) (on the noise-free test set) depending on the number of data points and the size of the additive noise for *kin-4-end* dataset as an illustration. The white line represent an arbitrary threshold below which we consider a successful solution of the interpolation and extrapolation task.