LEARNING THE GOVERNING EQUATIONS OF A DYNAMICAL SYSTEM FROM DATA

Dual Degree Project
V Vasistha Singhal
Department of Chemical Engineering
Indian Institute of Technology Madras
ch16b119@smail.iitm.ac.in

ABSTRACT

Traditionally, the problem of function learning has been treated as a nonlinear regression problem with the independent variables as inputs and dependent variables as outputs. The focus is on identifying a suitable function without attempting to gain insight into the mechanism of the system. To understand and generalize results, finding an interpretable function that can explain the relationship between the inputs and outputs should be the prime goal. Recently[1], it has been shown that specially configured neural networks are capable of learning nonlinear functions that describe the behaviour of simple dynamical systems. The purpose of this project is to explore whether such neural networks can be used to learn equations that describe the dynamical behaviour of unit operations encountered in chemical engineering, such as reactors and separators.

Introduction

The quality of a regression model is typically measured by its performance on previously unseen data from the same distribution as the training set. The generalization of a model essentially boils down to interpolation if the training data is sufficiently dense. However, when working with real-world data, e.g. controlling a robotic arm or a dynamic CSTR system, interpolation might not be sufficient. It is possible that the future data lies outside of the training domain. For robustness, it is desirable in such a case to have a model that continues to make good predictions even outside the domain of the training set. In this work, we aim at discovering the underlying relationship between the input and the output to obtain an interpretable function in a concise form. Our objective is not only to learn the parameters of the base functions, but also to identify their composition.

The task of finding equations for observations is also known as symbolic regression. In symbolic regression, a space of mathematical expressions is searched in order to find the model that best fits a given dataset, both in terms of accuracy and simplicity. Due to the exponential search space, the computational complexity becomes prohibitive for larger expressions and high-dimensional problems.

Genetic programming (GP), as an evolutionary computing technique, is one of the most popular methods for symbolic regression in recent years. The main idea behind GP is to apply Darwin's theory of natural evolution to the artificial world of computers and modelling. Theoretically, GP can get accurate results provided that the computation time is long enough. However, due to stochasticity, it is difficult to get repeated results with GP. In addition, convergence of GP is too slow in some cases. Despite these limitations, GP has played a crucial role in some successful symbolic regression algorithms.

Icke and Bongard[2] have proposed a hybrid algorithm which synergises the Fast Function Extraction (FFX) algorithm with a purely GP-based symbolic regression algorithm to solve symbolic regression problems more accurately and efficiently in comparison to either technique alone. The FFX algorithm performs feature extraction by employing a non-linear basis function expansion method that creates new features via unary and binary interactions of the input variables. These features are then passed to the GP-based symbolic regression algorithm for model building which returns multiple expressions containing different numbers of basis functions. The best model is then selected based on the error on validation data and expression complexity.

In the work by Rad, Feng and Iba[3], an extended version of GP called Kaizen programming is combined with Relevance Vector Machines (RVMs). The proposed approach to symbolic regression involves a sparse Bayesian kernel method which integrates a GP-based search of tree structures and a Bayesian parameter estimation employing automatic relevance determination. The non-determinacy of GP under random search is overcome by Kaizen programming which performs automatic feature engineering to build models from features generated by GP. Once the candidate functions are constructed, a sequential sparse Bayesian learning algorithm is used to solve an optimization problem which returns the desired parameters of the model.

A common theme in these studies is the complete traversal of a GP tree to yield all possible subtree functions (brute force algorithm) and then narrowing down the candidate functions using an optimization algorithm. In addition to these GP-based algorithms, other novel algorithms have also been employed in recent times to approximate governing equations. Chen et al.[4] proposed a non-evolutionary algorithm called Elite Bases Regression, which preserves only elite bases which are updated iteratively and help span the regression model. Qin et al.[5] have proposed a recurrent ResNet and a recursive ResNet for this task. The ResNet block that was initially proposed for image analysis was shown to play the role of a numerical integrator in time.

We now explore the use of artificial neural networks (ANNs) for symbolic regression. These networks are a collection of connected nodes called artificial neurons which model biological neurons. Every neuron is connected to one or more neurons and can transmit and receive signals. The signal at any connection is a real number and non-linear activation functions, like logistic, hyperbolic tangent and sigmoid are used to compute the output of each neuron. Neurons are arranged into layers, and a network consists of one or more layers. The signals travel from the first layer, the input layer, to the last layer, the output layer.

ANNs are trained by processing labelled examples, each of which is comprised of a known input and result. The associations between the input and the result are learned as weights in the connections and are stored in the structure of the network itself. These weights are adjusted using an error value which is computed using the difference between a processed output from the network and a known target output. As the weights are adjusted, the network produces an output which is increasingly closer to the target output.

Neural networks have been used for regression tasks, with the primary focus being the accurate prediction of the outputs. The independent variables are used as inputs to the network and the dependent variables are used as outputs. One or more hidden layers are present in between the input and output layers and these determine the degree of non-linearity of the relation between the inputs and outputs. The activation functions used in these networks, for instance, hyperbolic tangent and ReLU, do not occur in physical systems and as a result, these networks are not suitable for identifying the governing equations of dynamical systems. As a consequence, these networks are expected to perform poorly on test data that lies outside of the training domain (poor extrapolation capability). A general architecture for such a network is shown below:

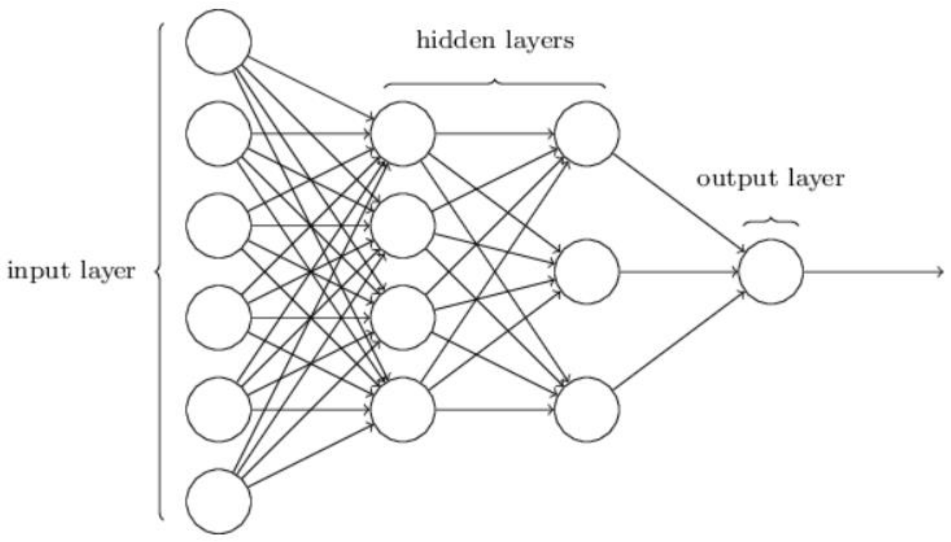

Figure 1: General architecture of a neural network for function approximation

The main interest of this paper lies on extrapolation in the context of learning the dynamics of physical systems that can be described by real-valued analytical expressions. Martius and Lampert[1] have proposed a new type of neural network, called the Equation Learner (EQL), that can learn analytical expressions and extrapolate to unseen domains. We wish to assess their approach and reproduce the results for simple systems like the pendulum. The present report presents a summary of their work and a critical evaluation of their approach.

Architecture of the EQL

The network architecture proposed in the paper by Martius and Lampert[1] is shown in Figure 2 below. It differs from typical feed-forward neural networks in two main aspects: multiplication units and possibility of sine and cosine functions as non-linearities. The multiplication units in each layer provide a unique ability to multiply two input values. Each layer in the EQL is comprised of u unary units and v pairs of binary units. The unary units are similar to those found in traditional artificial neural networks but have four possible base functions: identity, sine, cosine and sigmoid. Each pair of binary units takes two inputs and produces their product as the output.

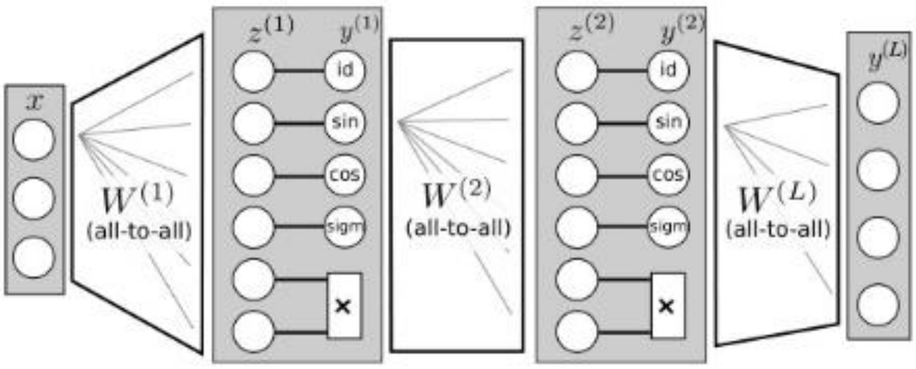

Figure 2: Network architecture of the proposed Equation Learner (EQL) for 3 layers and one neuron per type (u=4, v=1).

Description of System

A general nonlinear dynamical system can be represented using the following equation.

$$\dot{y} = f(y, u)$$

u and y represent the inputs and outputs of the system and f is a nonlinear function that describes the dynamics of the system. In our application, we use a neural network to learn the underlying function f of the system. For this purpose, the measured values of y and u are used as inputs to the neural network and the derivatives of the response variable y are used as outputs from the neural network. However, it is important to note that the derivatives of y are not directly measured and need to be computed from the sampled data for y using numerical methods.

Network Training

The EQL is fully differentiable in its free parameters $\{W^{(1)}, W^{(2)}, \ldots, W^{(L)}, b^{(1)}, b^{(2)}, \ldots, b^{(L)}\}$ and this allows training using end-to-end backpropagation. A Lasso-like objective function is adopted:

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

that is, a linear combination of L_2 loss and L_1 regularization. A stochastic gradient descent algorithm with mini-batches is used for calculating the updates:

$$\theta_{t+1} = \theta_t + \mathrm{Adam}\left(\frac{d\mathcal{L}(D)}{d\theta}, \alpha\right)$$

The role of L_1 regularization is to encourage networks with sparse connections, matching the intuition that a typical formula describing a system contains only a small number of terms. However, in a non-convex setting where local minima are likely to occur, this type of regularization can have undesirable effects: during optimization the weights hardly ever change their sign. Therefore, a hybrid regularization strategy is followed. No regularization is used at the beginning of training such that the parameters can freely vary and reach reasonable starting points. The regularization is switched on after a stipulated amount of epochs which has a sparsifying effect on the model. Finally, for the last steps of training, L_1 regularization is disabled but the same L_0 norm of the weights in enforced. This is achieved by keeping the weights that are close to zero at zero, i.e. say, 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. The equation learned by the network is transformed into a symbolic expression using a python computational algebra library (SymPy).

Experimental Evaluation

The ability of EQL to learn physically inspired models is showcased using synthesised data from a simple pendulum system. The pendulum's behaviour is governed by two ordinary differential equations:

$$\dot{x}_1 = x_2 \qquad \text{and} \qquad \dot{x}_2 = -g \sin x_1$$

where x_1 and x_2 are the angle of the pole in radians and angular velocity respectively and g is the gravitation constant. These are divided by g in order to balance the output scales and provide the two output values of the model as

$$y_1 = \frac{x_2}{g} \qquad \text{and} \qquad y_2 = -\sin x_1$$

Martius and Lampert generated training and test data for this system manually by sampling points uniformly in a hypercube. The training and test sets are as follows:

- Training set: 1000 points in $[-\boldsymbol{h},\boldsymbol{h}] \times [-\boldsymbol{h},\boldsymbol{h}]$ for h=2.
  x_1 and x_2 are sampled uniformly from this interval and are the inputs to the neural network.
- Test sets (1000 points each)
  The data for the test sets is chosen from intervals that are outside the training domain as follows:
  - Interpolation: from the same distribution as training set
  - Extrapolation (near): $[-\mathbf{1.5}\boldsymbol{h},\mathbf{1.5}\boldsymbol{h}] \times [-\mathbf{1.5}\boldsymbol{h},\mathbf{1.5}\boldsymbol{h}] \setminus [-\boldsymbol{h},\boldsymbol{h}] \times [-\boldsymbol{h},\boldsymbol{h}]$
  - Extrapolation (far): $[-\mathbf{2}\boldsymbol{h},\mathbf{2}\boldsymbol{h}] \times [-\mathbf{2}\boldsymbol{h},\mathbf{2}\boldsymbol{h}] \setminus [-\boldsymbol{h},\boldsymbol{h}] \times [-\boldsymbol{h},\boldsymbol{h}]$

The target values (outputs) of the neural network are computed directly using the known analytical expressions (y_1 and y_2 above). Now, a 2-layer EQL (one hidden layer), with 4 unary units is trained with all the weights initialized from a normal distribution. Hyperparameter tuning is carried out for the regularization parameter and the number of binary nodes. The network is trained for 10000 epochs to ensure convergence. These epochs are split over three phases (25%, 75% and 5% of epochs respectively) using a hybrid regularization strategy as mentioned in the Network Training section. The performance of the EQL is compared to a multi-layer perceptron (MLP) with standard tanh activation functions and a Support Vector Regression (SVR) model. The mean and standard deviations of the RMSE from 10 different runs and initialisations are shown:

	Interpolation	Extrapolation (near)	Extrapolation (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

*Results from the paper by Martius and Lampert[1]

As expected, all models are able to interpolate well. For extrapolation however, the performance differs. 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.

Limitations

The aforementioned analysis by Martius and Lampert, though promising, has certain limitations which need to be addressed. Firstly, the method used for data generation is non-standard and is therefore not representative of a real-world experimental setting. For dynamical systems, a sequence (time-series) of outputs is typically obtained experimentally for a given initial condition and input sequence, based on which the system is identified. Secondly, the derivatives of the outputs are not directly available and have to be computed from the given outputs numerically, if required. And finally, the outputs are inevitably corrupted by noise. We examine whether the method proposed by Martius and Lampert is able to learn the governing equations of a dynamical system under a realistic data generation setting.

Realistic Data Generation

Let us continue with the simple pendulum system. In a field experiment, the pendulum would be released from an initial point and be allowed to oscillate while measurements of the angle of the pole and the angular velocity are taken at regular intervals. To simulate this realistic setting, we use the Runge-Kutta method to integrate the differential equations from various starting points. The length of the pendulum is fixed as $l = 9.8m$ so that the angular frequency becomes 1 rad/s ($w = \sqrt{l/g}$). The time period is now 2π seconds and therefore the frequency of the system is 0.16Hz. The sampling interval is chosen as 0.02s or 50Hz. Zero-mean Gaussian noise is added to mimic a real life setting. Various signal-to-noise ratios (100, 50, 10) are chosen for this purpose.

Now that we have generated the input data for the neural network, we need to evaluate the derivatives of the inputs which will act as target values (output values) for the neural network. Forward finite differencing is used to obtain the derivatives of the input data. The sequence followed for data generation is shown in Figure 3.

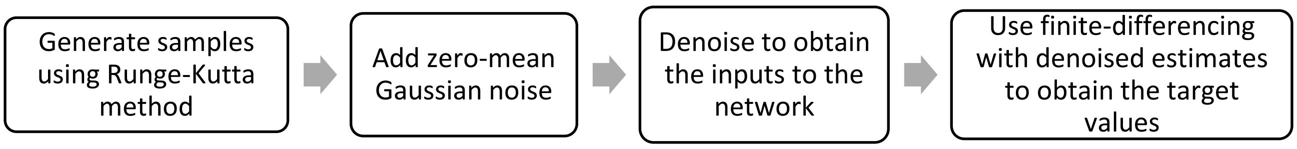
Generate samples using Runge-Kutta method
Add zero-mean Gaussian noise
Denoise to obtain the inputs to the network
Use finite-differencing with denoised estimates to obtain the target values

Figure 3: Sequence of steps followed for the generation of data (pendulum system).

Denoising Filters for Noisy Data

The network is expected to perform poorly if trained on noisy data. Hence, the training data, i.e the measurements of the angle of the pole and the angular velocity, must be denoised before they are used by the neural network. Two denoising filters, namely, the Savitzky-Golay (S-G) and the Hodrick-Prescott (H-P) filters are explored for this purpose. In Savitzky-Golay filters, successive sub-sets of adjacent points are fitted with a low-degree polynomial by method of linear least squares and the denoising is done using a 1-D convolution of the data with the obtained coefficients. This technique has two hyperparameters to be tuned: the window-length (w), which controls the number of adjacent points in a sub-set and the order of the polynomial (n) to be fitted to a particular sub-set of data points[6]. In Hodrick-Prescott filters[7], the trend estimate x_t (y_t is the original noisy signal) is chosen to minimize the weighted sum objective function below. λ is the hyperparameter to be tuned here.

$$\frac{1}{2}\sum_{t=1}^{n}(y_t - x_t)^2 + \lambda\sum_{t=2}^{n-1}(x_{t-1} - 2x_t + x_{t+1})^2$$

To compare the denoising performance of both these filters, 5000 data points are generated from the initial point $x_1=\pi/4$, $x_2=0$ using the methodology discussed before. Zero-mean Gaussian noise is then added to the signal such that a signal-to-noise ratio of 10 is maintained. Plots of x_1 vs x_2 are obtained for the best values of the hyperparameters, before and after denoising as shown in Figure 4.

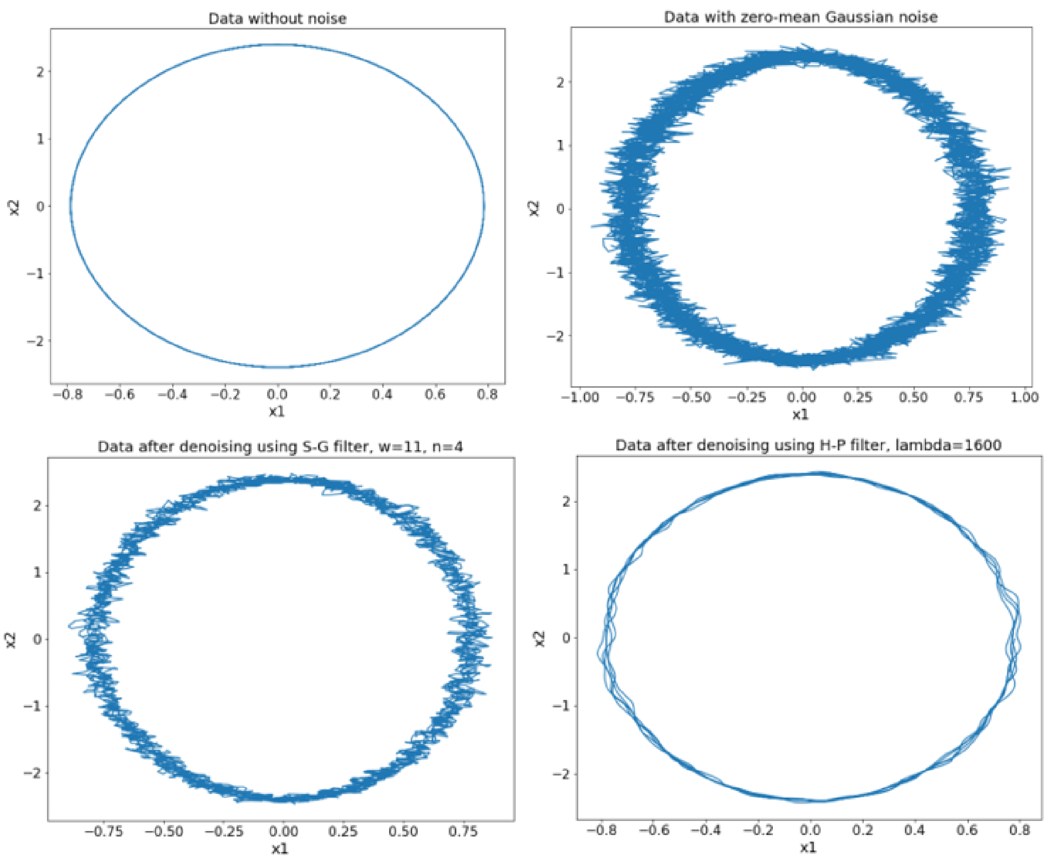

Figure 4: Visualisation of denoising using S-G and H-P filters.

It is clear that the Savitzky-Golay filter is unable to obtain smooth estimates, but the Hodrick-Prescott filter does so very well. The denoised data looks very similar to the data before the addition of the noise. We expect the model to perform better when using the denoised data from the H-P filter.

Training Results

5000 samples each (noisy, signal-to-noise ratio=10) from distinct initial points (x_o) are generated. These samples are denoised using both the filters as explained. The denoised estimates are then used to calculate the derivatives which will be used as the target outputs for the neural network. A 2-layer EQL model, with one hidden layer consisting of 4 unary units (sine, cosine, sigmoid and identity base functions) and one pair of binary units is trained for 5000 epochs. The model is trained on data obtained from different initial points to assess its dependence on the domain of the training data. 20% of the generated data was used as test data.

Results for data denoised using S-G filter

w	n	x_0	Equation Learned	Train RMSE	Test RMSE
11	4	(π/3,0)	$[0.102x_2, -0.933\sin(1.098x_1)]$	1.45	1.54
11	4	(π/3,0)	$[0.102x_2, -0.269\sin(0.281x_1) - 0.924\sin(0.994x_1)]$	1.10	1.12
11	4	(π/3,0)	$[0.102x_2, -0.953x_1]$	0.87	0.88
11	4	(π/3,0)	$[0.337\sin(0.313x_2), -0.953x_1]$	0.72	0.73

Results for data denoised using H-P filter

x_0	Equation Learned	Train RMSE	Test RMSE
(π/3,0)	$[0.102x_2, -1.008\sin(0.990x_1)]$	0.063	0.064
(π/3,0)	$[0.102x_2, -1.036\sin(0.963x_1)]$	0.048	0.055
(π/3,0)	$[0.101x_2, -1.003\sin(0.991x_1)]$	0.038	0.040
(π/3,0)	$[0.101x_2, -1.018\sin(0.980x_1)]$	0.032	0.033

The significantly lower train and test errors for the case when the H-P filter is used suggest that the model has learnt a very good approximation of the true equation. This is indeed the case since the true equation is learnt irrespective of the training domain. On the other hand, for the S-G filter, the success of the model in learning the correct underlying equation depends on the initial point chosen. For a bigger training domain, i.e x_0 = (π/3,0), the model is successful but fails to learn the correct underlying equations when a smaller training domain is used to generate the data. On the other hand, due to significantly better smoothness of estimates, the model performs well in all cases when the H-P filter is used for denoising. In addition, the estimated equations and their coefficients are also more consistent across different initializations for the latter case.

Extension to complex systems

The equation learning network has been used to learn the governing equations of a simple pendulum system. Many dynamic systems are governed by equations which contain more involved analytical expressions. We wish to demonstrate the ability of the EQL to learn complex expressions with various algebraic terms. For this purpose, we consider a continuously stirred tank reactor (CSTR) system[8].

Let us take a classical CSTR with an exothermic, irreversible reaction A → B. The temperature of the coolant (T_c) is manipulated to control the temperature of the reactor (T). This system is governed by the following equations:

$$\dot{C_A} = \frac{q\left(C_{Af} - C_A\right)}{V} - k_0 e^{\left(\frac{-E}{RT}\right)} C_A$$

$$\dot{T} = \frac{q\left(T_f - T\right)}{V} - \frac{\Delta H}{\rho C_p} k_0 e^{\left(\frac{-E}{RT}\right)} C_A + \frac{UA}{V\rho C_p}\left(T_c - T\right)$$

where q is the inlet flow rate, V is the volume of reactor, C_A is the concentration of the reactant, k_0 is the rate constant, E is the activation energy, R is the gas constant, ΔH is the heat of the reaction, ρ is the reaction mixture density, C_P is the specific heat of mixture, U is the heat transfer coefficient of the reactor and A is the area of the reactor.

The challenge of this problem is to incorporate terms like the exponential of the reciprocal of the measured variable into the model while ensuring convergence and accuracy. This can possibly be achieved by altering the set of hypothesis functions of the unary units in the architecture.

Modified architecture of the EQL

The EQL architecture proposed by Martius and Lampert discussed above has one major shortcoming. It is unable to represent divisions, thereby severely limiting the physical systems to which it can be applied. Sahoo et al.[9] have proposed an improved network for equation learning called EQL_DIV that overcomes this limitation. The modified architecture, which contains division units that calculate a/b is shown below.

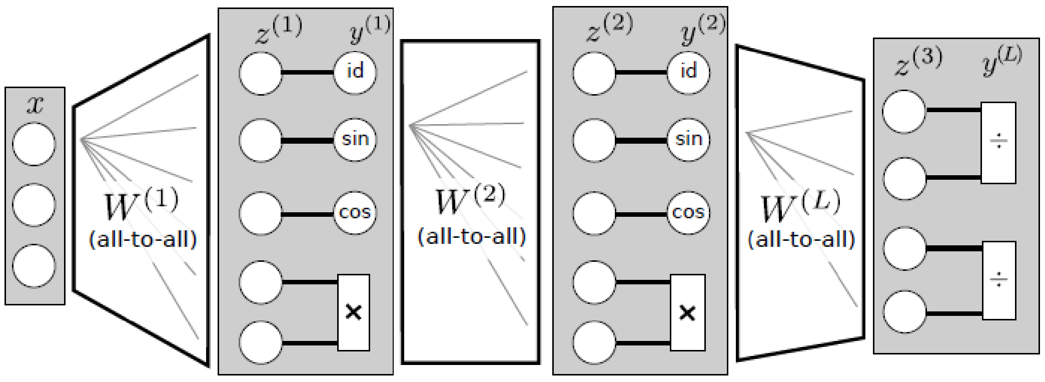

Figure 5: Network architecture of the proposed improved Equation Learner EQL_DIV for 3 layers (L=3) and one neuron per type (u=3, v=1), with the base function set S-4. The new division operations are placed in the final layer.

Regularized Division and Penalty Term

The inclusion of division units 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 this problem of divergence, it is important to realize that one does not encounter data at the pole for any real system 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, divisions are used only in the output layer.

The last layer of the EQL computes the regression values by a linear read-out represented by the equation

$$y^{[L]} = W^{[L]} y^{[L-1]} + w_0^{[L]}$$

This is replaced by the following equation for the modified EQL_DIV,

$$y^{[L]} = \{h_1^{\theta}(z_1^{[L]}, z_2^{[L]}), \ldots\ldots, h_m^{\theta}(z_{2m}^{[L]}, z_{2m+1}^{[L]})\}$$

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

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

$\theta \geq 0$ is a threshold. Using $h^{\theta} = 0$ as the value when the denominator is below θ sets the gradient to zero, avoiding any misleading parameter updates. Therefore, the discontinuity plays no role in practice. In addition, to steer the network away from negative values of the denominator, a cost term is added to the objective function which penalizes "forbidden" inputs to each division unit:

$$p^{\theta}(b) = max(\theta - \mathrm{b}, 0)$$

where θ is the threshold and b is the denominator input to the division-activation function. The global penalty term is then

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

where $z_{2j}^{(L)}(x_i)$ is the denominator of the division unit j for the input x_i.

Network Training

The modified EQL_DIV is also fully differentiable in its free parameters like the EQL, which allows us to train the model in an end-to-end fashion using backpropagation. The objective is Lasso-like

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

that is, a linear combination of L_2 loss and L_1 regularization extended by the penalty term for small and negative denominators. Like in the case of the EQL, a stochastic gradient descent algorithm is used with mini-batches and the same hybrid regularization scheme is implemented.

Results and Discussion

To showcase the improved learning capabilities of the EQL_DIV model over the EQL, an attempt is made to learn some non-trivial functions. The task at hand is to identify concise equations from a pool of selected activation functions (the base functions of the unary units in the network). Because of the strong non-convexity of this problem, the gradient-based optimization process may get stuck in local minima and therefore be unable to select the true governing equation as the best approximation. As an attempt to circumvent this problem, 5 independent runs with different sets of base functions in the unary units of the network are made. The different sets of base functions are as follows:

Set name	**Base functions**
S-1	Identity
S-2	Identity, exponential
S-3	Identity, reciprocal
S-4	Identity, sine, cosine
S-5	Identity, exponential, reciprocal

Inverse function:

Training set: 500 points sampled uniformly from [0.1,1) and 2000 points sampled uniformly from [1,50], total of 2500 data points.

Test set: 100 points sampled uniformly from [51,65]

For each value of the independent variable x, the target values were evaluated as 1/x and these were used as input to the neural network. A 2-layer EQL_DIV model, with one hidden layer consisting of one of the five base function sets for the unary units and three pairs of binary units was trained for 3000 epochs. The results from independent runs with the different base function sets are shown.

Base set	Equation Learned	Train RMSE	Test RMSE
S-1	$\frac{0.3031}{0.3031x + 4.74 \times 10^{-9}}$	1.05 x 10^{-9}	4.7 x 10^{-8}
S-2	$\frac{-1.64(0.0033x)(0.0025x + 0.528) + 0.326}{-x(0.0025x + 0.528)}$	0.014	0.032
S-3	$\frac{-0.7119}{-0.7119x + 5.035 \times 10^{-9}}$	4.5 x 10^{-8}	1.49 x 10^{-6}
S-4	$\frac{0.3604}{0.3604x + 7.45 \times 10^{-9}}$	1.77 x 10^{-9}	5.65 x 10^{-8}
S-5	$\frac{0.3643}{0.3643x - 1.28 \times 10^{-8}}$	4.42 x 10^{-8}	1.42 x 10^{-6}

It is observed that for four of the five base function sets, the test RMSE is sufficiently low, which suggests that the network has learned, at the very least, a very accurate approximation of the underlying equation. On simplification of the learned expressions, we find that the true underlying equation, $\frac{1}{x}$ has been successfully learnt. It is worth noting that for a smaller training set domain, the network learns a different equation for the data. When the training data is sampled only from [1,50], the learned equation is of the form $\frac{1}{ax^2+bx+c}$, which is not the true underlying equation. This suggests a dependence on the domain of the training data.

Exponential function:

Training set: 1000 points sampled uniformly from [-5,30]

Test set: 100 points sampled uniformly from [-6,-5] and [30,32], total of 200 data points

For each value of x, the target values were evaluated as e^{-x}, and these were used as inputs to the neural network. A 2-layer EQL_DIV model with specifications as in the previous case was trained for 3000 epochs. The results from independent runs with the different base function sets are shown.

Base set	Equation Learned	Train RMSE	Test RMSE
S-1	$\frac{-1.4x+0.95(0.34x+0.88)(0.19x+0.53)+0.43}{0.12x+0.28(0.34x+0.89)(0.19x+0.53)+0.58}$	0.75	4.2
S-2	$\frac{-x+2.58(0.14x+0.16)(0.15x+0.1)+0.96}{0.1x+0.29(0.14x+0.16)(0.15x+0.1)+0.46}$	1.98	20
S-3	$\frac{-1.54x+1.36(0.21x+0.48)(0.25x+0.66)+0.63}{0.14x+0.36(0.21x+0.48)(0.25x+0.66)+0.66}$	0.85	9.82
S-4	$\frac{-2.54x+1.36(0.24x+0.81)(0.31x+1.27)+0.89}{0.18x+0.69(0.24x+0.81)(0.31x+1.27)+0.94}$	0.43	34.6
S-5	$\frac{-2.28x+0.86(0.31x+0.53)(0.42x+0.61)+1.12}{0.28x+0.22(0.31x+0.53)(0.42x+061)+1.16}$	0.84	3.9

The high values of the test RMSE suggest that the network has been unable to learn a good approximation of the underlying equation. This is indeed the case since the model has learnt a ratio of polynomials as the underlying equation in all the cases. A possible reason for this is that the optimisation process gets stuck on a local optimum and gradient-descent is unable to drive the parameters in the right direction. We therefore attempt to learn an approximation of the exponential, $e^{-x} = \frac{1-0.5x}{1+0.5x}$

Pade approximation of exponential:

Training set: 500 points sampled uniformly from [1,5]

Test set: 100 points sampled uniformly from [6,8]

For each value of x, the target values were evaluated as $\frac{1-0.5x}{1+0.5x}$, and these were used as inputs to the neural network. The results from the five independent runs are shown below.

Base set	Equation Learned	RMSE (train)	RMSE (test)
S-1	$\frac{0.0150-0.0075x}{0.015+0.0075x}$	1.14×10^{-5}	1.17×10^{-5}
S-2	$\frac{-0.007x+0.014}{0.007x+0.014}$	7.73×10^{-5}	2.18×10^{-4}
S-3	$\frac{0.014-0.0035x^2}{0.0058x^2+0.029}$	0.0074	0.054
S-4	$\frac{0.0144-0.0072x}{0.0072x+0.0144}$	0.0015	0.0019
S-5	$\frac{0.0146-0.0073x}{0.0073x+0.0146}$	7.61×10^{-8}	9.03×10^{-8}

It is observed that for three of the five independent runs, the test RMSE is sufficiently low, which suggests that the network has learnt a good approximation of the underlying equation. Simplification of the learned equations during the runs with the base function sets S-1, S-2 and S-5 yields the true equation.

Conclusion

In this work, we describe the architecture of the novel neural network for learning equations proposed by Martius and Lampert[1]. It differs from typical feed-forward neural networks in two main aspects – multiplication units and sine and cosine as non-linearities. Despite its impressive extrapolating capabilities, the model is found to have limitations – non-standard data generation method, evaluation of derivatives and noise handling in inputs.

Numerical methods like the Runge-Kutta and finite differencing were used to address these issues. In addition, denoising filters are explored to enable the model to handle noisy training data. The EQL model is unable to learn complex functions such as exponential and inverse functions. So, to enhance the learning scope of the EQL model, division units are introduced in the final layer. The modified architecture is shown to be successful in learning reciprocal functions. It is also shown that learning the Pade approximation to the exponential function is easier than learning the exponential function. The inability of gradient-descent to drive the algorithm in the right direction after getting stuck on a local optimum is cited as a possible reason for this.

The EQL models have shown some limitations – dependence on training domain in some cases, tendency to get stuck on local optima and a need for independent runs with different base function sets for the EQL_DIV model. Nevertheless, the EQL architecture has proven successful in learning functions like sine, cosine, reciprocal and approximation to the exponential with a reasonable amount of training data and compute power. The task of enabling the model to learn the model of a CSTR, which involves a combination of these functions is left as a direction for future work.

References

[1] G. Martius and C. H. Lampert, “Extrapolation and learning equations,” arXiv:1610.02995 [cs], Oct. 2016, Available: http://arxiv.org/abs/1610.02995

[2] I. Icke and J. C. Bongard, “Improving genetic programming based symbolic regression using deterministic machine learning,” in 2013 IEEE Congress on Evolutionary Computation, Jun. 2013, pp. 1763–1770. doi: 10.1109/CEC.2013.6557774.

[3] H. I. Rad, J. Feng, and H. Iba, “GP-RVM: Genetic Programming-based Symbolic Regression Using Relevance Vector Machine,” arXiv:1806.02502 [cs], Aug. 2018, Available: http://arxiv.org/abs/1806.02502

[4] C. Chen, C. Luo, and Z. Jiang, “Elite Bases Regression: A Real-time Algorithm for Symbolic Regression,” arXiv:1704.07313 [cs], May 2017, Available: http://arxiv.org/abs/1704.07313

[5] T. Qin, K. Wu, and D. Xiu, “Data Driven Governing Equations Approximation Using Deep Neural Networks,” Journal of Computational Physics, vol. 395, pp. 620–635, Oct. 2019, doi: 10.1016/j.jcp.2019.06.042.

[6] M. Sadeghi and F. Behnia, “Optimum window length of Savitzky-Golay filters with arbitrary order,” arXiv:1808.10489 [eess], Aug. 2018, Available: http://arxiv.org/abs/1808.10489

[7] S.-J. Kim, K. Koh, S. Boyd, and D. Gorinevsky, “ℓ_1 Trend Filtering,” SIAM Rev., vol. 51, no. 2, pp. 339–360, May 2009, doi: 10.1137/070690274.

[8] M. A. Henson and D. E. Seborg, Eds., Nonlinear process control. Upper Saddle River, N.J: Prentice Hall PTR, 1997.

[9] S. S. Sahoo, C. H. Lampert, and G. Martius, “Learning Equations for Extrapolation and Control,” arXiv:1806.07259 [cs, stat], Jun. 2018, Available: http://arxiv.org/abs/1806.07259