Learning the governing equations of a dynamical system from data

Dual Degree Project Presentation

V Vasistha Singhal
Department of Chemical Engineering
Guide: Dr. Shankar Narasimhan

Introduction

- Function learning – **regression problem** with the independent variables – inputs, dependent variables - outputs
- Focus - identify some prediction function with small expected error, **no insight** into the mechanism of the system
- Quality of model – measured by interpolating capabilities
- Expected to perform poorly on data outside training domain
- Future data – might lie outside training domain, hence **extrapolation becomes important**!

Symbolic Regression

- Finding equations for observations – symbolic regression
- A space of mathematical expressions is searched – best model in terms of **accuracy and simplicity** chosen
- Genetic programming – evolutionary computing technique based on Darwin's theory of evolution, used widely for symbolic regression
- Icke et al. and Rad et al. have proposed GP-based SR algorithms that have been successful in solving problems accurately
- Common theme – complete traversal of a GP tree to yield all possible subtree functions, eliminating candidates using an optimization algorithm

Neural Networks

- Neural networks – collection of interconnected nodes or neurons
- Activation functions – tanh and ReLU, **not found in physical systems**
- Martius and Lampert proposed a novel equation learning network (EQL)
- Specially configured network – **learns nonlinear functions** governing dynamical systems

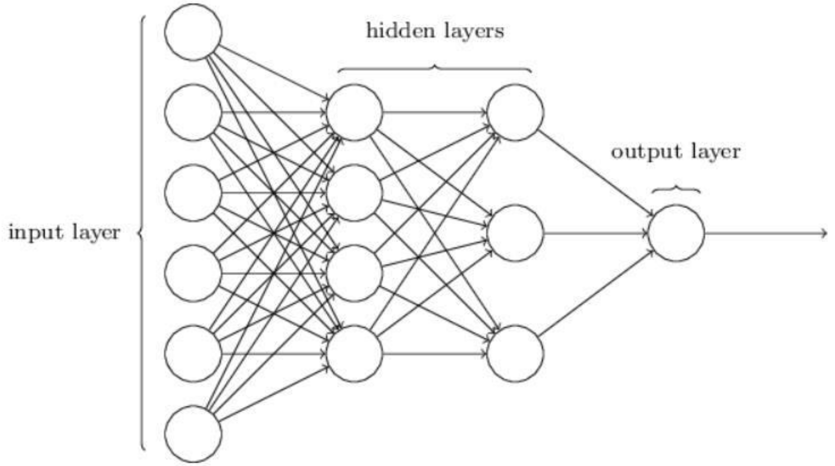

Figure 1: General architecture of a neural network for function approximation

Objectives

- To critically evaluate and assess the approach by Martius and Lampert
- To reproduce the results obtained by them for simple dynamical systems
- Can EQL be used to learn the analytical expressions describing complex nonlinear dynamical systems?

Description of system

- A general nonlinear dynamical system is shown below

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

- u, y - inputs and outputs of the system, f – nonlinear function
- Measured values of u and y – inputs to neural network
- Derivatives of y – target values for the neural network
- Note: Derivatives of y have to be computed numerically

Architecture of the EQL

- Key features – **multiplication units**, **sine and cosine** as non-linearities
- u unary units, v binary units in each layer
- 4 possible base functions – identity, sine, cosine, sigmoid
- Binary units take 2 inputs and produce their product as the output

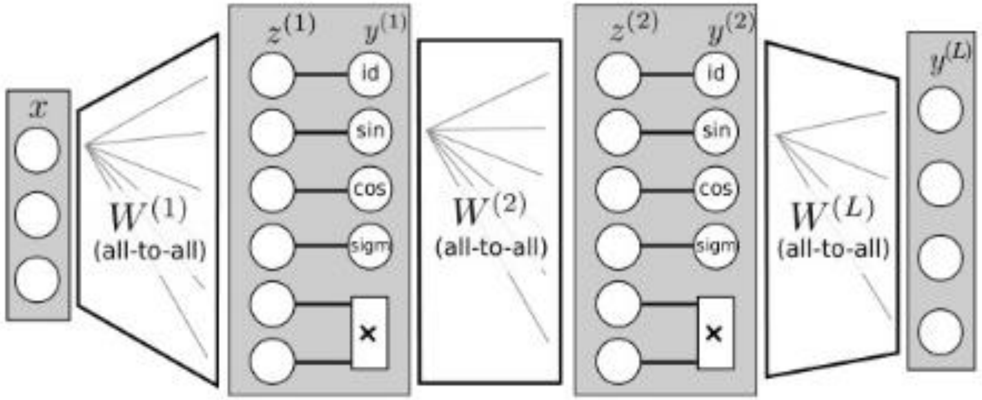

Figure 2: Network architecture of the proposed Equation Learner for 3 layers and one neuron per type (u=4, v=1).

Network Training

- EQL - **fully differentiable**, backpropagation training. A lasso-like objective is adopted:

$$L(D) = \frac{1}{N}\sum_{i=1}^{|D|} \|\varphi(xi) - yi\|^2 + \lambda \sum_{l=1}^{L} |W^l|_1$$

- Stochastic gradient descent algorithm with mini-batches for calculating the updates is used:

$$\Theta_{t+1} = \Theta_t + \text{Adam}\left(\frac{dL(D)}{d\Theta}, \alpha\right)$$

- **Role of $\boldsymbol{L_1}$** regularisation is to encourage networks with **sparse connections**, matching the intuition that a typical formula describing a system contains only a **small number of terms**

Experimental Evaluation

- Physically inspired model - **synthetic pendulum data**

- x_1 and x_2 are the angle of the pole in radians and angular velocity respectively

$$\dot{x}_1 = \mathrm{x}_2 \quad \text{and} \quad \dot{x}_2 = -\mathrm{g}\sin x_1$$

- Divide by g in order to balance the output scales and provide the two output values, (also the target equations)

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

Data generation

Training data: 1000 points sampled uniformly in the hypercube **[-h, h] x [-h, h]** for h=2.

Test sets (1000 points each):

- Interpolation: from the same distribution as training set
- Extrapolation(near): **[-1.5h, 1.5h] x [-1.5h, 1.5h]** \ **[-h, h] x [-h, h]**
- Extrapolation(far): **[-2h, 2h] x [-2h, 2h]** \ **[-h, h] x [-h, h]**

The target values (derivatives) are computed simply using their analytical expressions

Training and Performance

- **2-layer EQL -** trained with all weights initialized from a normal distribution
- Hyperparameter tuning – regularization and the number of binary nodes
- To ensure **convergence**, the model is trained for **10000 epochs**
- EQL compared with multi-layer perceptron (MLP) and Support Vector Regression (SVR). **EQL extrapolates well**, both SVR and MLP fail. The mean and standard deviations of the RMSE from 10 different runs and different 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

Limitations

- Method of data generation - **non-standard**, not representative of a **real-world** experimental setting
- Derivatives not directly available – to be computed numerically
- Data is inevitably corrupted by **noise** and this needs to be addressed
- Now, we wish to examine whether the EQL is successful with data generated in a more realistic manner

Realistic Data Generation

- Field experiment - pendulum system **released from an initial point**, allowed to oscillate while measurements of the angle of the pole and the angular velocity are taken at regular intervals
- Simulation **- Runge Kutta** method, integrate the differential equations from different initial points
- $l = 9.8m$, the angular frequency of the system becomes 1 rad/s ($w = \sqrt{\frac{l}{g}}$)
- The sampling interval is chosen as h=0.02s
- Derivatives – numerically calculated using **forward finite differencing**

Denoising Filters for Noisy Data

- **Zero-mean Gaussian noise** of varying standard deviations is added in order to have datasets with **varying signal-to-noise ratios**
- Two different denoising filters were explored – **Savitzky-Golay (S-G), Hodrick-Prescott (H-P)**
- In S-G filters, **successive sub-sets** of adjacent data 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
- In H-P filtering, the trend estimate x_t is chosen to **minimize** the weighted sum **objective function**: (y_t is the original noisy signal)

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

Visualisation of denoising performance

- **5000 data points -** generated using the initial point $x_1 = \pi/4$, $x_2=0$, zero-mean Gaussian noise added, **SNR = 10**
- For S-G filter, the hyperparameters to be tuned are the **window-length(w)** and the **order of the polynomial(n)**
- For H-P filter, the **regularisation parameter λ** is tuned

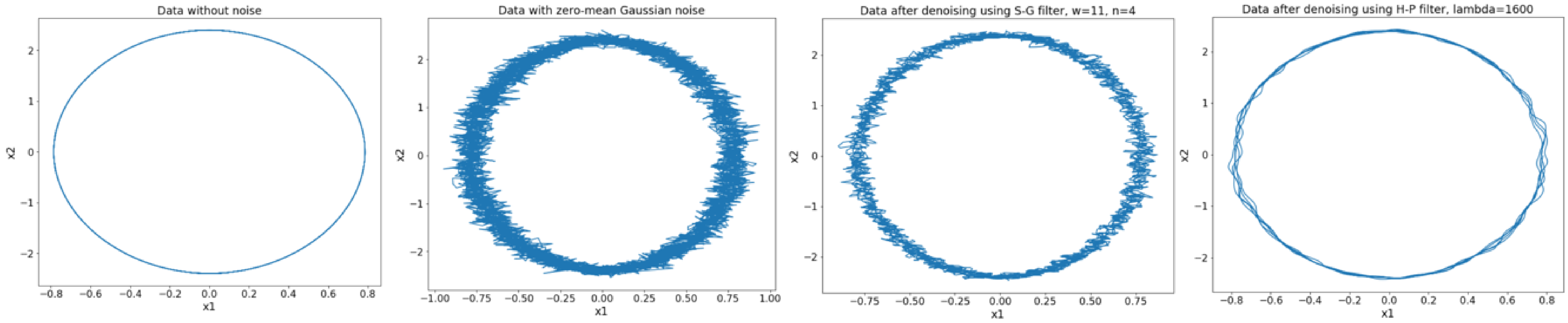

Figure 3: Visualisation of denoising using S-G and H-P filters.

Training Results

- 5000 samples are generated (noisy, SNR=10) and the network (one hidden layer, u=4, v=1) is trained for 5000 epochs
- The results with data denoised using S-G filter are:

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/4,0)**	$[0.102x_2, -0.269\sin(0.281x_1) - 0.924\sin(0.994x_1)]$	1.10	1.12
11	4	**(π/5,0)**	$[0.102x_2, -0.953x_1]$	0.87	0.88
11	4	**(π/6,0)**	$[0.337\sin(0.313x_2), -0.953x_1]$	0.72	0.73

- The results with data denoised using H-P filter are:

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/4,0)	$[0.102x_2, -1.036\sin(0.963x_1)]$	0.048	0.055
(π/5,0)	$[0.101x_2, -1.003\sin(0.991x_1)]$	0.040	0.040
(π/6,0)	$[0.101x_2, -1.018\sin(0.980x_1)]$	0.032	0.033

Inferences

- Lower test errors for H-P filter – very good approximation learnt, in fact, true equation learnt
- For S-G filter, success depends on **training domain**
- Model is successful only for a big enough training domain
- EQL performs better with H-P filter because **denoised estimates are smoother**
- Predicted equations are also more consistent for H-P filter

Modified Architecture

- To **expand applicability** – division units (a/b) are added in the output layer
- Non-trivial step, pole created at $b \rightarrow 0$, abrupt change in convexity
- Basis function - Single branch of hyperbola ${}^{1}/_{b}$ with $b > 0$

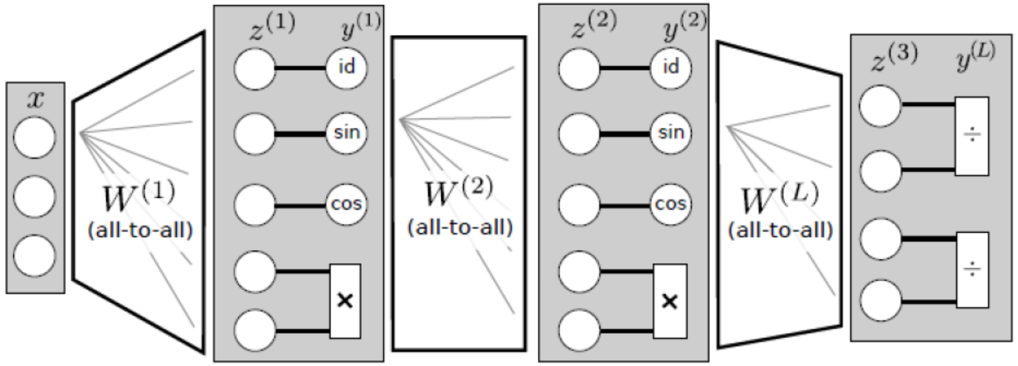

Figure 4: Network architecture of the improved Equation Learner for 3 layers and one neuron per type (u=3, v=1). The new division operations are in the final layer.

Regularised Division and Penalty term

- The final layer output is given by

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

where the division activation function is given by

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

- Cost term that **penalizes** bad inputs to each division unit and global penalty term

$$p^{\theta}(b) = \ max(\theta - \mathrm{b}, 0), \qquad P^{\theta} = \sum_{i=1}^{N}\sum_{j=1}^{n} p^{\theta}(z_{2j}^{(L)}(x_i))$$

- Modified objective function

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

Training Results

- To showcase improved learning capabilities, we attempt to learn
    - Inverse function
    - Exponential function
- Possible for SGD to get stuck in **local minima**
- To circumvent this problem, 5 independent runs are made

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: [0.1,1) – 500 points and [1,50] – 2000 points
- Test set: [51,65] – 100 points
- For each x, target value evaluated as 1/x
- 2-layer EQL_DIV model trained for 3000 epochs, 5 independent runs

Base set	Equation Learned	Train RMSE	Test RMSE
S-1	$\frac{0.3031}{0.3031x + 4.74 \times 10^{-9}}$	1.05×10^{-9}	4.7×10^{-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	$\frac{-0.7119}{-0.7119x + 5.035 \times 10^{-9}}$	4.5×10^{-8}	1.49×10^{-6}
S-4	$\frac{0.3604}{0.3604x + 7.45 \times 10^{-9}}$	1.77×10^{-9}	5.65×10^{-8}
S-5	$\frac{0.3643}{0.3643x - 1.28 \times 10^{-8}}$	4.42×10^{-8}	1.42×10^{-6}

Exponential function

- Training set: [-5, 30] – 1000 points
- Test set: [-6,-5] – 100 points and [30,32] – 100 points
- For each x, target value evaluated as e^{-x}
- 2-layer EQL_DIV model trained for 3000 epochs, 5 independent runs

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

Pade approximation

- Training set: [1, 5] – 500 points
- Test set: [6, 8] – 100 points
- For each x, target value evaluated as $\frac{1-0.5x}{1+0.5x}$
- 2-layer EQL_DIV model trained for 3000 epochs, 5 independent runs

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}

Conclusion

- EQL – **impressive extrapolating** capabilities
- Limitations addressed – Runge-Kutta, numerical methods
- Division units added to enhance learning scope
- Successfully learns inverse, approximation to exponential
- Challenge – to combine these ideas
- Limitations – **domain dependence**, SGD stuck in local optima
- Extension to complex systems – direction for future