Learning Partial Differential Equations from Noisy Data using Neural Networks

Kashvi Srivastava, Mihir Ahlawat, Jaskaran Singh and Vivek Kumar

Department of Applied Mathematics, Delhi Technological University, India
vivekaggarwal@dtu.ac.in

Abstract. The problem of learning partial differential equations (PDEs) from given data is investigated here. Several algorithms have been developed for PDE learning from accurate data sets. These include using sparse optimization for approximating the coefficients of candidate terms in a general PDE model. In this work, the study is extended to spatiotemporal data sets with various noise levels. We compare the performance of conventional and novel methods for denoising the data. Different architectures of neural networks are used to denoise data and approximate derivatives. These methods are numerically tested on the linear convection-diffusion equation and the nonlinear convection-diffusion equation (Burgers’ equation). Results suggest that modification in the hidden units/hidden layers in the network architecture have an effect on the accuracy of approximations with a significant rate. This is a further improvement on the previously known denoising methods of finite differences, polynomial regression splines and single layer neural network.

1. Introduction
A significant amount of real-life phenomena is modelled using differential equations. Both ordinary and partial differential equations and their solutions play a major role in study of disciplines such as physics, biology, economics and engineering. In physical sciences, many examples of partial differential equations are utilized as accurate descriptions of physical processes and their properties. The convection-diffusion equation explaining the behaviour and random movement of micro-particles, the Wave equation describing mechanical or light waves and differential equations giving the exponential decay of radioactive elements are some of the examples. Partial differential equations exist in the theory of general relativity, quantum mechanics, fluid and gas physics and many more fields. In biological sciences, along with other uses, PDEs are used to reveal the spatio-temporal heterogeneity of life systems. Some applications are the study of predator-prey cycle, neural excitement and biological membranes. PDEs are also extensively used in economics to compute consumer theory, producer theory, pricing of financial derivatives and stochastic calculus.

Research in these equations requires domain knowledge and experimental evidence through data collection. Data and its modelling is an important step for understanding experimental variance, analysis of results and communication of these results in every discipline. The advent of machine learning methods and regression techniques allows researchers to measure predictive relationships among descriptive variables. However, one can easily confirm, differential equations are more powerful tools of modelling data. If the learned mathematical model is in the form of a known differential equation, scientists could make use of existing physical or biological equations to investigate the underlying properties of the data. Classic works of PDE learning use sparse optimization and neural networks for data discretization. The sparse identification of nonlinear
dynamics (SINDy) algorithm \cite{1} and neural ordinary differential equations models \cite{2} are developed to discover systems of ordinary differential equations. New feed forward deep network (PDE-Net) \cite{3}, PDE functional identification of nonlinear dynamics (PDE-FIND) \cite{4} and Deep Galerkin Method (DGM) \cite{5} are some algorithms for learning partial differential equations through data-discretization. Even though these discoveries have been instrumental in learning system of differential equations from data, successful PDE learning from noisy data is a question of current research and challenge. High levels of noise are introduced during experimental collection of data due to various sources of errors. Instrumental resolution, environmental and physical factors are some of these sources. Realistic models of noisy data prevent the identification of accurate partial differential equations by introducing extra terms in sparse optimization \cite{6}. In this work, we investigate different methods for denoising data and compare their accuracies in approximating functions and their derivatives. These approximations are then used for learning the partial differential equations defining the data.

Finite differences and cubic spline for polynomial interpolation are established methods for approximating derivatives from noisy data \cite{7}. Further, Hornik et. al. \cite{8, 9} proved that feedforward networks with as few as a single hidden layer are universal approximators with given constraints on the activation function and number of neurons. A single layer artificial neural network with a maximal capacity neuron count has been numerically proven to perform better than finite differences or bi-spline methods on biological spatio-temporal data \cite{10}. However, an important area of sufficient number of neurons required for satisfactory results has not been considered in these works; thus, our aim is to study the effect of change in the number of neurons on the approximation of derivatives of continuous data. We also analyse the change in approximation accuracy if the feedforward networks are made complex by adding more hidden layers. Next, we pass the estimated data to prevalent equation learning methods to compare the results obtained. The denoising methods presented are tested on a few prominent partial differential equations from the world of physical sciences. These equations are applicable in fluid mechanics, gas dynamics, traffic flow, statistical physics and other related fields of engineering \cite{11}.

The paper is divided into sections as follows. Section 2 provides the motivation and method of learning partial differential equations from discretized data through candidate terms and sparse coefficients. Next, we present the denoising models used for approximating the functions and their derivations at various noise levels. The numerical experiments and results of denoising are provided in section 3. Following, we note the results from PDE-Learning with pruning in section 4 which is followed by the conclusion and remarks in section 5.

2. Methodology

2.1 Learning PDE from data

Henceforth, we consider partial differential equations of the form

$$u_t = F(u, u_x, u_{xx}, ...)$$

Where the right-hand side can include non-linear terms of the spatial derivatives. To understand the motivation behind PDE-Learning, consider the problem of learning the convection-diffusion equation from data.

$$u_t + \alpha_1 u_x = \alpha_2 u_{xx}$$

Where $\alpha_1$ and $\alpha_2$ are the convection and diffusion terms respectively. Data is obtained from the above equation and the analytic expression of the differential equation is unknown to the user. Thus, we consider a general second-order partial differential equation with unknown coefficients.

$$u_t = \alpha_1 + \alpha_2 u + \alpha_3 u^2 + \alpha_4 u_x + \alpha_5 u_{xx} + \alpha_6 u u_x + \alpha_7 u_{xx} + \alpha_8 u^2_{xx} + \alpha_9 u u_{xx} + \alpha_{10} u_x u_{xx}$$
Some non-linear terms are omitted for better understanding. The above partial differential equation can be written in the form of product of space-dependent candidate terms and fixed coefficients as

\[ u_t = [1\ u\ u^2\ u_x\ u_{xx}\ u^2\ u_x\ u_{xx}\ u_{xxx}\ u_{xxx} \alpha]. \]

Our aim is to estimate \( \alpha \) in a sparse fashion. Since the above equation holds at all \((x, t)\) points in a given dataset, the right-hand side at a time \(t\) is defined as

\[ X(t) = \begin{bmatrix} 1 & u(x_1, t) & u^2(x_1, t) & \cdots & u_xu_{xx}(x_1, t) \\ 1 & u(x_2, t) & u^2(x_2, t) & \cdots & u_xu_{xx}(x_2, t) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & u(x_n, t) & u^2(x_n, t) & \cdots & u_xu_{xx}(x_n, t) \end{bmatrix}. \]

Similarly, the left-hand side is defined as

\[ V(t) = \begin{bmatrix} u_t(x_1, t) \\ u_t(x_2, t) \\ \vdots \\ u_t(x_n, t) \end{bmatrix}. \]

For a given time, the following system of equations hold

\[ V(t) = X(t)\alpha. \]

Further, for all discrete points in time,

\[ V(t_1) = X(t_1) \quad V(t_2) = X(t_2) \quad \ldots \quad V(t_m) = X(t_m) \]

Produces a linear system of equations for determining \( \alpha \). As we are dealing with real data, the above inverse problem might not be analytically solvable. We use regularized optimization to get rid of the ill-posedness of the inverse problem. Also, \( \alpha \) needs to be the sparsest vector which avoids learning extra terms in the partial differential equation. This can be done by convex L1-regularized least-squares minimization for most cases.

In this work, we use the partial differential equation functional identification of nonlinear dynamics (PDE-FIND) [4] algorithm for learning PDEs. It uses a comprehensive list of candidate terms to accommodate many PDEs. Further, instead of least-squares minimization, PDE-FIND makes use of sequential threshold ridge regression (STRidge) to handle eccentricities. More details on the proposed variations of PDE-FIND for equation learning are provided in section 4.

2.2 Denoising Models

There are various methods to approximate the values of functions and their derivatives from noisy data. Let the learned data be denoted by \( f(x, t) \). Our aim is to minimize the error between \( f(x, t) \) and the actual data. This is especially challenging when various levels of noises are added to the numerically simulated data.

\[ \varepsilon = \frac{1}{s} \sum_{i=1,j=1}^{s} \left( \frac{f(x_i, t_j) - u(x_i, t_j)}{u(x_i, t_j)} \right)^2 \]

Denotes the relative least-squares error function where \( s \) is the size of the dataset. \( \varepsilon \) is a function of parameters used for denoising the noisy simulated data. A detailed explanation of the models and their parameters is provided below. For the final experimentation, models are chosen such that the above error is minimized.
2.2.1 Finite Differences

Finite difference approximations are used as an extension of terms obtained from Taylors’ series expansion of a continuous function. Central differencing is used for better accuracy.

Consider the point \( u(x_i, t_j) \), derivative approximations through central differences formulae are given by

\[
\frac{\partial u}{\partial t}(x_i, t_j) = \frac{u(x_i, t_{j+1}) - u(x_i, t_{j-1})}{t_{j+1} - t_{j-1}} + O(\Delta t^2),
\]

\[
\frac{\partial u}{\partial x}(x_i, t_j) = \frac{u(x_{i+1}, t_j) - u(x_{i-1}, t_j)}{x_{i+1} - x_{i-1}} + O(\Delta x^2).
\]

These are further substituted for evaluation of higher order derivative approximations. Finite difference is an interesting method to study denoising of data as numerical error intensifies over higher order approximations. As we move to challenging differential equations, performance of finite differences as a denoising model tends to deteriorate for data sets with greater noises.

2.2.2 Polynomial Spline Regression

Polynomial approximation through interpolation is a proven efficient method to replicate the behaviour of underlying function of a given dataset. We use local splines with constant variance error model (LCVSP) and non-constant variance error model (LNCVSP) for approximation [10]. The order of splines is adjusted as per the requirement of the partial differential equation in question, for example, order 3 for Burgers’ equation. We use the spline polynomial to analytically calculate the objective function values and derivatives as an approximation at various noise levels.

2.2.3 Neural Networks

The architecture of a basic neural network is mathematically defined as

\[ W_2(\lambda_1(W_1x + b_1)) + b_2 \]

where,

- \( W_1, W_2 \) represent weight matrices;
- \( b_1, b_2 \) represent bias vectors;
- \( x \) is the input vector of the noisy dataset;
- \( \lambda_1 \) represents the activation function.

Weights and biases are used to control the performance of the model and mainly constitute the network parameters set. As proven in [8], a single hidden layer neural network with the assumption that the activation function is continuous, bounded and non-constant acts as a universal function approximator. We investigate the problem of discovering sufficient number of hidden units required for satisfactory performance.

The activation function used here is softplus function i.e., \( \log(1 + \exp(z)) \) [10]. As we extend the structure to double layer networks, softplus function does not give desired results in the second hidden layer. We obtain multiple infinite and undefined values and thus, chose to use hyperbolic tangent function for the activation of the second hidden layer. This might be the case of gradient exploding while back propagation. Using tanh does not affect the results of the neural network as the data is being normalized such that the function values are between 0 and 1. An architecture chart for such a neural network is figure 1.

The choice of number of neurons varies as per the requirement of PDE. We analyse the approximation results over a wide range of neurons for both single and double hidden layer neural networks of comparable complexity. ‘ADAM’ optimizer [12] is used for optimizing weights and
biases over varying size of input batches. Batch size of 100 is considered to avoid overfitting for some experiments. We also use the concept of early stopping to prevent hard-coding of

![Model Summary](image)

Figure 1. Model Summary

the number of epochs required for convergence. This would allow us to efficiently control the performance on the different datasets.

3. Numerical Simulation and Experiments

For numerically solving the problem at hand, we simulate spatio-temporal datasets for physical partial differential equations through MATLAB packages. To understand the efficiency of these models on experimentally collected data, we add different levels of noises to the datasets. Noisy data $U$ can be generated from the simulated data $u$ by

$$U(x, t) = u(x, t) + \sigma * u(x, t) * n$$

Where $\sigma$ indicates the noise level and a random normal variable $n$ is multiplied to utilize the properties of a statistical model along with the numerical model. Through this, the parameters which remain uninterpreted in the mathematical analytic procedure are interpreted in the spectrum of the observed non-exact data [13]. Using the above formula with $\sigma = 0\%, 1\%, 5\%, 10\%, 25\%$ and $50\%$, we consider six different datasets for each equation.

3.1 Convection-Diffusion Equation

The convection-diffusion equation in one dimension is given by

$$u_t = -u_x + 0.01u_{xx}.$$
The dataset constitutes 101 spatial points over $x \in [0,1]$ and 300 points in time ranging over $t \in [0,0.9]$ with the initial condition $u_0 = 1$. The performance of neural networks is analysed by tuning the hyperparameter of number of hidden units in a single hidden layer model. Denoising is tested over six datasets of the convection-diffusion equation with 0%, 1%, 5%, 10%, 25%, and 50% noise levels. Table 1 presents the approximation results for single layer neural network with 500, 1000, 1500, and 2000 neurons. Relative mean squared error between the approximated values of the function values with their derivatives and the true values are presented.

We chose the diffusive coefficient to be small to make the problem more challenging. To numerically simulate the data points, the MATLAB PDE solver pdepe is used. This implements a spatial discretization for the method of lines solution of parabolic equations in one dimension [14].

It could be expected that with the increase in the number of neurons, the performance of neural network enhances for approximation. However, a greater number of neurons leads to a bulky model and even overfitting. This could be easily seen in the results of derivative approximations obtained for the convection-diffusion equation. Neural networks with 500 and 1000 hidden units give the least mean squared error for 0%, 01% and 05%. Some variation in the approximation of $u_{xx}$ is observed.
This is due to the increase in numerical error over higher order derivative calculations. Thus, networks with 1500 and 2000 neurons tend to approximate $u_{xx}$ better.

| Noise Level | Denoising Model | Relative Mean Squared Error |
|-------------|-----------------|-----------------------------|
|             | $u$             | $u$                         | $u$          |
| 00%         | FD              | 0.00e+00                    | 4.14e-05     | 2.82e-03 | 6.30e-02 |
|             | LCVSP           | 1.18e-02                    | 7.47e-01     | 6.36e-02 | 2.59e+00 |
|             | LNCVSP          | 6.80e-04                    | 6.69e+01     | 1.78e+01 | 8.82e+00 |
|             | NN              | 2.28e-03                    | 4.74e-01     | 6.46e-02 | 2.82e+00 |
| 01%         | FD              | 9.89e-05                    | 1.33e+02     | 2.29e-01 | 2.94e+02 |
|             | LCVSP           | 1.18e-02                    | 3.82e+00     | 8.03e-02 | 2.86e+00 |
|             | LNCVSP          | 6.51e-04                    | 2.29e+02     | 2.87e+01 | 2.17e+01 |
|             | NN              | 9.05e-03                    | 4.99e-01     | 9.08e-02 | 8.48e+00 |
| 05%         | FD              | 2.52e-03                    | 2.51e+03     | 6.87e+00 | 4.11e+03 |
|             | LCVSP           | 1.19e-02                    | 3.17e+01     | 2.41e-01 | 3.18e+01 |
|             | LNCVSP          | 1.46e-03                    | 5.78e+03     | 6.38e+02 | 3.29e+02 |
|             | NN              | 1.64e-03                    | 2.89e-01     | 1.98e-02 | 2.51e+00 |
| 10%         | FD              | 9.94e-03                    | 8.10e+03     | 3.57e+01 | 6.18e+03 |
|             | LCVSP           | 1.19e-02                    | 1.49e+02     | 2.03e+00 | 1.02e+02 |
|             | LNCVSP          | 1.56e-03                    | 2.73e+04     | 1.62e+03 | 9.54e+02 |
|             | NN              | 3.42e-03                    | 1.27e-01     | 4.40e-02 | 4.23e+00 |
| 25%         | FD              | 6.31e-02                    | 1.25e+05     | 1.45e+02 | 6.93e+04 |
|             | LCVSP           | 1.94e-02                    | 1.31e+03     | 2.90e+00 | 1.98e+02 |
|             | LNCVSP          | 9.21e-03                    | 6.91e+04     | 1.88e+03 | 4.72e+03 |
|             | NN              | 6.49e-03                    | 7.18e-01     | 2.85e-02 | 3.03e+00 |
| 50%         | FD              | 2.45e-01                    | 3.53e+05     | 7.45e+02 | 2.58e+05 |
|             | LCVSP           | 3.47e-02                    | 4.84e+03     | 1.69e+01 | 8.54e+02 |
|             | LNCVSP          | 2.80e-02                    | 3.13e+05     | 1.84e+04 | 1.06e+04 |
|             | NN              | 1.16e-01                    | 3.00e+00     | 6.16e-01 | 2.69e+01 |

Table 2. Relative mean-squared error between simulated data with its derivatives and the denoised data with its approximated derivatives for the Convection-Diffusion equation. FD denotes Finite differences; LCVSP denotes Local constant-variance spline method; LNCVSP denotes Local non constant-variance spline method; and NN denotes Neural Network with 1 hidden layer and 1000 neurons.

1000 neurons appear to be the perfect choice of sufficient number of neurons for satisfactory performance at 10% and 25% of noise. Adding more noise to the simulated data i.e., 50% requires around 2000 neurons to deliver the best results. We did not consider neural networks with more than 1 hidden layer as the convection-diffusion equation is a linear partial differential equation. Adding more layers would compromise our time-accuracy trade off in comparison to other models. We considered testing the data over neurons less than 500 and more than 2000 but obtained worse approximations in most cases. The final 4 structures are chosen for clarity and simplicity.

Results suggest that the accuracy of a model with respect to the number of hidden units used depends on the induced noise to the dataset of the given equation. Neurons over the range of 500 – 1000 are sufficient for handling data with up to 5% noise. 1000 – 1500 neurons are required to deal with data with up to 25% noise. For the noise level of 50%, a neural network with 1500 – 2000 neurons would give the best function and derivative approximations. According to the diverse results, we select the neural network with 1000 neurons for comparison with conventional denoising models. Equation learning is also depicted in section 4.1.
As shown in table 2, finite difference method performs the best at 0% noise as a denoising model. This is justified as the computations of finite differences are based on the assumption of zero error in the data set. The performance of finite differences rapidly deteriorates as the noise increases to the realistic levels found in experimental data. Local spline regression with non-constant variance approximates the function values with the highest accuracy even at large noises. However, the error in derivative approximations increases with the increase in the noise level for both the spline methods. The neural network with 1000 neurons tends to give the most accurate results at all noise levels except 00% with very limited deflections. Thus, this denoising method proves to be highly beneficial in applications in real world data learning.

### Table 3. Relative mean-squared error between simulated data with its derivatives and the denoised data with its approximated derivatives for the Burgers’ equation. NN1_250 denotes Neural Network with 1 hidden layer and 250 neurons; NN1_500 denotes Neural Network with 1 hidden layer and 500 neurons; NN1_1000 denotes Neural Network with 1 hidden layer and 1000 neurons; and NN2_200_50 denotes Neural Network with 2 hidden layers and 200 neurons in the first hidden layer and 50 neurons in the second hidden layer.

| Noise Level | Denoising Model | Relative Mean Squared Error |
|-------------|-----------------|----------------------------|
| 00% | NN1_250 | 1.17e-03 5.56e-01 2.00e-01 4.44e+00 |
| NN1_500 | 1.15e-03 1.31e+00 5.81e-02 4.49e+00 |
| NN1_1000 | 1.16e-03 4.82e-01 1.61e-01 9.46e-01 |
| NN2_200_50 | 8.54e-04 1.43e+00 7.08e-02 5.18e+00 |
| 01% | NN1_250 | 1.10e-03 8.54e-01 2.88e-01 5.85e+00 |
| NN1_500 | 1.12e-03 2.53e+00 1.56e-01 8.34e+00 |
| NN1_1000 | 1.13e-03 2.57e+00 1.44e-01 5.03e+00 |
| NN2_200_50 | 7.54e-04 7.03e-01 5.40e-02 4.73e+00 |
| 05% | NN1_250 | 1.17e-03 5.92e+00 4.75e-01 1.28e+01 |
| NN1_500 | 1.19e-03 2.06e+00 3.94e-01 1.05e+01 |
| NN1_1000 | 1.07e-03 3.20e+00 5.83e-01 9.18e+00 |
| NN2_200_50 | 1.10e-03 1.06e+00 5.35e-02 1.57e+00 |
| 10% | NN1_250 | 8.66e-04 5.55e+00 1.40e+00 1.41e+01 |
| NN1_500 | 1.17e-03 1.92e+01 2.03e+00 2.48e+01 |
| NN1_1000 | 1.22e-03 1.95e+01 3.05e+00 9.95e+00 |
| NN2_200_50 | 4.90e-04 2.18e-01 5.53e-01 2.39e+00 |
| 25% | NN1_250 | 6.41e-03 1.13e+00 6.34e+00 7.63e+00 |
| NN1_500 | 5.95e-03 2.42e+00 8.71e+00 1.55e+01 |
| NN1_1000 | 5.68e-03 8.12e-01 3.58e+00 9.77e+00 |
| NN2_200_50 | 1.83e-03 2.28e-01 1.66e-00 1.12e+01 |
| 50% | NN1_250 | 6.07e-02 2.73e+00 1.61e+01 1.20e+01 |
| NN1_500 | 5.98e-02 3.36e+00 1.50e+00 1.94e+01 |
| NN1_1000 | 5.81e-02 5.36e+00 2.59e+01 3.99e+01 |
| NN2_200_50 | 7.08e-02 5.66e-01 3.59e-01 7.85e+00 |

### 3.2 Burgers’ Equation
The Burgers’ equation in one dimension is given by

\[ ut = -u u_x + 0.1 u_{xx}. \]
To numerically simulate the data, we make use of the Chebfun package [15] to produce 51456 data points. This constitutes 256 steps ranging over \( x \in [-8,8] \) spatially and 201 steps ranging over \( t \in [0,10] \) in time with the initial condition \( u_0 = \exp(-(x + 2)^2) \). A spectral Fourier discretization with 256 modes and a fourth-order explicit Runge-Kutta temporal integrator with time-step size \( 10^{-4} \) is used [16].

| Noise Level | Denoising Model | Relative Mean Squared Error |
|-------------|----------------|-----------------------------|
|             |                | \( u \) | \( u \) | \( u \) |
| 00%         | FD             | 5.23e-35 | 5.96e-05 | 2.10e-05 | 2.23e-03 |
|             | LCVSP          | 1.64e-05 | 4.49e-03 | 1.36e-03 | 3.23e-02 |
|             | LNCVSP         | 2.22e-06 | 5.20e-03 | 2.72e-03 | 4.15e-02 |
|             | NN             | 8.54e-04 | 1.43e+00 | 7.08e-02 | 5.18e+00 |
| 01%         | FD             | 1.00e-04 | 2.71e+01 | 4.31e+00 | 1.18e+04 |
|             | LCVSP          | 2.26e-05 | 1.35e+00 | 4.47e-01 | 4.73e+01 |
|             | LNCVSP         | 7.95e-06 | 1.29e+00 | 1.17e+00 | 5.24e+01 |
|             | NN             | 7.54e-04 | 7.03e-01 | 5.40e-02 | 4.73e+00 |
| 05%         | FD             | 2.45e-03 | 6.51e+02 | 3.44e+02 | 3.89e+05 |
|             | LCVSP          | 1.37e-04 | 5.55e+00 | 1.77e+01 | 8.43e+02 |
|             | LNCVSP         | 1.28e-04 | 5.83e+00 | 3.52e+01 | 1.04e+03 |
|             | NN             | 1.10e-03 | 1.06e+00 | 5.35e-02 | 1.57e+00 |
| 10%         | FD             | 9.97e-03 | 2.20e+03 | 2.01e+03 | 1.20e+06 |
|             | LCVSP          | 5.35e-04 | 1.16e+02 | 2.37e+01 | 4.07e+03 |
|             | LNCVSP         | 5.46e-04 | 1.15e+02 | 8.88e+01 | 4.47e+03 |
|             | NN             | 4.90e-04 | 2.18e-01 | 5.53e-01 | 2.39e+00 |
| 25%         | FD             | 6.38e-02 | 5.84e+03 | 2.85e+03 | 5.97e+06 |
|             | LCVSP          | 3.26e-03 | 2.52e+02 | 7.21e+01 | 2.07e+04 |
|             | LNCVSP         | 3.58e-03 | 2.51e+02 | 3.66e+02 | 2.21e+04 |
|             | NN             | 1.83e-03 | 2.28e-01 | 1.66e+00 | 1.12e+01 |
| 50%         | FD             | 2.37e-01 | 3.91e+04 | 2.70e+04 | 2.50e+07 |
|             | LCVSP          | 1.21e-02 | 4.03e+02 | 8.16e+02 | 9.38e+04 |
|             | LNCVSP         | 1.58e-02 | 4.24e+02 | 3.13e+03 | 1.33e+05 |
|             | NN             | 7.08e-02 | 5.66e-01 | 3.59e-01 | 7.85e+00 |

**Table 4:** Relative mean-squared error between simulated data with its derivatives and the denoised data with its approximated derivatives for the Burgers’ equation. FD denotes Finite differences; LCVSP denotes Local constant-variance spline method; LNCVSP denotes Local non constant-variance spline method; and NN denotes Neural Network with 2 hidden layers and 200 neurons in the first hidden layer and 50 neurons in the second hidden layer.

To first compare the approximated data from different structures of neural networks (2.2.3), we test denoising over six datasets of Burgers’ equation with 0%, 1%, 5%, 10%, 25% and 50% noise levels. Table 3 includes results for neural networks of a single layer comprising of 250, 500 and 1000 neurons along with double layer neural network with 200 and 50 neurons in the first and second hidden layers respectively. We analyzed the performance on a wide range of neurons but found their approximation to be either similar or worse. Thus, we chose to display only these four structures in tabular form for simplicity.

We found that a neural network with 2 layers and sufficient hidden units performs the best in approximating the data and its derivatives for the Burgers’ equation at almost all levels of noise. However, as verified in [8], it is closely followed by a single hidden layer neural network with 500
neurons for higher noise levels. Neural network with 250 neurons produces the least accurate approximations for all the noise levels except 10%. This can be registered as a data anomaly because all the networks with neurons less than 500 performed worse as the noise increased. This should be the case as less neurons are incapable of learning the properties of the data properly. Thus, the accuracy of approximation increases as we increase the number of neurons involved in a single layer network at a significant rate until it reaches an optimal solution. Here, 500 neurons can be constituted as the sufficient number of hidden units required for satisfactory performance. Even though, including more neurons leads to better denoising of higher order derivative approximations, see values of \( NN1_1000 \) for column \( u_{xx} \) in table 3; its performance is quite comparable to \( NN1_500 \) in most cases. \( NN1_500 \) also outperforms \( NN1_1000 \) at 50% noise. We consider the double layer neural network to compare the performance between other denoising models and neural networks. This choice is further justified as all the different structures took limited and similar time in computation.

Comparing the results of different denoising models as shown in table 4, we found that the neural network performs poorly and finite differences provides the most accurate results at 0% noise. Spline polynomial regression methods perform better at denoising the data values even at large noises. However, the double hidden layer neural network outperforms all the previously prominent denoising models at various noise levels. The difference in the denoising of finite differences and neural networks tends to increase with the increase in noise. This is understandable as numerical error is intensified in the former. Similarly, the approximations done by polynomial regression follow closely that made by the neural network at 1% and 5% but deteriorate rapidly at 10%, 25% and 50% noise.

4. Results

We follow the PDE-FIND algorithm for equation learning from data [4]. The candidate library is adjusted as the partial differential equation becomes lengthy and non-linear. The algorithm uses Sequential threshold ridge regression for sparse optimization of the coefficient matrix. Further, greedy algorithm works well for learning biological models [10]. We test three optimization methods namely STRidge, Greedy algorithm and LASSO [17]. We only mention what works best for the given equation. Pruning is an important step in equation learning as discovered by [18, 19]. It removes the extra terms from a learned equation if they do not improve the accuracy after certain iterations. The pruning threshold value is carefully chosen as it plays a major role in analysing the performance of the equation learning algorithm. The value is kept high for simple and linear equations. However, for learn non-linear and higher order partial differential equations in this work, we keep the value of the pruning threshold not more than 0.5.

4.1 PDE-FIND for Convection-Diffusion equation

The equations learnt through PDE-FIND for the convection-diffusion equation are presented in table 5. This implements the denoised data after using the neural network with 1000 neurons. Clearly, the partial differential equations learnt depict the underlying PDE correctly. The approximated sparse coefficients of spatial terms are quite accurate with absolute error of the order \( 10^{-4} \) even at large noises. The simulated data points and the learnt equations from the neural network after denoising at 50% noise level are plotted as figure 2. Contrary to this, finite differences resulted in the correct PDE only for 00% noise. The spline methods learnt similar coefficients for both 00% and 01% but led to extra unwanted coefficients after that. Adjusting the pruning threshold did not result in better performance for the denoising models. Conclusively, the convection-diffusion equation with the computationally challenging diffusive term of 0.01 was correctly determined from noisy data after denoising by the neural network model with 1000 neurons.

4.2 PDE-FIND for Burgers’ equation

Both \( NN1_500 \) and \( NN2_200_50 \) showed similar results while applying PDE-FIND along with pruning. They learnt quite accurate coefficients as depicted in table 6 and table 7 respectively and performed exponentially better than all the denoising models. At 50% noise, the simulated data and learnt equations are depicted as figure 3. The neural networks with structures other than these two
were not able to learn the accurate equation for higher noises. The local spline model with constant variance gave satisfactory results for only 0% noise. The spline model with non-constant variance gave similar results for both 0% and 1% noise with the approximation error of the order $10^{-3}$.

\[
\text{True Equation} \\
\dot{u} = 0.01u_{xx} - u_x
\]

| Noise | Method   | Learned Equation |
|-------|----------|------------------|
| 00%   | NN1_1000 | $\dot{u} = 0.001036u_{xx} - 0.997890u_x$ |
| 01%   | NN1_1000 | $\dot{u} = 0.009964u_{xx} - 0.969538u_x$ |
| 05%   | NN1_1000 | $\dot{u} = 0.010149u_{xx} - 1.006632u_x$ |
| 10%   | NN1_1000 | $\dot{u} = 0.009700u_{xx} - 0.987436u_x$ |
| 25%   | NN1_1000 | $\dot{u} = 0.009729u_{xx} - 1.005117u_x$ |
| 50%   | NN1_1000 | $\dot{u} = 0.009499u_{xx} - 0.940078u_x$ |

Table 5: Learned coefficients for denoised data by the Neural Network with 1 hidden layer and 1000 neurons at different noise levels for Convection-Diffusion equation

\[
\text{True Equation} \\
\dot{u} = 0.1u_{xx} - uu_x
\]

| Noise | Method   | Learned Equation |
|-------|----------|------------------|
| 00%   | NN1_500  | $\dot{u} = 0.101835u_{xx} - 0.99388uu_x$ |
| 01%   | NN1_500  | $\dot{u} = 0.100845u_{xx} - 1.000281uu_x$ |
| 05%   | NN1_500  | $\dot{u} = 0.099878u_{xx} - 0.988272uu_x$ |
| 10%   | NN1_500  | $\dot{u} = 0.100791u_{xx} - 0.970612uu_x$ |
| 25%   | NN1_500  | $\dot{u} = 0.101378u_{xx} - 0.949908uu_x$ |
| 50%   | NN1_500  | $\dot{u} = 0.108522u_{xx} - 0.846738uu_x$ |

Table 6: Learned coefficients for denoised data by the Neural Network with 1 hidden layer and 500 neurons at different noise levels for Burgers’ equation

Figure 2: (a) Solution of the convection-diffusion equation; (b) Data plot after introducing 50% noise; (c) Solution of the learnt equation from Neural Network with 1 hidden layer and 1000 neurons at 50% noise

Finite differences did not learn the correct equation for any of the noises other than 0% which is justified by the denoising results. The neural network with 2 hidden layers allows PDE-FIND algorithm to learn the accurate Burgers’ equation even at high realistic noise levels.
True Equation

$$u_t = 0.1u_{xx} - uu_x$$

| Noise | Noise | Noise |
|-------|-------|-------|
| 00%   | \(NN2.200.50\) | \(u_t = 0.098805u_{xx} - 1.003823uu_x\) |
| 01%   | \(NN2.200.50\) | \(u_t = 0.101052u_{xx} - 1.006146uu_x\) |
| 05%   | \(NN2.200.50\) | \(u_t = 0.099206u_{xx} - 0.977137uu_x\) |
| 10%   | \(NN2.200.50\) | \(u_t = 0.105717u_{xx} - 1.002486uu_x\) |
| 25%   | \(NN2.200.50\) | \(u_t = 0.099390u_{xx} - 0.945086uu_x\) |
| 50%   | \(NN2.200.50\) | \(u_t = 0.104143u_{xx} - 0.779895uu_x\) |

Table 7: Learned coefficients for denoised data by the Neural Network with 2 hidden layers and 200 neurons in the first layer and 50 neurons in the second layer at different noise levels for Burgers’ equation

Figure 3: (a) One-soliton solution of the Burgers’ equation; (b) Data plot after introducing 50% noise; (c) Solution of the learnt equation from Neural Network with 1 hidden layer and 500 neurons at 50% noise; (d) Solution of the learnt equation from Neural Network with 2 hidden layers and 200 and 50 neurons at 50% noise

5. Conclusion

Learning partial differential equations from discretized data allows scientists to understand the underlying mechanics properly. Several methods have been developed for denoising data to evaluate the objective values and derivatives. Deep learning models have proven to provide the required accuracy in many prominent studies. Parameter tuning plays a major role in determining the success of such a model. Even though, a single hidden layer is sufficient for approximating derivatives, the number of hidden units involved is a major question of research. Through the above work, we investigate the given problem. For the convection-diffusion equation, a neural network with 1000 neurons gives the best approximations at almost all levels of noise. However, for the Burgers’ equation, a neural network with 200 neurons in the first layer and 50 neurons in the second hidden layer performs the best among other structures. Burgers’ equation consists a non-linear term in the
space dimension. This shows how the structure of the underlying PDE could affect the number of units or layers required for producing accurate approximation results.

For this work, two partial differential equations used commonly in the world of physical sciences are considered for testing. In further research, we aim to include more complex physical PDEs in this study. We applied the neural network model and its variations of suitable hyperparameters to the Korteweg–de Vries (KdV) equation. The KdV equation for one dimension can be defined as

\[ u_t = 6uu_x - u_{xxx} \]

Where the constant 6 is conventional. Implementing the denoising models on numerically simulated data of the KdV equation, the approximation results obtained looked promising. This was particularly the case for a neural network with 2 hidden layers and sufficient number of hidden units in both layers. As the equation is non-linear and consists of spatial derivatives of up to the third order, increasing the number of layers and units provided better results for most cases. However, as we moved to learning the PDE from the denoised data, we did not obtain the correct equation from any of the methods even at small noise levels. PDE-FIND along with pruning could not learn the KdV equation through approximated function values and its derivatives. Numerical errors and noises tend to intensify with higher order derivative terms and non-linear terms. Thus, we opted out of presenting the approximation results in this work for more complex PDEs. In future study, we can work with the current structures of neural network to improve our results and correctly learn PDE with higher order terms. Another problem of research can be using the different deep learning models and techniques, which have been successful in other problems, for denoising. These include implementing Gated Recurrent Unit (GRU) networks, Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNN) for approximating scientifically important functions and their derivatives through data discretization.

References

[1] Brunton S L, Proctor J L and Kutz J N 2016 Discovering governing equations from data by sparse identification of nonlinear dynamical systems Proc. Natl Acad. Sci USA 113(15) 3932-3937 doi: https://doi.org/10.1073/pnas.1517384113
[2] Ricky T Q and Chen 2018 Neural ordinary differential equations Advances in Neural Information Processing Systems
[3] Long Z 2018 PDE-Net: Learning PDEs from Data Proceedings of the 35th International Conference on Machine Learning PMLR 80 3208-3216
[4] Rudy S H 2017 Data-driven discovery of partial differential equations In: Science Advances 3(4) 602614 doi: 10.1126/sciadv.1602614
[5] Sirignano J and Spiliopoulos K 2018 DGM: A deep learning algorithm for solving partial differential equations Journal of Computational Physics 375 doi: https://doi.org/10.1016/j.jcp.2018.08.029
[6] Schaeffer H 2017 Learning partial differential equations via data discovery and sparse optimization Proc. R. Soc. A 473.20160446 doi: https://doi.org/10.1098/rspa.2016.0446
[7] Knowles I and Robert J R 2014 Methods for numerical differentiation of noisy data Electronic Journal of Differential Equations Conference 21
[8] Hornik K, Stinchcombe M and White H 1989 Multilayer Feedforward Networks are Universal Approximators Neural Networks 2 359-366
[9] Hornik K 1990 Approximation Capabilities of Multilayer Feedforward Networks Neural Networks 4 251-257
[10] Lagergren J H 2020 Learning partial differential equations for biological transport models from noisy spatio-temporal data Proc. R. Soc. A 476: 20190800 doi: http://dx.doi.org/10.1098/rspa.2019.0800
[11] Mayur P B 2018 A systematic literature review of Burgers’ equation with recent advances Pramana – J. Phys doi: https://doi.org/10.1007/s12043-0181559-4
[12] Kingma D P and Jimmy Ba 2017 Adam: A Method for Stochastic Optimization doi: https://arxiv.org/abs/1412.6980v9
[13] Anderssen R S and Bloomfield P 1974 Numerical Differentiation Procedures for Non-Exact Data”. In: Numerische Math 22 157–182 doi: https://doi.org/10.1007/BF01436965
[14] Skeel R D and Berzins M 1990 A Method for the Spatial Discretization of Parabolic Equations in One Space Variable SIAM Journal on Scientific and Statistical Computing 11
[15] Driscoll T A, Hale N and Trefethen L N 2014 Chebfun Guide
[16] Raissi M 2018 Deep Hidden Physics Models: Deep Learning of Nonlinear Partial Differential Equations
[17] Tibshirani R 1996 Regression Shrinkage and Selection via the Lasso Journal of the Royal Statistical Society. Series B (Methodological) 58 (1) 267-288
[18] Anders U and Korn O 1999 Model selection in neural networks Neural Networks 2(2) doi: https://doi.org/10.1016/S0893-6080(98)00117-8
[19] David J C and Mackay 1995 Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks Network: Computation in Neural Systems 6 doi: https://doi.org/10.1088/0954-898X_6_3_011