AMATH 563 Homework 4

Christopher Liu

June 14, 2020

Abstract

Neural networks are powerful tools that allow us the predict the future state of a dynamical system. In this assignment we will examine 3 different systems, the Kuramoto–Sivashinsky equation, the Reaction-Diffusion system, and the Lorenz System. Our goal will be to train a neural network to accurately predict the future state of a system for a given set of initial conditions by comparing it with a numerical solution. We observe that the sheer number of parameters and options available when constructing and training a neural network coupled with limited computing power make it very difficult to train an accurate network for future state prediction given the limited time frame.

1 Introduction and Overview

In this assignment, we will be applying neural networks to the Kuramoto–Sivashinsky equation, the Reaction-Diffusion system, and the Lorenz System. The goal of this assignment is to train neural networks that can accurately predict the future-state of the system given a set of initial conditions. By comparing our forecast from our neural network to a numerically calculated solution, we can then determine the overall accuracy of the network and also how far into the future each neural network can accurately predict. Furthermore, for each system, we will vary the way we train the network. For the Kuramoto–Sivashinsky equation, we will train the data on only one set of initial conditions before cross-validating on another set. For the Reaction-Diffusion system, we will train the network using data that has been projected to a low-dimensional subspace via the SVD. Finally for the Lorenz system, we will also vary the constant ρ in the system.

2 Theoretical Background

2.1 Neural Networks

Similar to the linear regression we applied in Homework 1 and the machine learning algorithms we explored in Homework 3, the goal of a neural network is to map a set of input data to a set of output data [1]. However, unlike linear regression, neural networks are capable of also employing nonlinear mappings and intermediate output layers that increase flexibility and performance. Therefore, our mapping from our input layer x to our output layer y can be described as the following.

$$y = f_M(A_M, \dots, f_2(A_2, f_1(A_1, x)) \dots) \tag{1}$$

where each A_k represents the weights connecting the kth layer to the $(k+1)$th layer and f_k are functions which are our nonlinear mappings.We are then able to train our neural networks work by optimizing the following function using stochastic gradient descent and back propagation algorithms.

$$\underset{A_j}{\operatorname{argmin}}(f_M(A_M, \dots, f_2(A_2, f_1(A_1, x)) \dots) + \lambda g(A_j)) \tag{2}$$

for a given regularization $g(A_j)$) and a scaling constant λ[1]. 3 types of activation functions are utilized in this assignment,

$$\text{Linear - } f(x) = x$$

$$\text{Rectified Linear Unit (ReLU) - } f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

$$\text{Sigmoid - } f(x) = \frac{1}{1 + e^{-x}}$$

2.2 Dynamical Systems

The 3 different dynamical systems we will study in this assignment are the following:
Kuramoto–Sivashinsky equation:

$$u_t = -u \cdot u_x - u_{xx} - u_{xxxx}$$

with periodic boundary conditions.

Reaction–diffusion system:

$$\partial_t q = D\nabla^2 q + R(q)$$

where q is the unknown vector function, D is a diagonal matrix of diffusion coefficients and R accounts for all local reactions [2].

Lorenz system:

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy - \beta z$$

where σ, β and ρ are constants.

3 Algorithm Implementation and Development

A separate neural network was trained for each dynamical system. Due to time constraints and with the information provided in the class notes, the number of hidden units, layers, choice of activation functions, optimizer and number of epochs was more or less arbitrary. All 3 neural networks use the adam optimizer. Before we train our neural networks, a numerical solution was created using provided code. Rather than attempting to classify each observation (like in previous assignments), the input is the state of the system at time t and the output label is the state of the same system at time $t + \delta t$. Once the model has been trained, we then use it to predict future states by feeding the model a set of initial conditions. The output of that is then used to predict the state of the system at the next time step. This process is repeated iteratively until the desired future forecast is reached.

3.1 Kuramoto–Sivashinsky Equation

For the K-S equation, two sets of data with different initial conditions are used. Little is done to the data beforehand other than to create an input set where each row is the state of the system at time t for each spatial coordinate and the corresponding output is the state of the system at time $t + \delta t$ for each spatial coordinate. A neural network with two layers is used with a linear activation function followed by a ReLU function. 100 epochs are used to train the model.

3.2 Reaction–diffusion System

For the R-D system, we also use two sets of data with different initial conditions. The data is arranged such that each column contains the value of u and v for all x and y at a given time. To determine the rank, we take the SVD of our data and examine the singular values. We then truncate the resulting V^T matrix to an appropriate rank and use that as our input and output data. We are then able to project the low-dimensional data back into the original high-dimensional subspace by multiplying the output by the properly rank truncated U and Σ. 2-layers are used this for this network, each with 100 units and with a sigmoid function followed by a ReLU function. 100 epochs are used to train the model.

3.3 Lorenz System

The Lorenz System differs from the other 2 as we are also varying a constant and not just the initial conditions. To train our model, 100 different random initial conditions are constructed using $\rho = 10, 28, 40$. Each row of our training data is then the x, y and z coordinates at a given time and the value of ρ. Each row of our output data is then the x, y and z coordinates at the next time step. A similar approach is used to cross-validate our model except we now use $\rho = 17, 35$. A 3-layer model is used this time, each with 10 units. The activation functions are linear followed by sigmoid and then ReLU. 30 epochs are used to train the model.

4 Computational Results

4.1 Kuramoto–Sivashinsky Equation

Figure 1 shows the future forecast of the system with the same initial conditions as the training data for a variety of different spatial coordinates. Note that this is different from using the entire training data set. We would expect that calculating the forecast iteratively even for the training data would produce good results. However, we note that this model is only somewhat accurate till about $t = 50$.

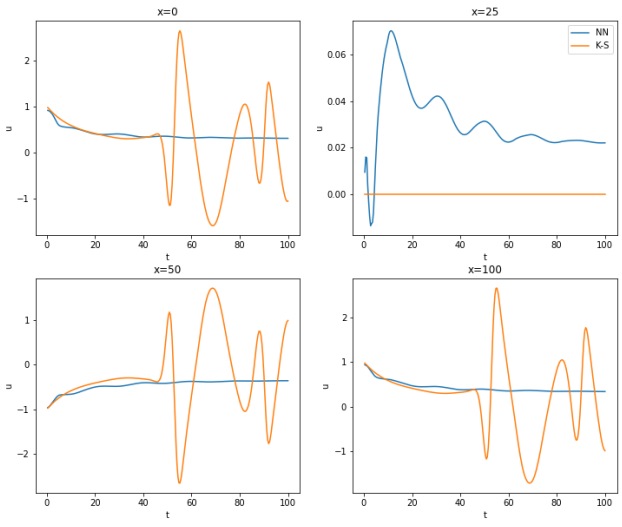

Figure 1: Model forecast on the initial conditions from the training data

Next, we try to forecast another set of initial conditions. A few evolutions for different spatial coordinates are shown in Figure 2. The model does a very poor job of forecasting the behavior of the system although it is able to capture some of the oscillatory behavior.

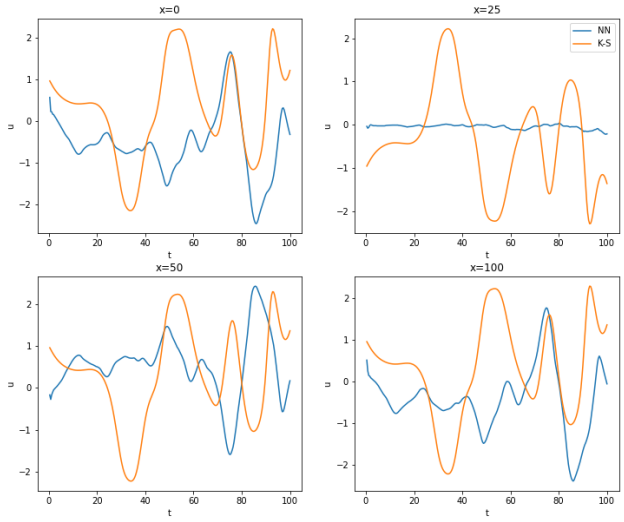

Figure 2: Model forecast using different initial conditions

4.2 Reaction–diffusion System

The singular values of our data are plotted on Figure 3. We note that there are 3 singular values that pop out which is expected since our system depends on x, y and t.

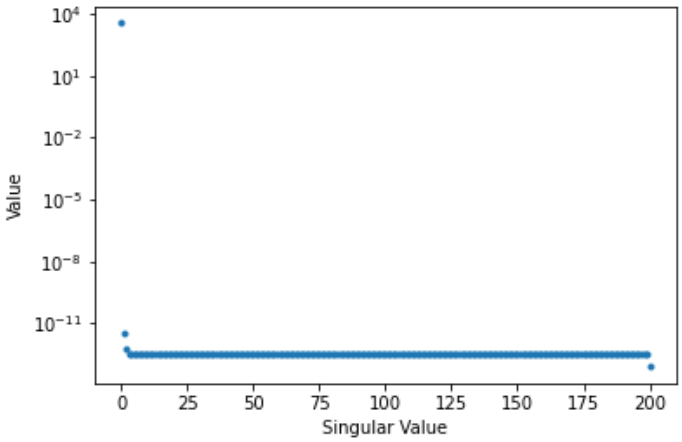

Figure 3: Singular values of R-D System

Using a rank 10 truncation, we attempt to simulate the system using our neural net. The state of the system at $t = 0.5$ is shown on Figure 4. This appears to be a pretty good prediction by the neural network.

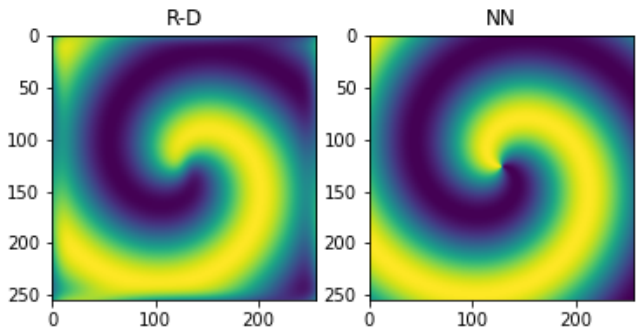

Figure 4: R-D Solution vs Neural net at $t = 0.5$

Figure 5 shows the state of the system at $t = 1.5$. We note that the NN forecast begins to differ from that of the R-D system. Similar to the K-S equation, our neural network starts to become inaccurate as time progresses.

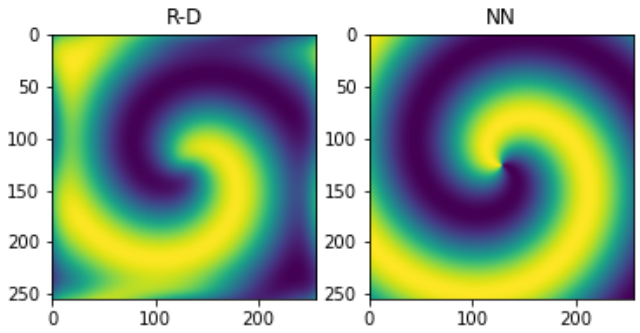

Figure 5: R-D Solution vs Neural net at $t = 1.5$

4.3 Lorenz System

Figures 6 and 7 show trajectories of the future state prediction for $\rho = 17$ and $\rho = 35$ respectively. Similar to the K-S equation, we observe that the forecast is good up until a certain time. However, given the simplicity of the model and arbitrary choice of parameters, the model seems to be able to pick out some key features in the system.

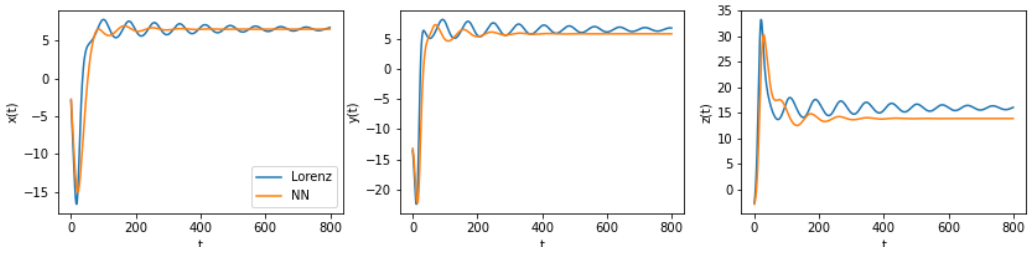

Figure 6: Future state prediction with $\rho = 17$

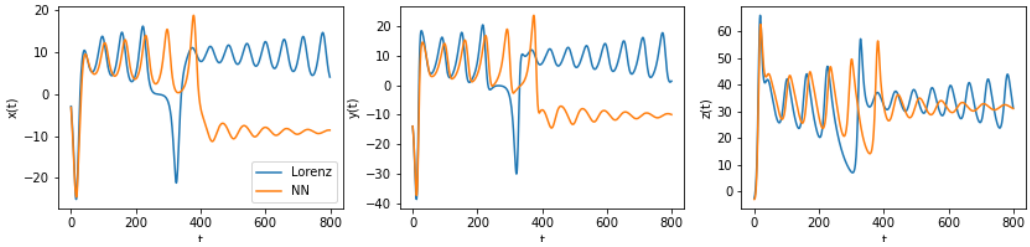

Figure 7: Future state prediction with $\rho = 35$

5 Summary and Conclusions

In this assignment, we were able to show the potential for neural networks to predict the behavior of dynamical systems. Even with simple models and arbitrarily chosen parameters, the models perform okay. However, unlike methods such as linear regression, neural networks require a considerable amount of time and computing power to create accurate models. For more simple systems, it appears that using classification or regression algorithms covered in previous assignments is more suitable. Nonetheless, the flexibility of neural networks give it the upper hand when more complex systems and problems are considered.

References

[1] Jose Nathan Kutz and Steven Brunton. *Data-driven science & engineering machine learning, dynamical systems, and control.* Cambridge University Press, 2019.

[2] Wikipedia contributors. *Reaction–diffusion system* Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 23 May. 2020. Web. 10 Jun. 2020.

Appendix A Python Functions

- `scipi.io.loadmat()` Loads a .mat file
- `plt.imshow(A)` Displays data as an image
- `numpy.linalg.svd(A)` Computes the SVD of a given matrix
- `keras` A deep learning API used to create the neural networks in this assignment documentation can be found at https://keras.io/

Appendix B Python Code

https://github.com/chrismhl/AMATH563/tree/master/Homework4