Recurrent Neural Network Models for Predicting Ordinary Differential Equation Dynamics

Camille Renaud & Dr. Tan Bui-Thanh*

Oden Institute for Computational Engineering and Sciences
Moncrief Summer Internship

August 7, 2020

Abstract

Recurrent Neural Networks (RNN) have shown to have remarkable capabilities when it comes to modeling sequential data and have been at the forefront of research in time-series forecasting problems in a variety of fields. First looking at RNNs with a mathematical focus, we saw whether claims of model stability from published and unpublished papers could be rigorously defended. We then investigate the practical abilities of RNNs through their implementation on certain systems of ordinary differential equations (ODE). We consider two approaches of Long Short-Term Memory (LSTM) Recurrent Neural Network architectures for predicting future time steps of solutions to first and second order ODEs as well as systems of coupled ODEs to use as a guide for more complicated systems in the future. This project was made possible by the wonderful experience of the Moncrief Summer Internship at the Oden Institute of Computational Engineering and Sciences and under the mentorship and guidance of Dr. Tan Bui-Thanh and the Probabilistic and High Order Inference, Computation, Estimation, and Simulation Group.

Contents

1 Introduction

Recurrent Neural Networks are a class of Neural Networks that are popular modeling techniques for solving sequence learning problems. Vanilla RNNs have the capacity to accept both variable length sequences on the input as well as variable length sequences on the output which makes their applications incredibly versatile. Recurrent Neural Networks have been found to have remarkable results in tasks such as machine translation, time-series forecasting, speech recognition and synthesis, and numerous other problems. A relatively new field of study, however, is using RNNs to predict solutions for systems of differential equations. Throughout this study, we consider two models that utilize various popular techniques in RNNs and look at how they can predict the solutions of ordinary differential equations.

1.1 Mathematical Theory

In general, a RNN has a recurrent core cell that takes some input x, feeds it into the RNN that has an internal hidden state, which then is updated for every input x. This updated internal state is then fed back into the model when it reads the next input. Also typically, the RNN will produce an output y at each time step. Within the RNN, there is a recurrence relation that we can apply to x as well as old hidden states to achieve the new hidden state as follows

$$h_t = f_W(h_{t-1}, x_t)$$

Where f is some function that depends on parameters W and accepts the previous hidden state h_{t-1} and the current input vector x_t at the current time-step t. Thus, the simplest functional form of a RNN, referred to as a Vanilla RNN, is made up of the update of a hidden state h_t and the computation of an output y_t

$$\begin{aligned} h_t &= \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t) \\ y_t &= W_{hy} \cdot h_t \end{aligned}$$

where W_{hh}, W_{xh}, and W_{hy} are weight matrices, and tanh represents the activation function that squashes the hidden state between -1 and 1. The recursions above explain the forward pass of the network that steps forward through time and then calculates a loss based on the training data inputted. A helpful visualization of this idea of a recurrence relation is to unroll the computational graph for multiple time-steps which allows for a visualization of the hidden-state. Below, figure 1 displays a graph of this sort in which each time-step is unrolled to see how the inputs, hidden states, weights, and outputs move through an RNN.

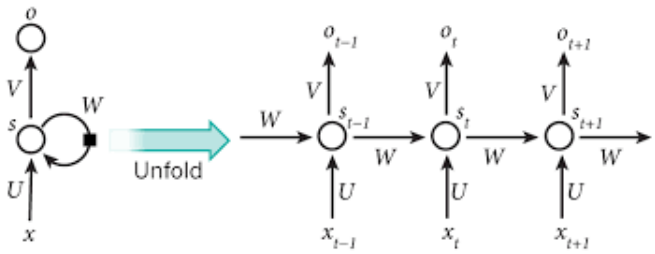


Figure 1: The unrolled feed-forward structure of a Vanilla RNN

After computing the forward pass of the RNN and the loss associated with its prediction, we then want to train the weights and effectively minimize the loss calculated. In RNNs, a common technique is Backpropagation through time (BPTT) where you pass backwards through the entire sequence of the RNN to compute the gradients of the output with respect to the weights, and then update the weights. The back-propagation process is the same for artificial neural networks (ANN), but with RNNs, the current time step is calculated based on the previous time step [7]. Computing the gradients of the change in the error with respect to the change in weights, we can update the weights the minimize the lose through multiple forward and backward passes. Ultimately, the stochastic gradient descent algorithm finds the global minimum of the cost function which is your optimized network that will produce the best results.

Just like ANNs, Vanilla RNNs suffer from the problems with learning long-term dependencies due to vanishing or exploding gradients. These issues where the gradients getting either exponentially small, or exponentially large makes it difficult for Vanilla RNNs to effectively update the weights of previous time-steps. This issue happens because as the model backpropagates the error through the time-steps, the gradients are being calculated via chain-rule, in which the gradients involve multiple factors of the same weight matrix W.

There are various ways that this vanishing/exploding gradient problem is rectified, and it is still an open problem today on how to best fix this issue. But one way that the problem has been shown to improve is through a RNN architecture class called Long-Short Term Memory RNNs.

1.2 Long-Short Term Memory RNN

RNNs suffer from short-term memory where they are not able to carry information from earlier time steps to later ones due to the vanishing gradient problem. One solution to this issue is a gradient based method called Long Short Term Memory (LSTM) [6]. Instead of the simple recurrence relation to update the hidden state in a Vanilla RNN, in an LSTM, there are two hidden states that are maintained at every time-step. One is the h_t which acts similar to the hidden state in the Vanilla RNN, and the other is the cell state C_t. Both of these hidden states are computed through four gates that serve to govern whether the input data should be added to the cell state or forgotten. The mathematical structure is as follows

$$
\begin{aligned}
f_t &= \sigma(x_t \odot W_{fx} + h_{t-1} \odot W_{fh}) \\
i_t &= \sigma(x_t \odot W_{ix} + h_{t-1} \odot W_{ih}) \\
o_t &= \sigma(x_t \odot W_{ox} + h_{t-1} \odot W_{oh}) \\
\tilde{C}_t &= \tanh(x_t \odot W_{cx} + h_{t-1} \odot W_{ch}
\end{aligned}
$$

$$
\begin{aligned}
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\
h_t &= o_t \odot \tanh(C_t)
\end{aligned}
$$

where the forget gate f_t determines whether to erase the cell, the input gate i_t determines whether to write the cell, the candiate cell gate $\tilde{C)}_t$ determines how much should be written to the cell, the output gate o_t determines how much to reveal to the hidden cell, $\odot$ denotes element wise multiplication, and $+$ denotes the element wise addition.

The cell state is an internal state which allows for information to flow easily and relatively unchanged from one cell to the next. In Figure 2 below, a diagram of the LSTM cell architecture.

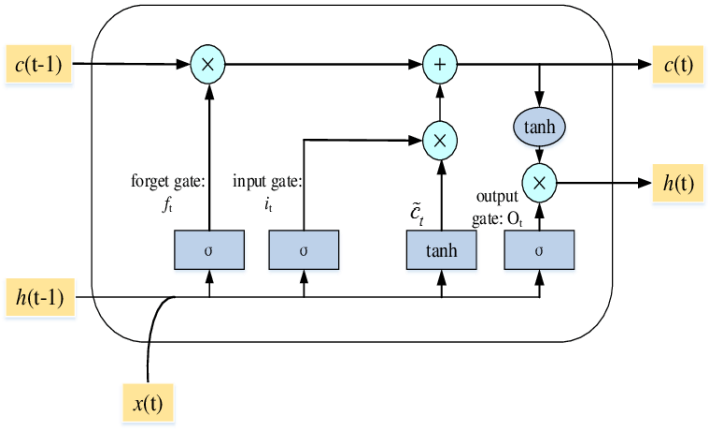


Figure 2: The cell structure of an LSTM RNN architecture for time step t

During the backwards pass of computing the gradients of the cell state, the problem of vanishing gradients is solved by design. As we backpropagate through the cell state, the only change that happens to the gradient is an element-wise multiplication with the forget gate f_t that can vary between each time-step, instead of before when there was matrix multiplication by the same weight matrix W which is what led to the exploding or vanishing gradients. This cell state can be interpreted as an uninterrupted gradient flow.

1.3 Connections Between RNNs and ODEs

RNN models are extremely powerful for modeling sequence data, and while the study of ordinary differential equations and dynamical systems is vast, the connections between RNNs and dynamical systems have not been well explored. However, there are some comparisons one can make when looking at the structure alone of RNNs and how they relate to the solutions of ODEs. Throughout the 1990s, there was what was considered milestone research in training Neural Networks (NN) to solve some of the issues of traditional numerical methods such as Shooting Method, Runge-Kutta based methods, or Finite Difference Methods [13]. Using NNs were shown to be superior to these traditional numerical methods in providing approximate solutions. There has also been recent studies able to theoretically and structurally connect neural network architectures to the stability and dynamics of ODEs in a new architecture of NNs called Neural Ordinary Differential Equations. These Neural ODEs show the potential of differential equations within the structure of NNs for time series data analysis [2, 3].

In the realm of RNNs, there has been much research surrounding its ability to handle long-term dependencies, and in terms of ODEs, there are some very recent papers that are proposing a redesign of traditional RNN architectures that leverages the mathematical foundation of ODEs to optimize their performance [4].

1.3.1 Stability Analysis of Implicit Runge-Kutta and ODERNNs

Coming from a mathematical background, we began the summer looking at RNNs with a math focus, and seeing if results in published and unpublished papers could be rigorously defended [1, 3]. Particularly, we looked at the BN stability of a proposed architecture of a ODERNN that mimicked implicit Runge-Kutta numerical method and looked to be able to carefully fill in the missing details and prove the BN-stability of the general implicit Runge-Kutta numerical method which then maps to the paper's proposed architecture of RNN [1]. The proof of BN-stability for implicit Runge-Kutta Method (Theorem 1) can be seen in Appendix A.

With the LSTM RNN architectures specifically designed to model temporal sequences, we consider their capabilities in training on previous solution data to produce future solutions in time for various ODE systems. Through this paper, a general study of two LSTM architectures are analyzed in their ability to produce future solutions of systems of ordinary differential equation where hyper-parameter and parameter optimization, and various data augmentation techniques were used to minimize the error of our solutions.

2 Model Descriptions

Various approaches and modifications to LSTM RNN models have been found that adhere to certain problems better than others [9, 11, 13]. As the problem of predicting or extending solutions of ODEs is similar to time-series forecasting problems and can be formatted in various ways, there are two approaches that were considered using standard LSTM architectures that have success with their respective goals. The first approach being to extend an exact solution an arbitrary amount by a recursive multi-step approach, so that the entire solution starting from initial condition is extended one point at a time. The second approach has in mind the goal of extending a particular part of the solution in a multi-step approach in which the solution of multiple time-steps is used to predict the next entire time frame in a single-shot manner. Both of these models were tested on varying types of ODEs including 1st and 2nd order ODEs, and systems of multiple coupled ODEs as well.

2.1 Model 1 - Recursive Multi-Step LSTM

The first model that we consider in solving solutions of ODEs for future time-steps was a recursive multi-step RNN model that learns to predict values over a future time period based on solution training data of the past. This model takes previous solutions and information from the past solutions to predict the next solution where the input for say time-step t is the output from a previous time-step in the prediction set. The model's architecture includes a data generator that trains the model by unrolling the model and producing batches of data to train off of [11]. To increase the robustness of this model, we also did not assume that the output for x_t is always x_{t+1}. Instead, we randomly sampled an output from the set $x_{t+1}, x_{t+2}, \dots x_{t+N}$ where N is a small window size.

We defined our hyper-parameters for this model including the number of unrollings which denotes how many continuous time steps you consider for a single optimization step. Instead of standard

single-step LSTM models, as we optimize through BPTT, the model looks at many time-steps in the past. The number of unrollings that optimized error on the ODEs was 100. Batch size, or how many data samples you consider in a single time-step was determined through the trail process as well. We determined that a large batch-size of 100 did the best for these types of problems. We also considered number of layers of LSTMs to add and the number of neurons at each layer. Since our dataset was relatively small, we found that only one layer of LSTMs was needed to produce good results on our certain ODEs, and any more layers resulted in fast overfitting of the model. A learning rate of 0.0001 was also selected for each of the examples.

2.2 Model 2 - One-Shot Multi-Step LSTM

The second model that we consider serves a slightly different purpose within the goal of solving future solutions of ODEs. With this model, we wanted to be able to predict an entire range of future values from multiple inputs through one-shot predictions where the model makes the entire sequence prediction in a single step [9]. For example, given 100 time-steps of past solutions, the model will predict the next 100 data points. This was done by implementing a Dense layer into our model. For this model, a window object is created to split up our dataset into slices which consist of training data and target data. The model is then trained on the training data and learns to predict solutions given slices of the target data.

This is a slightly modified view of the problem, but the outcome can be looked at as the same type of challenge of predicting future solutions of ODEs. For our model, we created windows of size 200 where they were split into training data of size 100, and testing data of size 100. We also determined a batch-size of 32, and the dense layer of size 100 optimized the loss. As these windows are smaller than in Model 1, the visualizations will look slightly different, but the results can be interpreted similarly.

3 Experiments and Results

In order to investigate the abilities of RNNs on predicting future solutions of certain ODEs, we chose some simple examples of systems of ODEs that would challenge the models in different ways. The systems of ODEs range from 1st order equations, to 2nd order forced systems, to systems of multiple coupled ODEs all posed as initial value problems that have exact solutions. These systems are described as follows [10, 12].

1st Order System:

$$\frac{2ty}{t^2+1} - (2 - \ln(t^1 + 1))y' = 0 \qquad\qquad y(5) = 0$$

2nd Order Systems:

$$a.) \quad y'' + 2y' + 17y = -2\sin(3t) \qquad y(0) = 2, \quad y'(0) = 2$$
$$b.) \quad \frac{1}{2}y'' + 8y = 10\cos(\pi t) \qquad y(0) = 0, \quad y'(0) = 0$$

Coupled Systems:

Lotka-Volterra Equations:

$$\frac{dx}{dt} = x(a - by) \qquad x(0) = 1.5$$
$$\frac{dy}{dt} = -y(c - dx) \qquad y(0) = 1$$

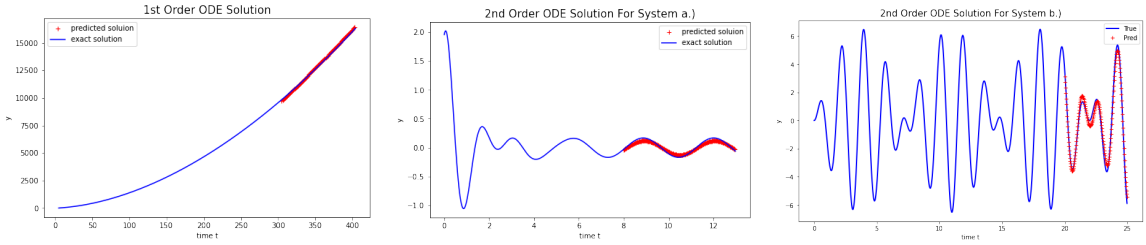
Chemical Reaction System

$$\frac{dA}{dt} = -k_1 AB \qquad A(0) = 1$$
$$\frac{dB}{dt} = -k_1 AB - k_2 BC \qquad B(0) = 1$$
$$\frac{dC}{dt} = k_1 AB - k_2 BC \qquad C(0) = 0$$
$$\frac{dD}{dt} = k_2 BC \qquad D(0) = 0$$

Throughout all of the examples in both models, the data was created from the exact solution equations of each of these systems and was of size 3000 points within their varying time frames. the first 2000 data points were for training, and the last 1000 points were set aside for testing and validation in each model.

3.1 Model 1 on First and Second Order ODEs

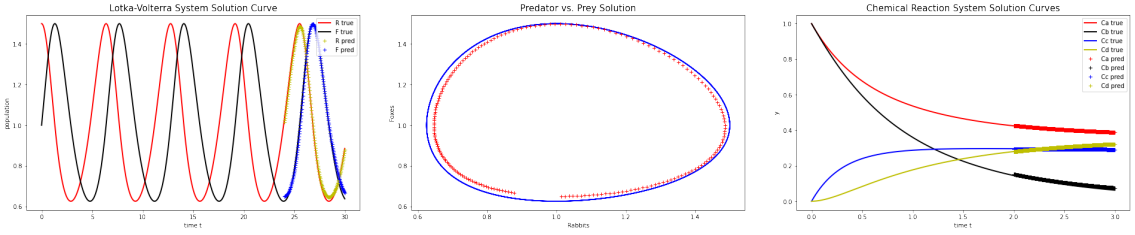
For the implementation of our examples into the first model, we considered the entire training dataset for each run, and the entire testing dataset for what we wanted the model to predict. For the first order and second order examples, training was relatively similar for each model. The first order model converged in approximately 10 epochs, while the second order model a.) converged in 30 epochs and the second order model b.) converged in 20 epochs. All of these examples were evaluated using the mean relative point-wise error, of which, they all were consistently under 7% mean relative point-wise error.



The 1st order ODE prediction had 0.820% mean relative point-wise error, the 2nd order ODE system a.) had a 3.58% mean relative point-wise error, and the 2nd order ODE system b.) had a 6.67% mean relative point-wise error. The first order equation can be seen as the minimal working example where there are no features the model is challenged with. The two 2nd order equations are more involved forced harmonic oscillators with different types of challenges for the model to handle. One has information in the past that no longer matters, and the other has a repetition of a longer sequence that it needs to capture. While still fairly basic, these various equations captured the models ability to produce good predictions on working examples so that we can conclude that the model has potential to be used for more complicated systems or observational data problems.

3.2 Model 1 on Coupled Systems of ODEs

Next, we consider systems of coupled ODEs to see how the model does on models that, while simplified, can be seen as models for real world applications. For the two systems of coupled ODEs, we had to modify the model so that it could accept vector inputs and effectively be trained on 1-dimensional vectors instead of singular points. This added a challenge of robustness to the model as all equations had to be run on the same parameters and hyper-parameters. The first coupled system of ODEs was a classic Lotka-Volterra equations (a.) which consist of a pair of first-order non-linear ordinary differential equations. They represent a simplified model of the change in populations of two species which interact via predation. Graphed was both solutions on t, as well as the solutions graphed on each other showing the cyclical nature of simplistic predator-prey models. The second system (b.) models a chemical reaction network from species balances. Where as the chemicals react, species A and B react to create chemical C which then creates D in varying amounts and speeds based on initial conditions and parameters k_1 and k_2.

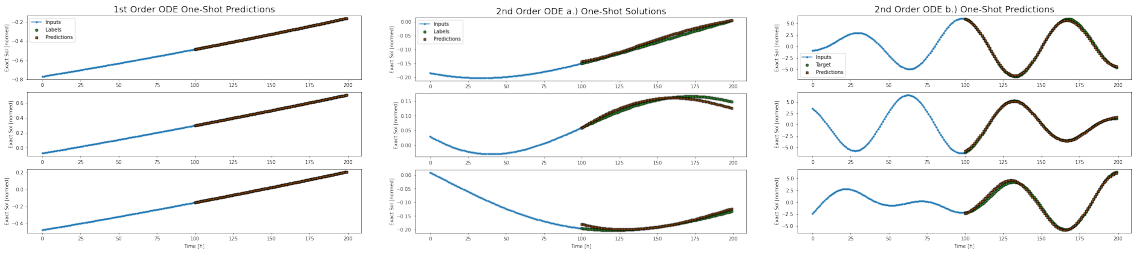


For these two systems, there is an added challenge from the single equation systems in which the model must train on all of the curves using the same parameters and hyper-parameters. For the Lotka-Volterra system, the solutions are essentially the same, but shifted. The system modeling a chemical reaction between four species of chemicals however exhibit different behaviors that was

more challenging to find a successful combination that optimized the errors below 10%. After trails of different parameter combinations, the model was able to produce errors of 1.36% and 1.72% mean percent errors respectively for the prey and predator solutions, and for the chemical reaction system, the errors were 2.16%, 5.86%, 3.51%, and 0.149% for species A, B, C, and D respectively.

3.3 Model 2 on First and Second Order ODEs

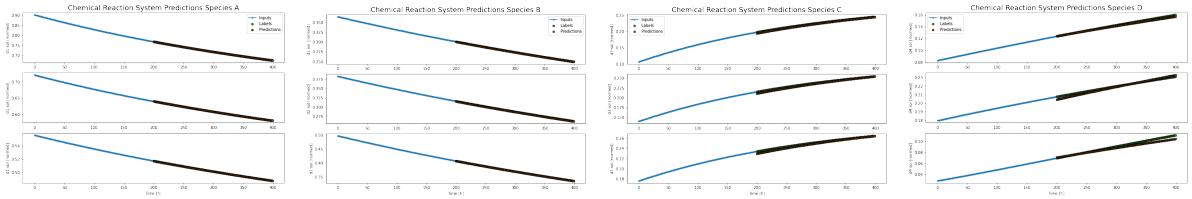
For Model 2, we use the same equations, but view the problem in a slightly different way which results in different visualizations of the solutions. For this model, the focus was on developing an approach that could output the entire prediction sequence in a one-shot manner, where multiple inputs lead to multiple outputs at a time. With this method, since the generated slices of data were less than the total training data set, there were multiple slices created in the target data to test the model on. With this in mind, we graphed 3 prediction solutions per system to see how the model did at different points in the testing data. Below are the graphs of a slice of the target data that the solution was predicted upon which is superimposed on the exact solutions for reference.



For Model 2, the 1st order ODE prediction had 0.121% mean relative point-wise error, the 2nd order ODE system a.) had a 7.04% mean relative point-wise error, and the 2nd order ODE system b.) had a 6.77% mean relative point-wise error. While slightly worse for the second order ODE b.), this One-Shot model does comparatively well based on Model 1. Similarly to model 1, the first order equation is seen as a simple working example, and then two second order equations were slightly extended working examples to challenge the model a bit more.

3.4 Model 2 on Coupled Systems of ODEs

For the second model, as the goal was one shot predictions, we were able to implement the coupled system of 4 ODEs that modeled a chemical reaction as 4 separate equations, but not as vector solutions since our model was specific to a single input, for one-shot solutions. Because of the fact that this system is now essentially more singular ODEs, we only consider the results from the coupled system modeling a chemical reaction between 4 chemicals, and leave out the Lotka-Volterra system from the results. For the chemical reaction system, the 4 species solutions were predicted in the one shot manner and the graphics below display 3 different slices of each of the four species solutions that trained on the training data.



The model did quite well on the above ODEs with errors being 0.235%, 0.328%, 0.803%, and 2.24% respectively for chemicals A, B, C, and D. Once we had these slices, we thought it was beneficial to be able to visualize what these predictions look like on the graph of the 4 coupled solutions. Below displays the figure of the solutions with 4 one shot predictions each corresponding to a the solutions to the ODEs of each chemical in the chemical reaction.

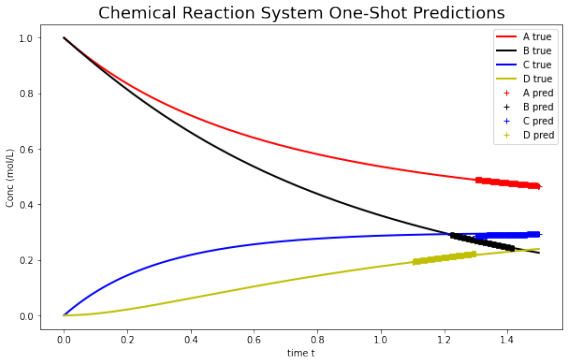


Figure 3: One-Shot Predictions of each chemical species superimposed on the true solutions

Being able to implement all four ODEs at once into the model would be an interesting next step for the future directions of this project. This would involve a new model that can take in multiple inputs, output multiple outputs, all in the one shot manner that the model currently does for singular ODEs.

4 Discussion

RNN models have shown to have remarkable learning power when it comes to sequential data. In looking at the connections between RNNs and ODEs, we can see that RNNs that complete time-series predictions have lots of potential for being able to learn the dynamics of differential equations of future time-steps. Throughout this study, we considered two models that, through different approaches, successfully predicted the future solutions of ODEs.

4.1 Model Comparison

Model 1 used a recursive multi-step approach in which solutions were predicted using both the training data, as well as sampling from the predictions of prior time steps and using them as the input for the prediction at the following time steps. This model did well with both singular ODEs as well as systems of multiple coupled ODEs, and was able to be modified to take in vectors as inputs instead of singular values.

On the other hand, Model 2 used a one-shot multi-step approach in which the entire solution is predicted at the same time. This model was more complex as there were dependencies being learned between inputs and outputs as well as between outputs and outputs. Model 2 did well with the singular ODEs, but could not take in vectors while keeping the same model architecture. Thus, looking at the coupled ODE systems, they were analyzed separately per individual solution, and put together after training was completed. Looking at just the chemical reaction system of 4 ODEs, the model did very well on these equations just as it did on the other singular ODEs.

4.2 Future Directions

With such a broad and expandable topic, there is an unlimited number of future directions that this project could be expanded to. The first direction that could be taken is to try to create a multi-input one-shot model that successfully completes the task that Model 1 did, but in the approach that Model 2 took predicting the entire solution in a one shot manner. Our investigation into the abilities of various approaches of LSTMs to predict certain systems of ODEs can hopefully be used as a guide to expand their ideas to real world applications and data. Another interesting direction that one could take is to look at architectures of RNNs that resemble the growing topic of Neural ODEs that use numerical ODE solver methods like Runge Kutta to inspire the architecture and investigate this type of architecture's ability to predict differential equation solutions [13]. With real world applications in mind, we could also consider looking at solutions of Partial Differential Equations (PDE) such as the acoustic wave equation where solutions would be 2-dimensional solution snapshots of acoustic waves [6].

5 Conclusion

In the conclusion of this summer research experience, I would also like to reflect on the summer and program as a whole. Throughout the 10 weeks, I have been repeatedly amazed at how much I am learning, and how quickly time can go by when being challenged everyday. As an Applied Mathematics major, one of my main goals for the summer was to get immersed into the field of Computational Mathematics and Sciences, and I don't think I could have chosen a better program to meet those goals. At the beginning of the summer, the focus of my internship was making these connections between my mathematical point of view and the computational point of view of the Pho-Ices Group. This project, while constantly challenging, was a great mixture of connecting my knowledge of differential equations and the mathematical theory behind stability, and the computer modeling aspects of RNNs that have complexities that cannot necessarily be explained by math alone. Coming to look at RNNs from a mathematical focus helped me recognize the truly interdisciplinary aspects of applied math, and get inspired to use mathematical theory to

solve real world problems in the future. I would like to thank my advisor Dr. Tan Bui-Thanh for constantly challenging me to think in different ways and for his wisdom, generosity, and patience with me throughout the summer. I would also like to thank the Oden Institute for its support of undergraduate students and willingness to modify and continue this incredible program even in these uncertain times.

References

[1] Muphy Yeuzhen Nui, Isaac Chuang, and Lior Horesh. *Recurrent Neural Networks in the Eye of Differential Equations.* arXiv:1904.12933, 2019.

[2] Mansure Habiba and Barak Pearlmutter. *Neural Ordinary Differential Equation Based Recurrent Neural Network Model.* Neural Information Processing Systems (NeurIPS), 2018.

[3] Bo Chang, Minmin Chen, Eldad Haber, and Ed Chi. *AntisymmetricRNN: A Dynamical System View on Recurrent Neural Networks.* International Conference on Learning Representations (ICLR), 2019.

[4] Ricky Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. *Neural Ordinary Differential Equations.* Neural Information Processing Systems (NeurIPS), 2018.

[5] Catherine Higham and Desmond Higham. *Deep Learning: An Introduction for Applied Mathematicians.* Society for Industrial and Applied Mathematics (SIAM), 2019)

[6] Hasim Sak, Andrew Senior, and Francoise Beaufays. *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling.* International Speech Communication Association, 2014.

[7] John Miller and Moritz Hardt. *Stable Recurrent Models.* ICLR, 2018.

[8] Martin Arjovsky, Amar Shah, and Yoshua Bengio. *Unitary evolution recurrent neural networks.* In ICML, pp. 1120–1128, 2016.

[9] TensorFlow Authors. *Tensorflow timeseries forecasting.* TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, 2015.

[10] Thomas Judson. *The Ordinary Differential Equations Project.* Department of Mathematics and Statistics, Stephan F. Austin State University, 2018.

[11] Thushan Ganegedara. *Stock Market Predictions with LSTM in Python.* DataCamp, 2020.

[12] Paul Dawkins. *Exact Equations of First Order DE's.* Pauls Online Math Notes - Lamar University, 2018.

[13] Coryn Bailer-Jones, David MacKay, and Philip Withers. *A Recurrent Neural Network for Modelling Dynamical Systems.* Computation in Neural Systems, 1998

[14] Kevin Burrage and J. C. Butcher. *Stability Criteria for Implicit Runge-Kutta Methods.* Society for Industrial and Applied Mathematics (SIAM), 1979.

6 Appendix A

Definitions

We define the general implicit Runge-Kutta method as

$$\vec{u}_n = \vec{u}_{n-1} + \delta \sum_{j=1}^{m} b_j f(t_{n-1} + c_j\delta, \vec{v}_j)$$
$$\vec{v}_i = \vec{u}_{n-1} + \delta \sum_{j=1}^{m} a_{ij} f(t_{n-1} + c_j\delta, \vec{v}_j) \qquad \text{for } n, i = 1...m$$

Where $\vec{v}_i$ denotes the intermediate approximations used in the computation of $\vec{u}_n$ from $\vec{u}_{n-1}$.

The coefficients a_{ij}, b_j, and c_j $(i, j = 1 \cdots m)$ are arranged in the Butcher tableau

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1m} \\ c_2 & a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_m & a_{m1} & a_{m2} & \dots & a_{mm} \\ \hline & b_1 & b_2 & \dots & b_m \end{array}$$

which is a simple mnemonic device for specifying a Runge–Kutta method.

We define the symmetric positive semi-definite matrix M with components of the form

$$M = m_{ij} = b_i a_{ij} + b_j a_{ij} - b_i b_j$$

We also define test equations of the form

$$y'(x) = f(x, y(x)), \quad f : \mathbb{R}^{N+1} \to \mathbb{R}^N, \tag{1}$$

satisfying the monotonicity condition where the inner product between variable vector and function vector is less than or equal to zero

$$\langle f(x, y) - f(x, z), y - z \rangle \leq 0 \tag{2}$$

for all $y, z \in \mathbb{R}^N$ and $x \in \mathbb{R}$

Definition. Implicit Runge-Kutta method is BN-stable if for two solution sequences $\cdots, \hat{u}_{n-1}, \hat{u}_n \cdots$ and $\cdots, u_{n-1}, u_n \cdots$ applied to any problem (1) where (2) holds, $||\hat{u}_n - u_n|| \leq ||\hat{u}_{n-1} - u_{n-1}||$.

Lemma 1. If $a_{ij}, b_j \in \mathbb{R}$, then $\sum_{i=1}^{m} \sum_{j=1}^{m} b_i a_{ij} = \sum_{i=1}^{m} \sum_{j=1}^{m} b_j a_{ji}$

Proof. Observe that

$$\begin{aligned}\sum_{i=1}^{m}\sum_{j=1}^{m} b_i a_{ij} =& b_1a_{11} + b_1a_{12} + \cdots b_1a_{1m} \\ & b_2a_{21} + b_2a_{22} + \cdots b_2a_{2m} \\ & b_3a_{31} + b_2a_{32} + \cdots b_3a_{3m} \\ & \cdots\cdots + \cdots\cdots + \cdots\cdots + \\ & b_ma_{m1} + b_ma_{m2} + \cdots b_ma_{mm}\end{aligned}$$

and

$$\begin{aligned}\sum_{i=1}^{m}\sum_{j=1}^{m} b_j a_{ji} =& b_1a_{11} + b_2a_{21} + \cdots b_ma_{m1} \\ & b_1a_{12} + b_2a_{22} + \cdots b_ma_{m2} \\ & b_1a_{13} + b_2a_{23} + \cdots b_ma_{m3} \\ & \cdots\cdots + \cdots\cdots + \cdots\cdots + \\ & b_1a_{1m} + b_2a_{2m} + \cdots b_ma_{mm}\end{aligned}$$

We can see, through simple observation, that these two equations produce the same output. Therefore, $\sum_{i=1}^{m}\sum_{j=1}^{m} b_i a_{ij} = \sum_{i=1}^{m}\sum_{j=1}^{m} b_j a_{ji}$.

□

Theorem 1. If the implicit Runge-Kutta method satisfies $b_1, b_2, \cdots b_m \geq 0$, and M is positive semi-definite, then the method is BN-stable.

Proof. Let $\vec{y}_n = \hat{u}_n - u_n$, $\vec{K}_j = \hat{v}_j - v_j$, $\vec{y}_0 = \hat{u}_{n-1} - u_{n-1}$, $\vec{w}_j = \delta(f(t_{n-1}+c_j\delta, \hat{v}_j) - f(t_{n-1}+c_j\delta, v_j)$ for $n, j = 1 \cdots m$.

Taking the difference between Runge-Kutta formulas for $\hat{u}_n$ and u_n gives

$$\vec{K}_i = \vec{y}_o + \sum_{j=1}^{m} a_{ij}\vec{w}_j$$

$$\vec{y}_n = \vec{y}_0 + \sum_{j=1}^{m} b_j\vec{w}_j.$$

Multiplying each side by itself in $\vec{y}_n$ gives

$$\begin{aligned}\vec{y}_n \cdot \vec{y}_n &= (\vec{y}_o + \sum_{j=1}^{m} a_{ij}\vec{w}_j) \cdot (\vec{y}_o + \sum_{j=1}^{m} a_{ij}\vec{w}_j) \\ &= \vec{y}_0 \cdot \vec{y}_0 + (\sum_{j=1}^{m} a_{ij}\vec{w}_j) \cdot \vec{y}_0 + (\sum_{j=1}^{m} a_{ij}\vec{w}_j) \cdot \vec{y}_0 + (\sum_{j=1}^{m} a_{ij}\vec{w}_j) \cdot (\sum_{j=1}^{m} a_{ij}\vec{w}_j).\end{aligned}$$

Simplifying both the right hand side and the left hand side component-wise gives

$$(\vec{y}_n) \cdot (\vec{y}_n) = \sum_{i=1}^{m} y_{ni} \cdot y_{ni} = \sum_{i=1}^{m} y_{ni}^2 = y_{n1}^2 + y_{n2}^2 + \cdots = ||\vec{y}_n||^2$$

$$(\vec{y}_0) \cdot (\vec{y}_0) = \sum_{i=1}^{m} y_{0i} \cdot y_{0i} = \sum_{i=1}^{m} y_{0i}^2 = y_{01}^2 + y_{02}^2 + \cdots = ||\vec{y}_0||^2$$

$$(\sum_{j=1}^{m} a_{ij}\vec{w}_j) \cdot (\vec{y}_0) = (b_1\vec{w}_1) \cdot (\vec{y}_0) + (b_2\vec{w}_2) \cdot (\vec{y}_0) + \cdots = b1(\vec{w}_1 \cdot \vec{y}_0) + b2(\vec{w}_2 \cdot \vec{y}_0) \cdots = \sum_{i=1}^{m} b_i\langle \vec{y}_0, \vec{w}_i \rangle = \sum_{i=1}^{m} b_i \vec{y}_0 \cdot \vec{w}_i$$

$$(\sum_{j=1}^{m} a_{ij}\vec{w}_j) \cdot (\sum_{j=1}^{m} a_{ij}\vec{w}_j) = \sum_{i=1}^{m}\sum_{j=1}^{m} b_i\vec{w}_i \cdot b_j\vec{w}_j = \sum_{i=1}^{m}\sum_{j=1}^{m} b_i b_j \vec{w}_i \cdot \vec{w}_j.$$

Thus,

$$||\vec{y}_n||^2 = ||\vec{y}_0||^2 + 2\sum_{i=1}^{m} b_i \vec{y}_0 \cdot \vec{w}_i + \sum_{i=1}^{m}\sum_{j=1}^{m} b_i b_j \vec{w}_i \cdot \vec{w}_j$$

.

Note that since all values are real, inner products are the same at standard vector multiplication and hence $\langle x, y\rangle = x \cdot y = y \cdot x = y, x\rangle$.

Now take the inner product of $\vec{K}_i$ and $\vec{w}_i$:

$$\begin{aligned}\vec{K}_i \cdot \vec{w}_i &= (\vec{y}_0 + \sum_{j=1}^{m} a_{ij}\vec{w}_j) \cdot \vec{w}_i \\ &= \vec{y}_0 \cdot \vec{w}_i + \sum_{j=1}^{m} a_{ij}\vec{w}_j\vec{w}_i.\end{aligned}$$

Solving for $\vec{y}_0 \cdot \vec{w}_i$ gives

$$\vec{y}_0 \cdot \vec{w}_i = \vec{K}_i \cdot \vec{w}_i - \sum_{j=1}^{m} a_{ij}\vec{w}_j\vec{w}_i$$

and substituting what we just determined for $\vec{y}_0 \cdot \vec{w}_i$ into $||\vec{y}_n||^2$ then returns

$$||\vec{y}_n||^2 = ||\vec{y}_0||^2 + 2\sum_{i=1}^{m} b_i \left(\vec{K}_i \cdot \vec{w}_i - \sum_{j=1}^{m} a_{ij}\vec{w}_j\vec{w}_i \right) + \sum_{i=1}^{m}\sum_{j=1}^{m} b_i b_j \vec{w}_i \cdot \vec{w}_j$$

. Thus

$$||\vec{y}_n||^2 = ||\vec{y}_0||^2 + 2\sum_{i=1}^{m} b_i \vec{K}_i\vec{w}_i - 2\sum_{i=1}^{m}\sum_{j=1}^{m} b_i a_{ij}\vec{w}_i\vec{w}_j + \sum_{i=1}^{m}\sum_{j=1}^{m} b_i b_j \vec{w}_i\vec{w}_j. \tag{3}$$

By Lemma 1, we have established that $\sum_{i=1}^{m}\sum_{j=1}^{m} b_i a_{ij} = \sum_{i=1}^{m}\sum_{j=1}^{m} b_j a_{ji}$, and since $\vec{w}_i \cdot \vec{w}_j$ does not change value when the dummy variable is changed, we can also conclude that $\sum_{i=1}^{m}\sum_{j=1}^{m} b_i a_{ij}\vec{w}_i\vec{w}_j = \sum_{i=1}^{m}\sum_{j=1}^{m} b_j a_{ji}\vec{w}_i\vec{w}_j$. Now, looking at the last two components of the right hand side of equation (3), we can rearrange as follows:

$$\begin{aligned}
-2\sum_{i=1}^{m}\sum_{j=1}^{m} b_i a_{ij}\vec{w}_i\vec{w}_j + \sum_{i=1}^{m}\sum_{j=1}^{m} b_i b_j \vec{w}_i\vec{w}_j &= -\left(\sum_{i=1}^{m}\sum_{j=1}^{m} b_i a_{ij}\vec{w}_i\vec{w}_j + \sum_{i=1}^{m}\sum_{j=1}^{m} b_j a_{ji}\vec{w}_i\vec{w}_j - \sum_{i=1}^{m}\sum_{j=1}^{m} b_i b_j \vec{w}_i\vec{w}_j\right) \\
&= -\left(\sum_{i=1}^{m}\sum_{j=1}^{m} b_i a_{ij}\vec{w}_i\vec{w}_j + b_j a_{ji}\vec{w}_i\vec{w}_j - b_i b_j \vec{w}_i\vec{w}_j\right) \\
&= -\left(\sum_{i=1}^{m}\sum_{j=1}^{m} (b_i a_{ij} + b_j a_{ji} - b_{ij}) \cdot \vec{w}_i\vec{w}_j\right) \\
&= -\left(\sum_{i=1}^{m}\sum_{j=1}^{m} m_{ij} \cdot \vec{w}_i\vec{w}_j\right).
\end{aligned}$$

Thus,

$$||\vec{y}_n||^2 = ||\vec{y}_0||^2 + 2\sum_{i=1}^{m} b_i\vec{K}_i \cdot \vec{w}_i - \sum_{i,j=1}^{m} m_{ij}\vec{w}_i \cdot \vec{w}_j$$

.

For stability to hold, we must show that $2\sum_{i=1}^{m} b_i\vec{K}_i \cdot \vec{w}_i - \sum_{i,j=1}^{m} m_{ij}\vec{w}_i \cdot \vec{w}_j \leq 0$.

For the first term, $2\sum_{i=1}^{m} b_i\vec{K}_i \cdot \vec{w}_i \leq 0$ because of the conditions $j \geq 0$ and the monotonicity condition set on f is exactly $\vec{K}_i \cdot \vec{w}_i \leq 0$.

For the second term, we want to show that $\sum_{i,j=1}^{m} m_{ij}\vec{w}_i \cdot \vec{w}_j \geq 0$. Recall that M is a symmetric positive-semi-definite matrix with components m_{ij}.

Let W be the $m \times m$ matrix with components $W_{ij} = \vec{w}_i \cdot \vec{w}_j$. Observe that since M is real,

$$\begin{aligned}
\sum_{i,j=1}^{m} m_{ij}\vec{w}_i \cdot \vec{w}_j &= \sum_{i,j=1}^{m} m_{ij} \cdot W_i j \\
&= tr(M^T W) \\
&= \langle M, W\rangle_F
\end{aligned}$$

where $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius norm. First, we show that matrix W is positive semi-definite.

Let V be a matrix whose rows are $\vec{w}_i$, then $W = VV^T$. W is said to be positive semi-definite if the scalar $x^T W x$ is non-negative for every real, non-zero column vector x. Note that $x^T W x = x^T V V^T x$

and thus it holds that

$$\begin{aligned} x^T V V^T x &= (V^T x)^T (V^T x) \\ &= (V^T x)^2 \\ &= \left(\sum_{i=1}^{m} x_i w_i \right)^2 \\ &= ||V^T x||_F^2 \geq 0. \end{aligned}$$

Therefore, matrix $W = VV^T$ is positive semi-definite. Since W and M are both positive semi-definite matrices, we can prove that $\sum_{i,j=1}^{m} m_{ij} \cdot W_{ij} \geq 0$ as follows.

Observe that $\langle M, W \rangle_F = Tr(MW^T) = Tr(M^{\frac{1}{2}} M^{\frac{1}{2}} W^{\frac{1}{2}} W^{\frac{1}{2}})$, and through the cyclical properties of Trace, $Tr(M^{\frac{1}{2}} M^{\frac{1}{2}} W^{\frac{1}{2}} W^{\frac{1}{2}}) = Tr(M^{\frac{1}{2}} W^{\frac{1}{2}} M^{\frac{1}{2}} W^{\frac{1}{2}})$.

Let matrix $A = M^{\frac{1}{2}} W^{\frac{1}{2}}$. Since the square root of a positive semi-definite matrix is symmetric,

$$\begin{aligned} Tr(M^{\frac{1}{2}} W^{\frac{1}{2}} M^{\frac{1}{2}} W^{\frac{1}{2}}) &= Tr(AA^T) \\ &= ||A||_F^2 \geq 0. \end{aligned}$$

Thus, $\sum_{i,j=1}^{m} m_{ij} \vec{w}_i \cdot \vec{w}_j \geq 0$. Hence, $||\vec{y}_n||^2 \leq ||\vec{y}_0||^2$ and therefore $||\hat{u}_n - u_n|| \leq ||\hat{u}_{n-1} - u_{n-1}||$ which is the required result for the BN-stability of the method to hold.

□