AM205: Final Project Report

Jiawei Zhuang, Xu Si, Rui Zhao, Chin Hui Chew

December 2017

1 Introduction

(to add: application of ODEs, e.g. population growth, chemical kinetics)

Ordinary differential equations (ODEs) are generally solved by finite-differencing methods, from the simplest forward Euler scheme to higher-order schemes like the Runge-Kutta methods. Numerical solutions obtained by those schemes are typically stored in a discretized form, i.e. in an array of floating point numbers.

The artificial neural network (ANN) method [**?**] provides an way to obtain ODE solutions in a closed analytical form. The solutions are stored as neural network parameters, which requires much less memory than storing the solution as a discretized array. Also, because the solution is analytically differentiable, it can be superior in some applications like sensitivity analysis.

2 Method

2.1 Fitting functions by artificial neural networks

The universal approximation theorem [**?**] states that any continuous function can be approximated by a feed-forward neural network with a single hidden layer. This ANN can be written in a matrix multiplication form:

$$N(x; w) = W_2\sigma(W_1 x + b_1) + b_2 \tag{1}$$

where W_1 and W_2 are weight matrices and b_1 and b_2 are bias terms. σ is a nonlinear activation function such as *tanh*. We use w to represent all parameters [W_1, W_2, b_1, b_2].

To fit a scalar function $y(x)$, the neural network takes a scalar input x and returns a scalar output $N(x)$. In this case, W_1 and W_2 degrade to row and column vectors.

The optimal parameters w can be found by minimizing the loss function

$$L(w) = \int_a^b [y(x) - N(x; w)]^2 dx \tag{2}$$

In practice, the integral is approximated by a summation

$$L(w) = \sum_i [y(x_i) - N(x_i; w)]^2 \tag{3}$$

where $\{x_i\}$ is a set of training points covering the domain $[a, b]$.

If the loss is small enough, then the ANN can be considered as a good approximation to the original function over the domain the domain $[a, b]$:

$$N(x; w) \approx y(x) \tag{4}$$

2.2 Solving one ODE by artificial neural networks

Now we consider constructing an ANN that can approximate the solution to the first-order ODE:

$$y'(t) = F(y(t), t), \;\; y(t_0) = y_0 \tag{5}$$

If we use a standard neural network

$$N(t; w) = W_2\sigma(W_1 t + b_1) + b_2 \tag{6}$$

It will not satisfy the initial condition, i.e. typically $N(t_0; w) \neq y_0$. But we can force the initial condition by rewriting the ANN solution as

$$\hat{y}(t; w) = y_0 + (t - t_0)N(t; w) \tag{7}$$

For any parameters w, there will always be $\hat{y}(t_0; w) = y_0$. We further require this ANN solution $\hat{y}(t; w)$ to satisfy the ODE:

$$\hat{y}'(t; w) \approx F(\hat{y}(t; w), t) \tag{8}$$

Note that the derivative $\hat{y}'(t; w)$ can be derived analytically without any finite-difference approximation

$$\hat{y}'(t; w) = \frac{\partial[y_0 + (t - t_0)N(t; w)]}{\partial t} = \frac{\partial(t - t_0)}{\partial t} N(t; w) + (t - t_0)\frac{\partial N(t; w)}{\partial t} \tag{9}$$

The optimal parameters can be found by minimizing the cost function

$$L(w) = \int_{t_0}^{t_1} [\hat{y}'(t; w) - F(\hat{y}(t; w), t)]^2 dt \tag{10}$$

Or in practice,

$$L(w) \approx \sum_i [\hat{y}'(t_i; w) - F(\hat{y}(t_i; w), t_i)]^2 \tag{11}$$

where $\{t_i\}$ is a set of training points covering the domain $[t_0, t_1]$.

If the loss is small enough, then the ANN solution should be able to approximate the true ODE solution over the domain $[t_0, t_1]$:

$$\hat{y}(t; w) \approx y(t) \tag{12}$$

2.3 Solving ODE systems by artificial neural networks

The above ANN method can be directly generalize to a system of ODEs. For simplicity, consider two ODEs

$$y'(t) = F_1(y(t), z(t), t), \ \ y(t_0) = y_0 \tag{13}$$

$$z'(t) = F_2(y(t), z(t), t), \ \ z(t_0) = z_0 \tag{14}$$

We can use two separates ANNs for two variables y and z

$$\hat{y}(t; w) = y_0 + (t - t_0)N_1(t; w_1) \tag{15}$$

$$\hat{z}(t; w) = z_0 + (t - t_0)N_2(t; w_2) \tag{16}$$

Then loss function is the sum of losses for two ODEs

$$L(w_1, w_2) \approx \sum_i \left[[\hat{y}'(t_i) - F_1(\hat{y}(t_i), \hat{z}(t_i), t_i)]^2 + [\hat{z}'(t_i) - F_2(\hat{y}(t_i), \hat{z}(t_i), t_i)]^2 \right] \tag{17}$$

3 Results and Discussions

In this section, we will apply the Artificial neural networkmethod to four common types of ODE problems: (1)simple ODE, (2)unstable and periodic ODE, (3)unstable and chaotic ODE and (3)stiff ODE system and then compare Neural Network results with traditional finite difference method which we treat as ground truth. We have found that in most cases, ANN method can approximate the ground truth well, whereas sometimes we need to train the network multiple times to get ideal results. Also, our method only works for a given range of data, yielding unreliable fits in extrapolation.

In all the examples, we train our ANN using BFGS whose performance of convergence out beats all other optimisers. We also compare ground truth and ANN fit both in the training domain and outside domain. To test the stability of our ANN method, we retrain the ANN 100 times and plot average RMSE

3.1 Simple ODEs

Overview

Consider a first-order ODE system:

$$\frac{dx}{dt} = \cos t + x^2 + y - (1 + t^2 + \sin^2 t) \tag{18}$$

$$\frac{dy}{dt} = 2t - (1 + t^2) \sin t + xy \tag{19}$$

where $t \in [0, 3]$.

Mathematics

The analytical solutions are:

$$x = \sin t \tag{20}$$

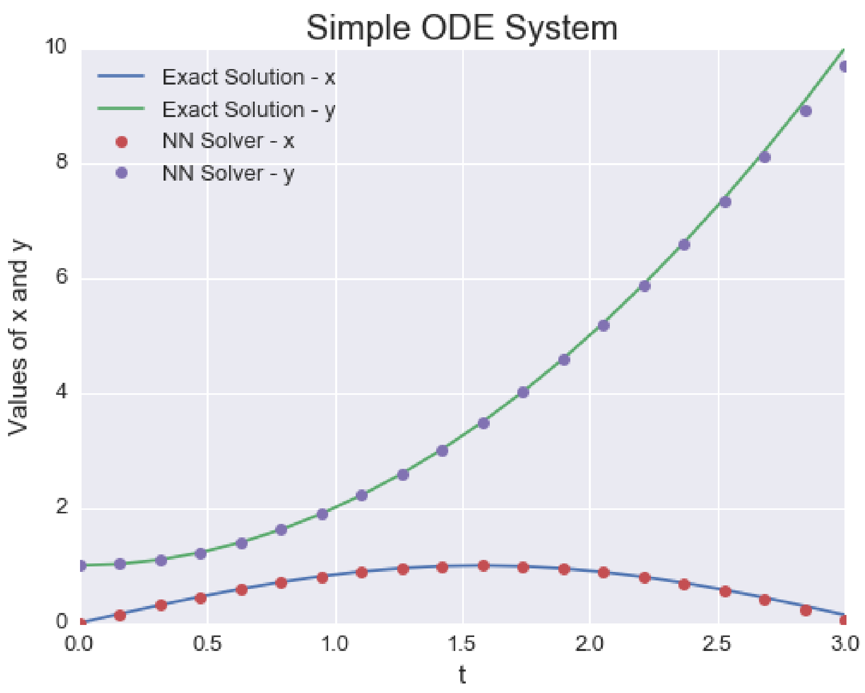

Figure 1: Comparing solutions found using neural net solver versus exact analytical solutions. Average RMSE = 0.0524.

$$y = 1 + x^2 \tag{21}$$

Ideally, our NN solver should attain the same solution as the exact analytical solutions within the time range specified.

Results

The solutions using the Neural Net solver could be found in Fig. 1. The network was trained using a grid of 20 equidistant points in $t = [0, 3]$ and 50 hidden units in the hidden layer.

We could see almost an exact fit with the analytical solution. In terms of training convergence, we could see that the neural net rapidly converges to a solution with log loss value of order 10^{-4} at 200 iterations from Fig. 2. BFGS is selected as the optimization method because it results in the lowest loss at 200 iterations.

Discussions

To enable us to evaluate the fit of the two functions in this system of ODE, we propose taking an average of the RMSE, in this case we will have:

$$RMSE_{overall} = \frac{1}{2}[RMSE_x + RMSE_y] \tag{22}$$

The fitting attained as shown in Fig. 1 has $RMSE_{overall} = 0.0524$.

Note that average RMSE enables us to determine the reproducibility of a good performing model for the same ODE system, but it is not necessary a good metrics to compare the fit for different ODE systems.

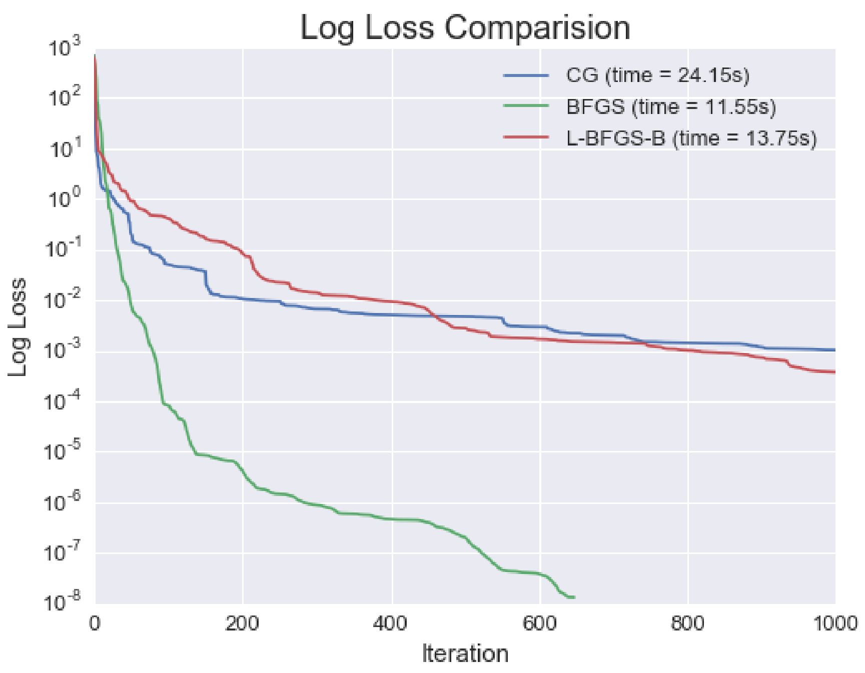

Figure 2: Comparison across different optimization methods for simple ODE system.

In terms of reproducibility, we repeated the same fitting process 100 times. The average RMSE histogram distribution for 100 fittings is shown in Fig. 3. 55% of the fittings are relatively close fit to the exact analytical solution, with RMSE of up to 0.0378, while our plot in Fig. 1 has RMSE = 0.0524.

Please refer to SimpleODE.ipynb for the code and visualizations of this section.

3.2 Lotka-Volterra 2-species model

Overview

The Lotka-Volterra prey-predator equations are a pair of first-order, non-linear differential equations commonly used to describe the interactions between two species, the prey and predator.

Mathematics

The populations of the prey and predator can be described by the following equations:

$$\frac{dx}{dt} = ax - bxy \tag{23}$$

$$\frac{dy}{dt} = -cy + dxy \tag{24}$$

where

- x is the density of prey
- y is the density predator
- t represents time

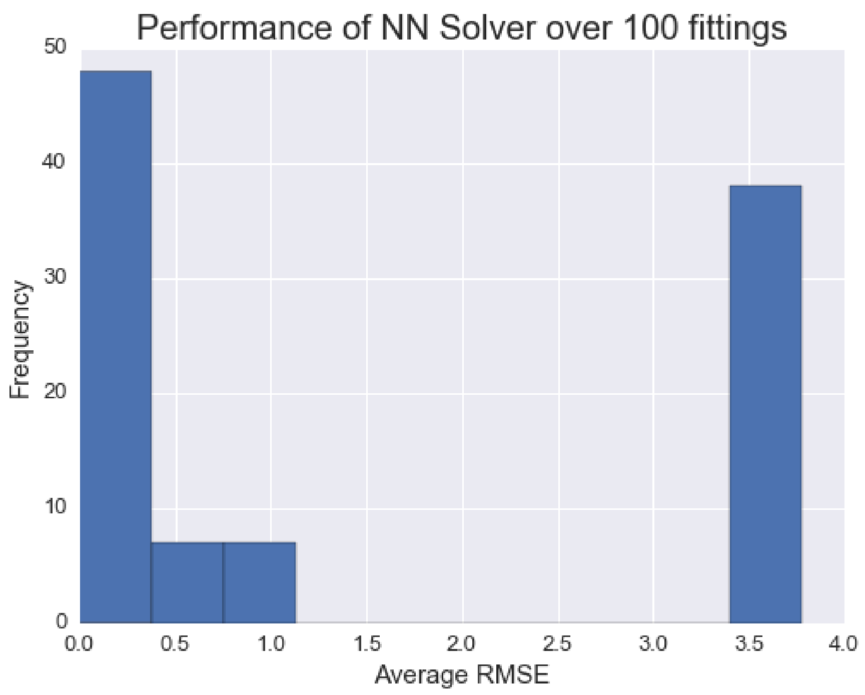

Figure 3: Average RMSE Distribution for 100 Repeated Fittings for Simple ODE System.

- $a, b, c, d > 0$

Eq. 6 indicates that in the absence of a predator ($y = 0$), the prey would grow at a constant rate a, with the assumption that the prey have an unlimited food supply. The rate of predation upon the prey is assumed to be proportional to the rate at which the predators and the prey are present concurrently, represented by bxy. If either x or y is zero then no predation is possible.

Similarly, Eq. 7 shows that in the absence of prey ($x = 0$), the density of predators would decrease at a constant rate c, due to natural death or emigration. This equation assumes that the predator population only preys upon the same species of prey identified in Eq. 6. dxy represents the growth of the predator population [**?**].

The model assumes that throughout the process the external environment remains the same and none of the species is favored [**?**].

Results

From Fig. 5 we see that our neural net solver attains a close fit to the ODE solver. Training was done with a grid of 100 equidistant points in $t = [0, 10]$ and 50 hidden units in the hidden layer.

In terms of training convergence, from Fig. 6, we can see that BFGS is selected as the optimization method as it results in the lowest loss compared to other methods.

Discussions

The same as before, to enable us to evaluate the fit of the two functions in this system of ODEs, we propose taking an average of the RMSE, in this case we will have:

$$RMSE_{overall} = \frac{1}{2}[RMSE_{prey} + RMSE_{predator}] \tag{25}$$

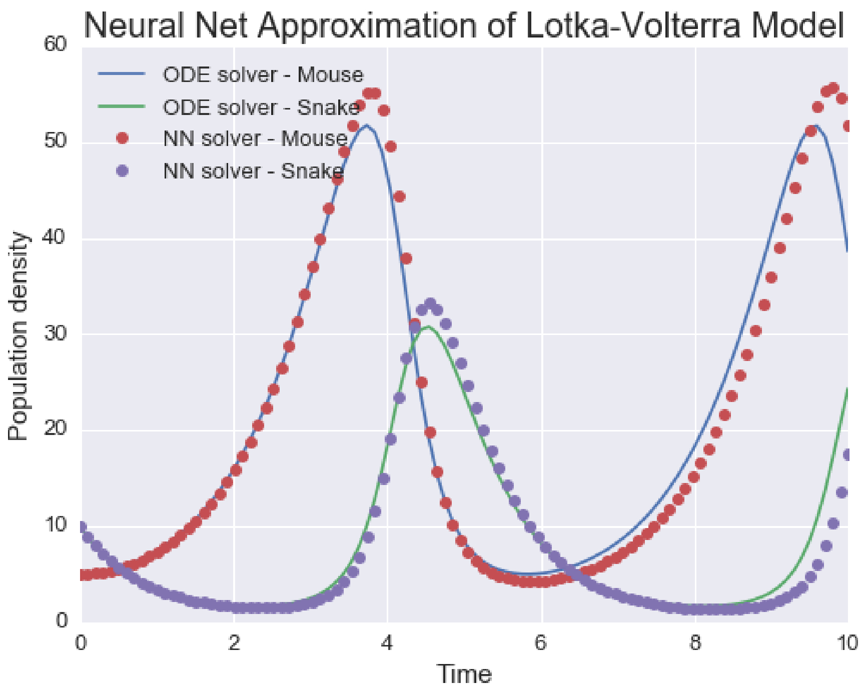

Figure 4: Comparing across results using standard ODE solver from Scipy library versus using neural net solver, at initial parameters of $a = 1, b = 0.1, c = 1.5, d = 0.75$. Average RMSE = 2.204.

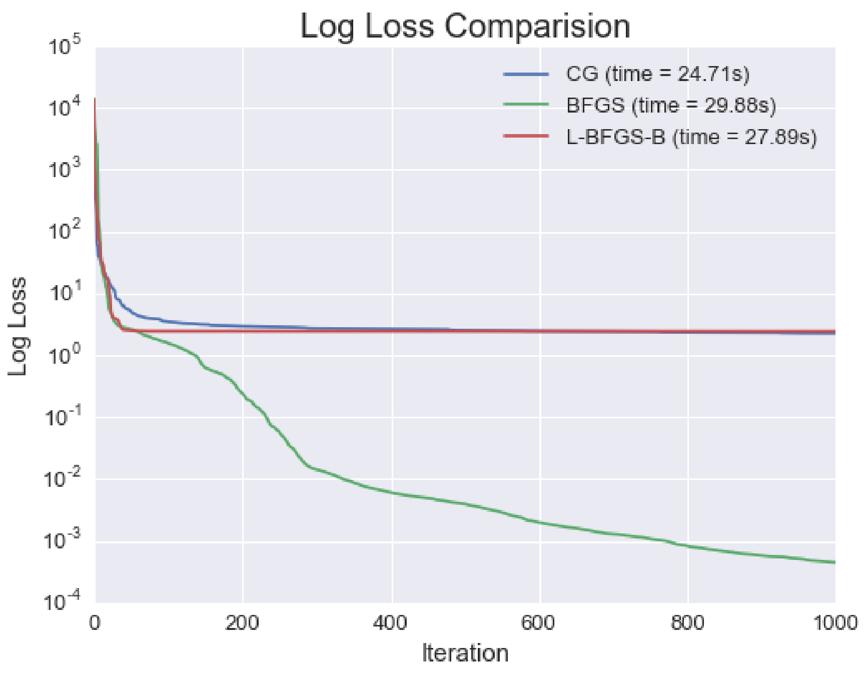

Figure 5: Convergence to final solutions for 2-species model

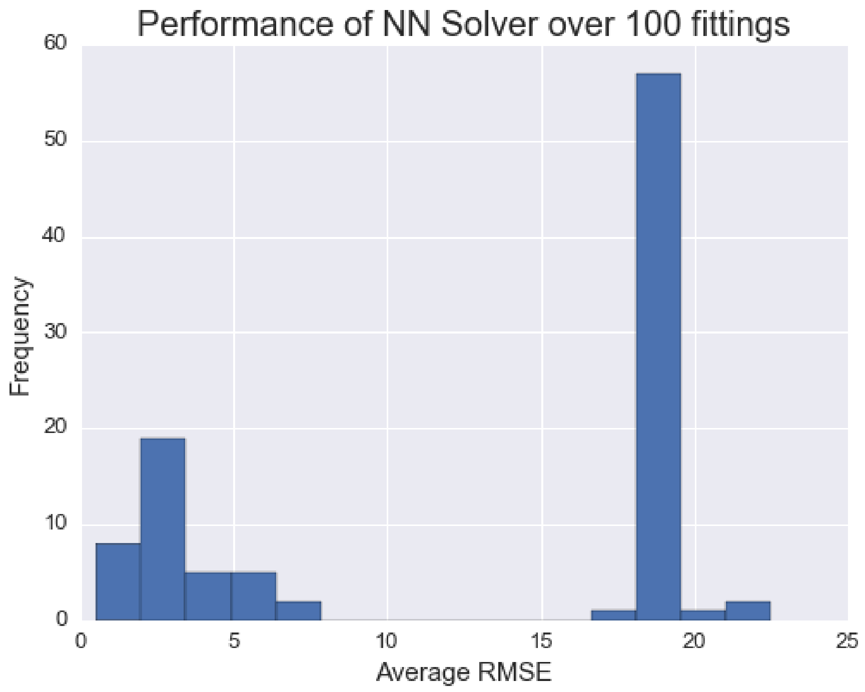

Figure 6: Running NN solver 100 times to see the performance histogram

For this particular plot, the overall average RMSE is at 2.204. It is found that while our proposed solver using neural net is able to provide a continuous function, the results tend to fluctuate highly. As such, we ran the same fitting over 100 times as shown in Fig. 7. 32% of the fittings result in relatively good fitting with average RMSE of less than 5.

Please refer to LV2.ipynb for the code and visualizations of this section.

3.3 Lotka-Volterra 3-species Model

Extending to the 2-species model discussed earlier, we wish to model a linear three-species food chain where the lowest-level prey x is preyed upon by a mid-level species y, which, in turn, is preyed upon by a top level predator z. An example of a three-species food chain is mouse-snake-owl [**?**].

$$\frac{dx}{dt} = ax - bxy \tag{26}$$

$$\frac{dy}{dt} = -cy + dxy - eyz \tag{27}$$

$$\frac{dz}{dt} = -fz + gyz \tag{28}$$

where

- $a, b, c, d, e, f, g > 0$

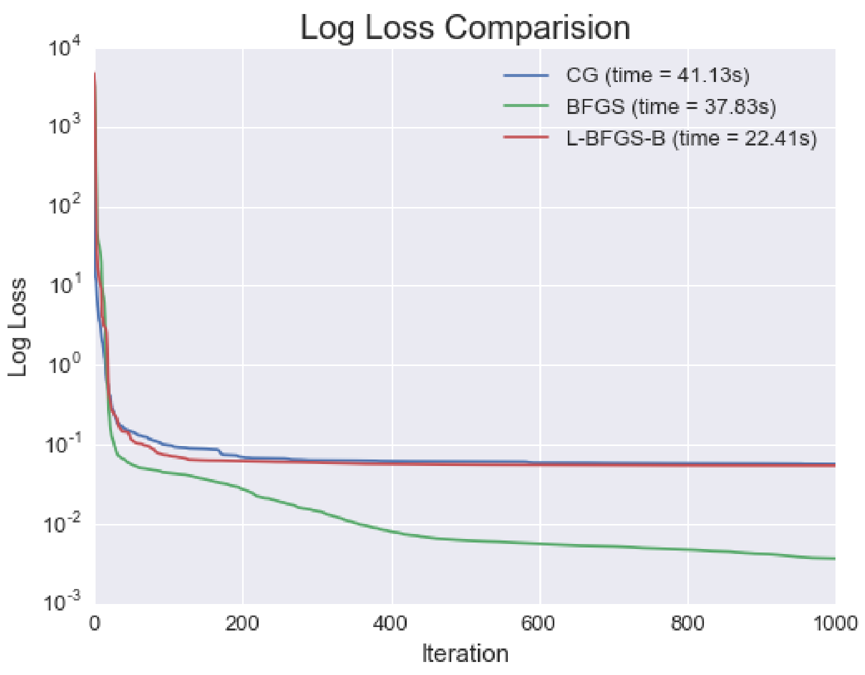

Figure 7: Convergence to final solutions for 3-species model

- a, b, c, d are as in the 2-species Lotka-Volterra equations
- e represents the effect of predation on species y by species z
- f represents the natural death rate of species z in the absence of prey
- g represents the reproduction rate of species z in the presence of prey y

Results

The neutral net is trained with 50 hidden units over 100 equidistant points over $t \in [0, 10]$. As before, the optimization method is BFGS because the log loss is the smallest compared to other methods at the 200 iterations as shown in Fig. **??**.

From Fig. 9, we can see that the performance is worse compared to the 2-species Lotka-Volterra model. The fit is close to the solution from ODE solver only up to $t = 2.5$ for the three equations. No further model fitting is carried out to examine reproducibility as the fitting is poor and the average RMSE score is not a good gauge to identify strong fit.

Please refer to LV3.ipynb for the code and visualizations of this section.

3.4 Van der Pol oscillator

Overview

Van der Pol oscillator is a non-conservative oscillator with a linear spring force and a non-linear damping force [**?**]. The equation is given by:

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = 0 \qquad (29)$$

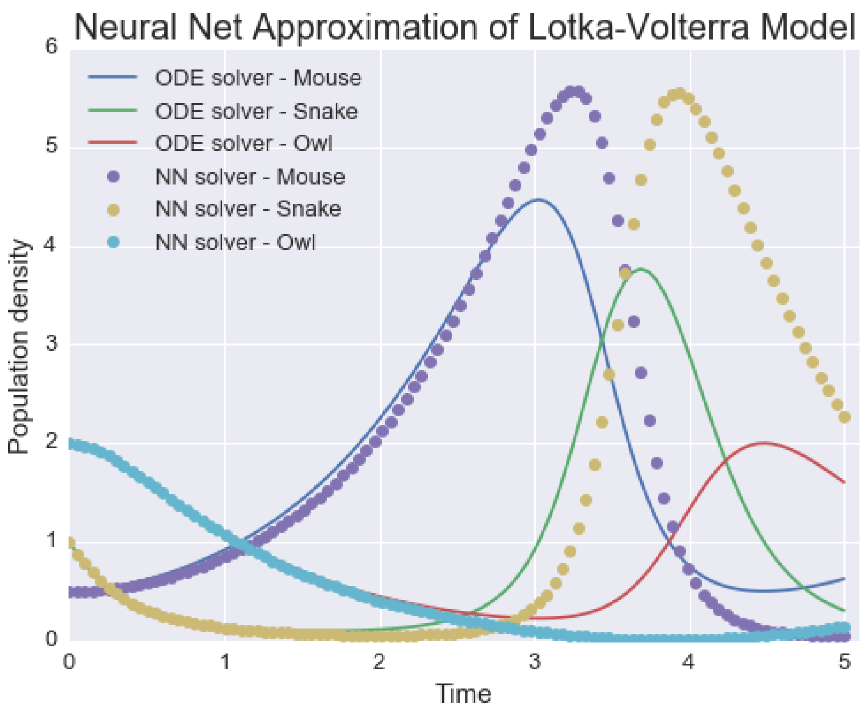

Figure 8: Comparing across results using standard ODE solver from Scipy library versus using neural net solver, at initial parameters of $a = b = c = d = e = f = g = 1$. Average RMSE score of 0.803.

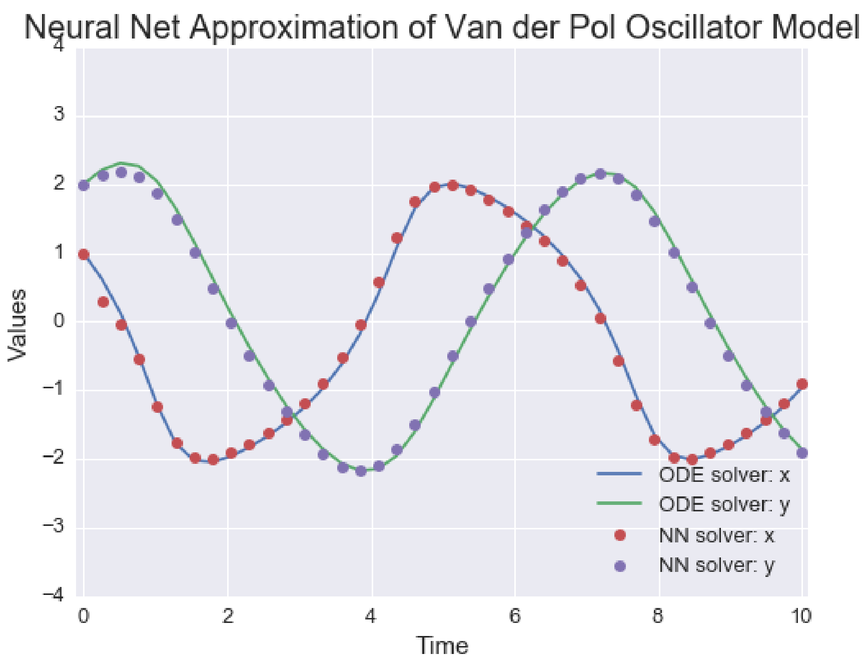

Figure 9: Running NN solver with even spacing on time points. Average RMSE of 0.0999.

Mathematics

Applying the Liénard transformation $y = x - \frac{x^3}{3} - \frac{1}{\mu}\frac{dx}{dt}$, the equation can be written as a system of ODE [**?**]:

$$\frac{dx}{dt} = \mu(x - \frac{1}{3}x^3 - y) \tag{30}$$

$$\frac{dy}{dt} = \frac{x}{\mu} \tag{31}$$

Results

From Fig. 11 we see that our neural net solver attains a close fit to the ODE solver. Training was done with a grid of 40 equidistant points in $t = [0, 10]$ and 20 hidden units in the hidden layer. BFGS is used as the optimization method as we could see from Fig. **??** that the method attains the lowest log loss.

Discussions

The same as before, to evaluate the fit of the two functions in this system of ODEs, we propose taking an average of the RMSE, in this case the overall average RMSE for our fit in Fig. 11 is at 0.0999.

Since the fitting is satisfactory, we run the same fitting over 100 times to ensure reproducibility. From Fig. 12, 76% of the fittings result in relatively good fitting with average RMSE of less than 3.2.

Please refer to Van_der_Pol_oscillator.ipynb for the code and visualizations of this section.

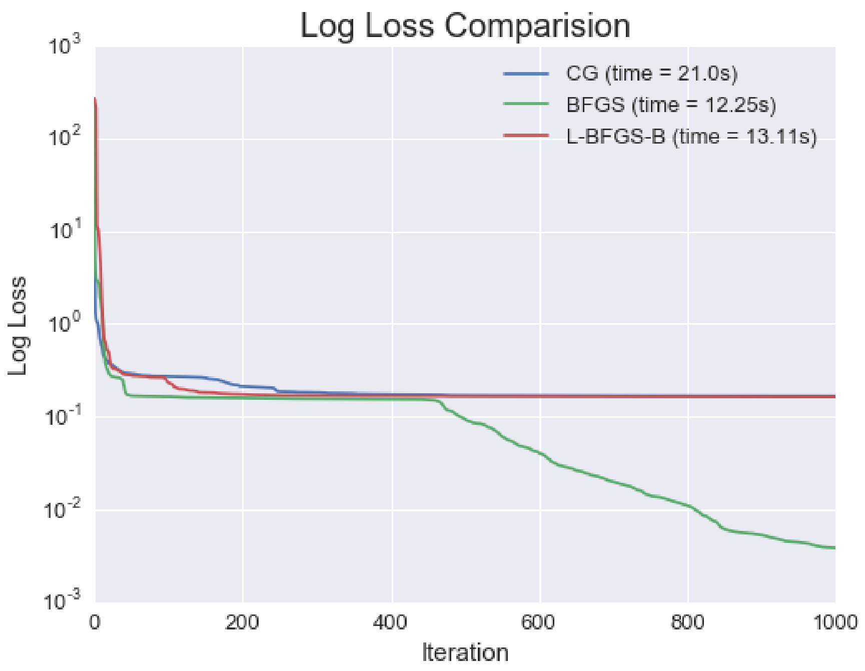

Figure 10: Convergence to final solutions for VDP.

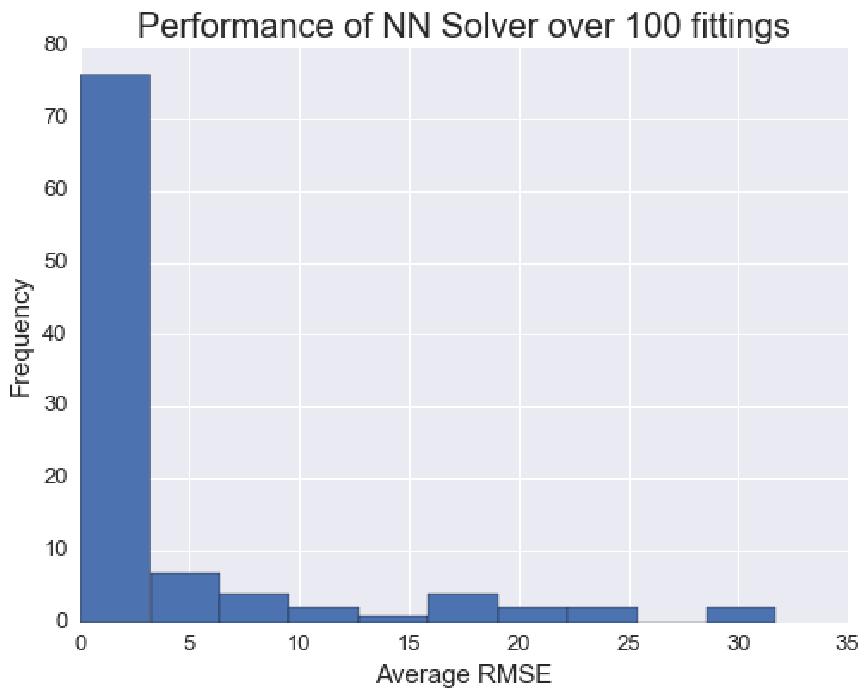

Figure 11: Running NN solver 100 times to see the performance histogram

3.5 Unstable, chaotic ODEs

3.5.1 Rssler Attractor

Introduction
Mathematics
Results Discussion

3.6 Stiff ODE system

3.7 Flexible initial conditions

4 Conclusions

Limitation: t cannot be every large; ANN has trouble fitting fast-oscillating solutions.

5 Reference

References

[1] Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. ”Multilayer feedforward networks are universal approximators.” Neural networks 2.5 (1989): 359-366.

[2] Lagaris, Isaac E., Aristidis Likas, and Dimitrios I. Fotiadis. ”Artificial neural networks for solving ordinary and partial differential equations.” IEEE Transactions on Neural Networks 9.5 (1998): 987-1000.

[3] https://en.wikipedia.org/wiki/Lotka-Volterra_equations

[4] http://www.math.harvard.edu/library/sternberg/slides/11809LV.pdf

[5] http://people.kzoo.edu/barth/math280/articles/3speciesLV.pdf

[6] http://www2.me.rochester.edu/courses/ME406/webexamp5/vanpol.pdf

[7] Kaplan, D. and Glass, L., Understanding Nonlinear Dynamics, Springer, 240244, (1995).