Neural Networks for Dynamical Systems

Echo Liu

June 15, 2020

Abstract

Neural Networks (NN) have been shown to be a very powerful tool to build non-linear model to conduct classification and prediction tasks, such as system control, object detection, sequence recognition. In this project, we use NN to help us make prediction on dynamic systems, Kuramoto-Sivashinsky (KS) equation, Lorenz equation and reaction-diffusion system. In Chaos theory, such as butterfly effect, a slight different initial state can bring very drastic change in later states. However, with the exact same initial state, the later state is determined. This task can be very difficult because it can be rather high-dimensional, and a good model for this system can be very complicated non-linear system. For those reasons, NN can be a very useful tool since it tackles such problems through activation functions. We thus try to use NN to build a model to make prediction on how the systems would evolve.

1 Introduction and Overview

1.1 Problem Description

A lot of times, dynamic systems are very difficult to predict. The butterfly effect, the underlying principle of chaos theory, has established that a small disturbance in the system can lead to very different state in the future, thus making the prediction very difficult. However, there should be an underlying system describing such chaotic behavior. Using the simplest Lorenz system as an example, such behavior is governed by system of ordinary differential equation. Some systems are not very easy to solve analytically, thus finding pattern from the data themselves is crucial.

1.2 General Approach

For each system, KS, Lorenz and reaction-diffusion, we first obtain multiple simulations of how the system evolves along time and within space, and then we embed them along time to create training input and output. Those data are then feed into certain NN for training. After training, we then obtain new data for testing with the same procedure to see how well the model behaves.

2 Theoretical Background

As we learned from our textbook [1], the essence of NN is optimization, since we hope to optimize all the parameters in the NN to minimize the error or maximize the accuracy, depending on the object function is defined, with certain regularization. The advantage of NN is that the model built can non-linear because of non-linear activation functions. To conduct optimization task, back-propagation is used, of which chain rule is the core idea to obtain gradients to update the model. With stochastic gradient descent, the optimization in global sense is made possible, meaning it's easier to get out of the local minimum or maximum. NN can also be very costly because of large quantity of parameters, and the constant improvement of hardware makes it possible to have better model. Different tasks require different structure of NN. For image processing tasks, Deep Convolutional Neural Network (DCNN) is used and for sequence prediction, Long Short Term Memory (LSTM) is used for its memory on old information. Within DCNN architecture, two main layers are used, convolutional layer and poolong layer. Each convolutional layer captures different feature of the object in image, and pooling layer is used to maximize or average the convolutional layer. Both of them are used to

further compress the data in images to prepare them for fully-connected layers later for classification task or some other work. For some data such as financial data, the data is auto-correlated, thus some information from a while before can still have impact on future state. Recurrent Neural Network, such as LSTM, can be used to deal with this type of situation. A lot of times, the date can be very high-dimensional, which makes it hard for NN to be trained. A way to reduce the dimensionality is to use autoencoder, which is another type of NN, designed to reduce dimensions.

A way to decompose data is

$$A = \hat{U}\hat{\Sigma}V^* \tag{1}$$

where $\hat{U}$ have orthonormal columns, V is an unitary matrix and $\hat{\Sigma}$ is a diagonal matrix. This can be used for any matrix A. The basic idea of SVD is to "stretch", "rotate" and "compress" the data, to make the variables independent. The idea of PCA here is that in this way, we are able to rank the variables based on how much variance they have, thus reducing the variables based on how important they are to the data.

An important thing during training and testing is cross-validation. This allows us to see different combination of training and testing dataset and see how well the algorithm behaves generally.

3 Algorithm Implementation and Development

1. Simulate KS equation and store data except the last time data points into `training input` and the data except the first time data points into `training output`
2. construct `net` to be NN with 3 hidden layers with size 10 and activation functions logsig, radbas, and purelin
3. feed `training input` and `training output` into `net` to start training
4. after training, simulate KS equation and use the first time data points to predict the future states, and plot to compare the two results
5. to compare the results with LSTM, construct input layers, two lstm layers with sequence output, followed by dropoutlater, one fully connected layer and regressionlayer
6. train network with the layers
7. use `training input` and `training output` before to train the new network
8. test the model using the same method
9. to see how SVD can help us reduce the dimension of the data, SVD is then applied to `training input` and `training output` and obtain results `U in, S in, V in` and `U in, S in, V in`
10. visualize `U, S, V`
11. `U, S, V` are then analyzed in the next section.
12. construct feedforward NN and train it as the steps before
13. do above things for reaction-diffusion system and Lorenz system as well
14. for Lorenz system, use different ρ to train the model and compare the results

4 Computational Results

Three models are built for KS systems. The first one is using feedforward NN. This model takes a long time to train (about a whole day). Figure 1 shows how the performance is improving through training (the loss is reducing), and figure 2 shows how the error distributes. These two graphs indicate our model would probably not perform very well since the error is still high. Figure 3 and 14 shows the real trajectories of testing data and the result with NN.

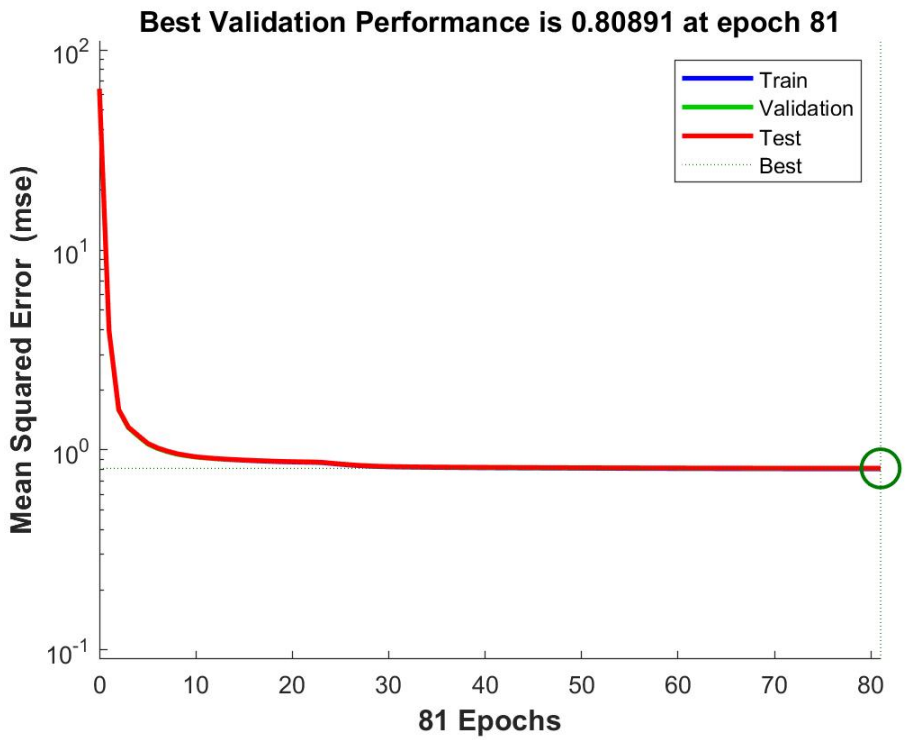

Figure 1: Performance of Feedforward NN for KS

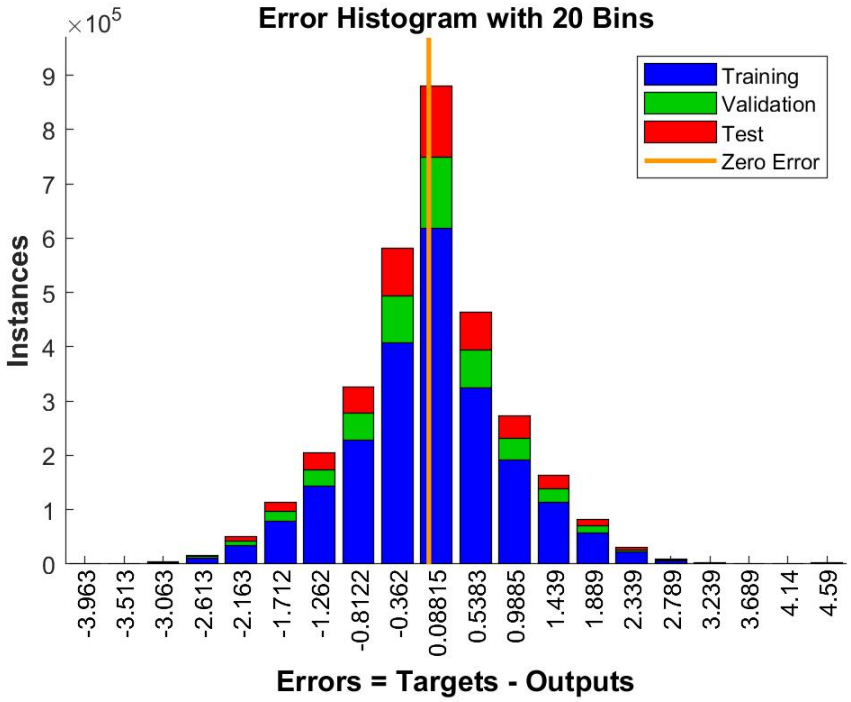

Figure 2: Error histogram of Feedforward NN for KS

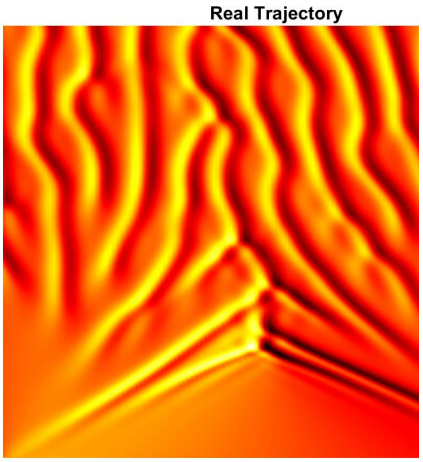

Figure 3: real trajectory

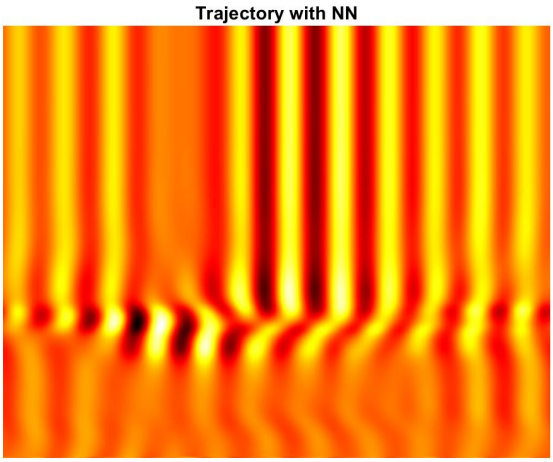

Figure 4: the result with NN

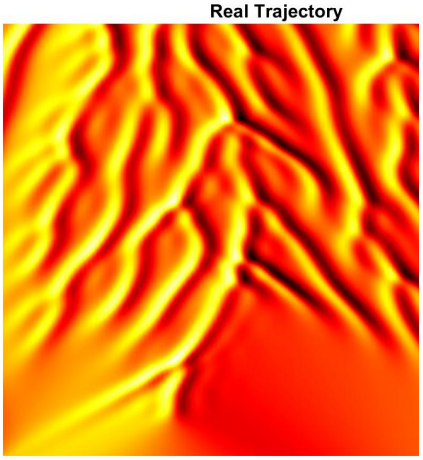

Figure 5: real trajectory

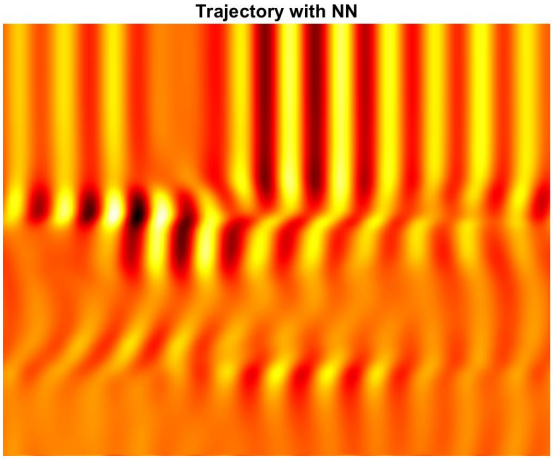

Figure 6: the result with NN

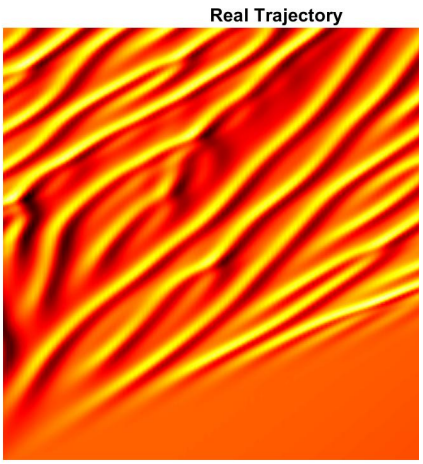

Figure 7: real trajectory

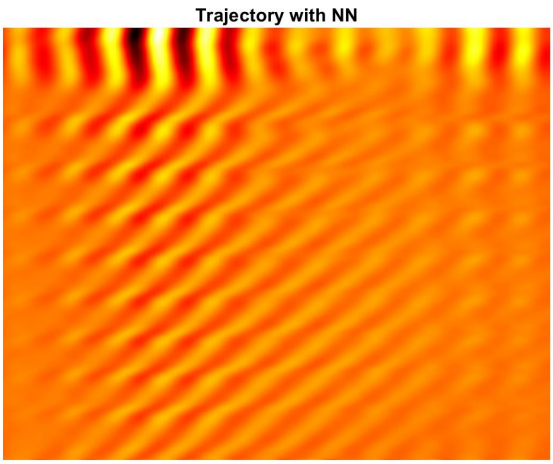

Figure 8: the result with NN

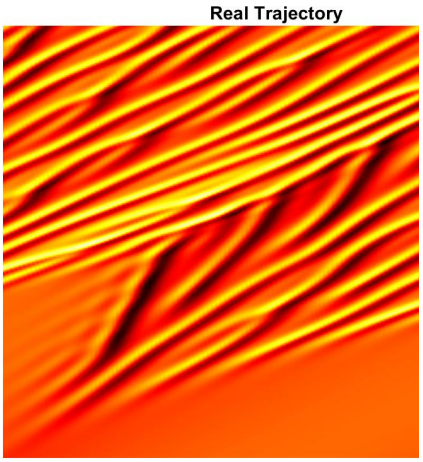

Figure 9: real trajectory

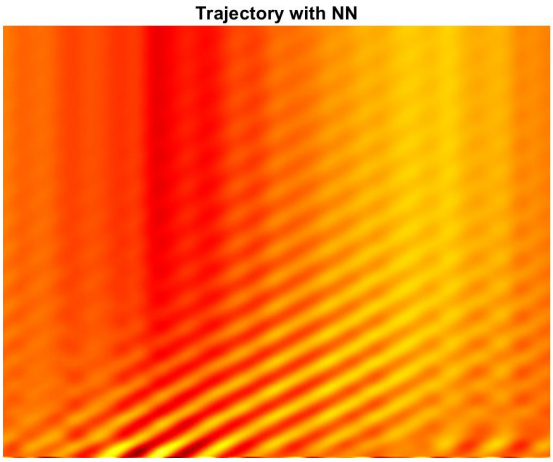

Figure 10: the result with NN

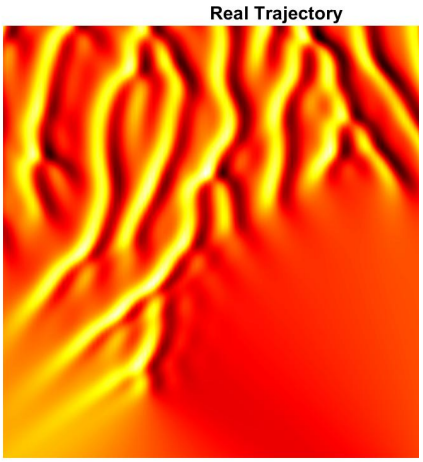

Figure 11: real trajectory

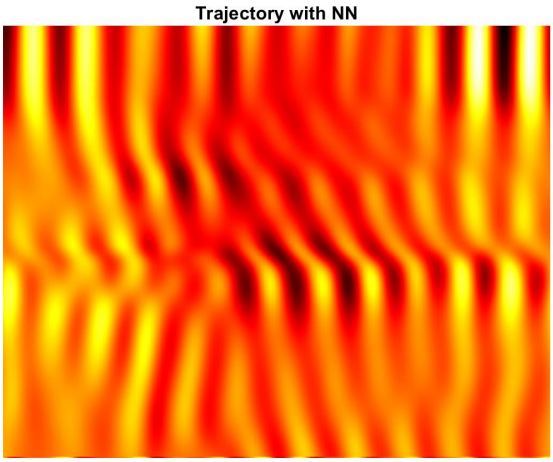

Figure 12: the result with NN

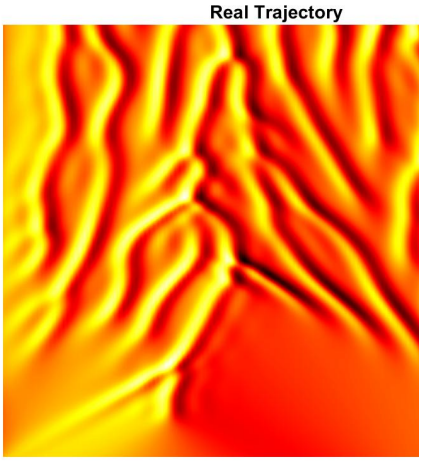

Figure 13: real trajectory

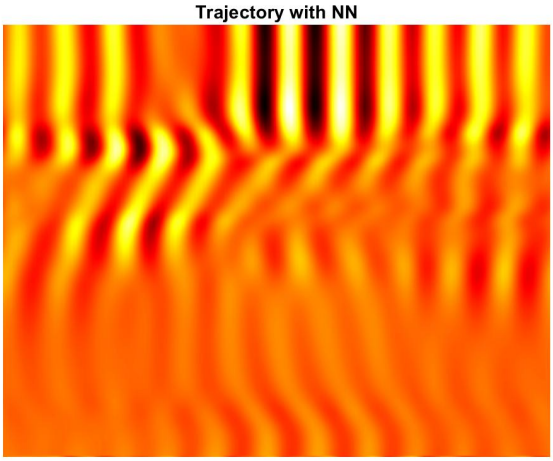

Figure 14: the result with NN

We can see that the prediction is very accurate, but in figure 8-10, it can tell the direction of how the wave propagate. This poor result could be the result of lack of training time, meaning we need more time for training this model because of its complex nature, or the NN structure is too simple and we need a deeper or more complex model.

The second model is built with LSTM. Figure 15 shows how the performance is improving through training (the loss is reducing), and figure 16 shows how the error distributes. The result is similar to the ones from the first model, which is not very good.

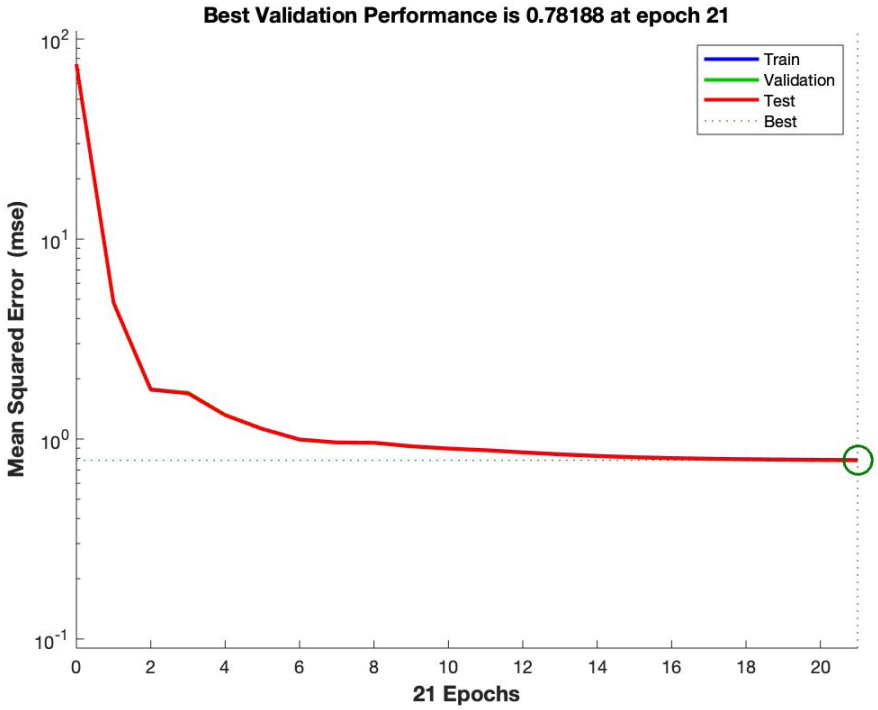

Figure 15: Performance of RNN for KS

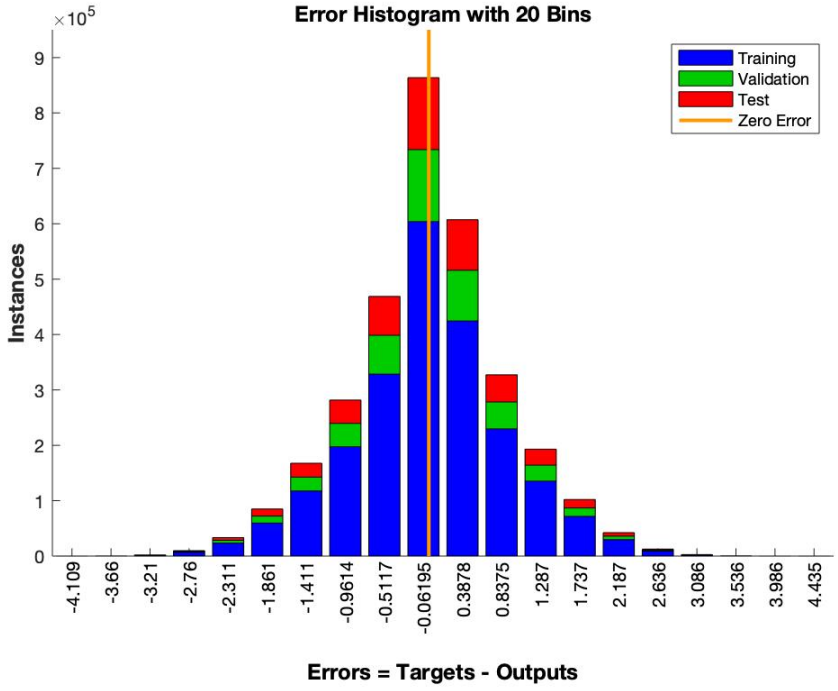

Figure 16: Error histogram of RNN for KS

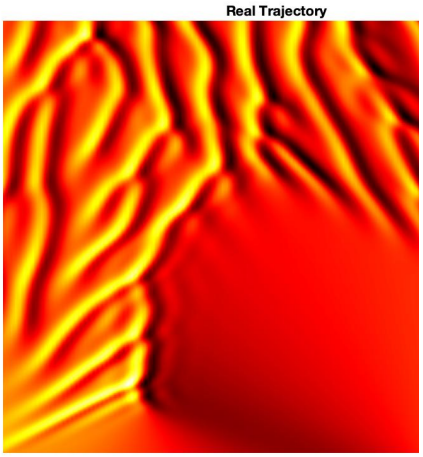

Figure 17: real trajectory

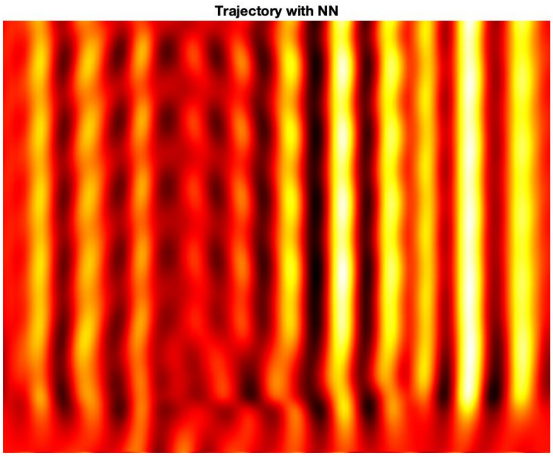

Figure 18: the result with NN

In the third model, we try to use SVD to reduce the dimension of the input data first before putting into the training model. After SVD, the energy of the input and output data are visualized in figure 19, which tells us the rank of the data is around 60. Figure 20 shows how the performance is improving through training (the loss is reducing), and figure 21 shows how the error distributes. These two figures seem to be very promising, but the prediction ability is very limited as well, and it seems to predict the average of the system with certain initial state, which is not very useful.

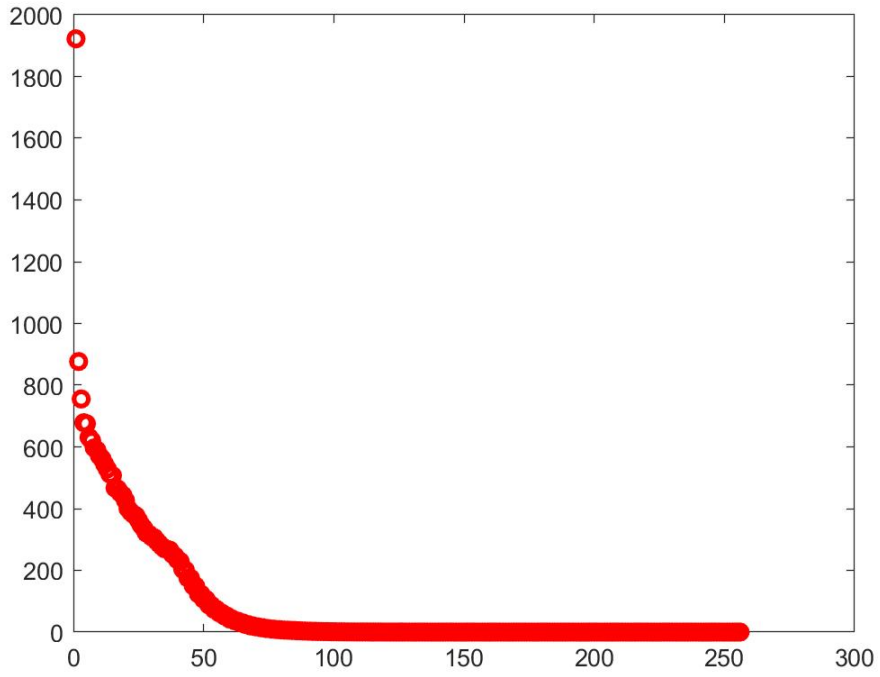

Figure 19: Energy of input and output data

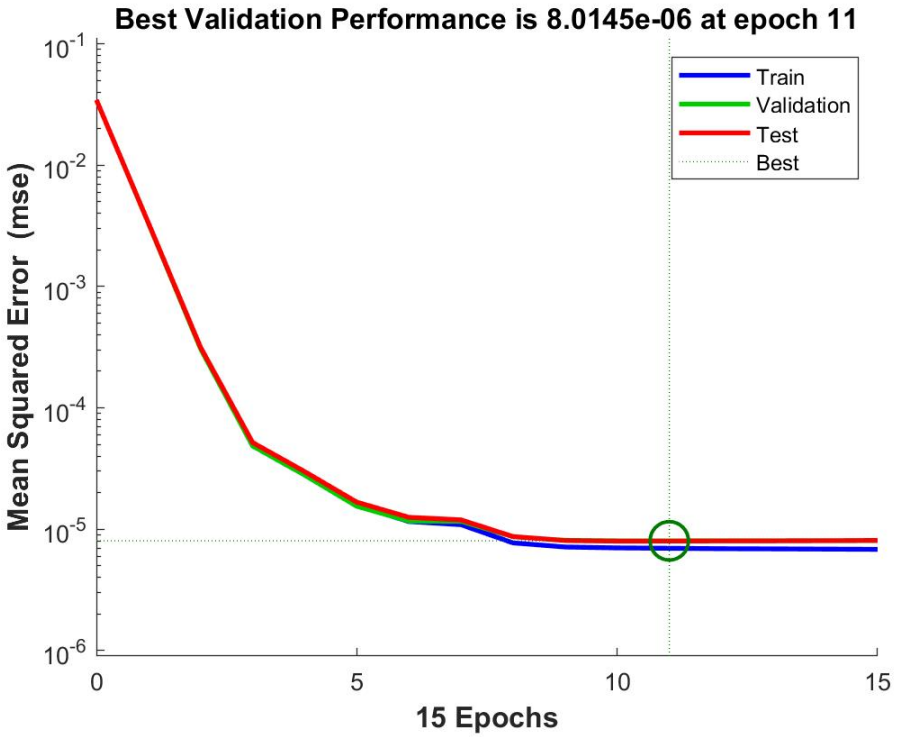

Figure 20: Performance of Feedforward NN for KS with SVD

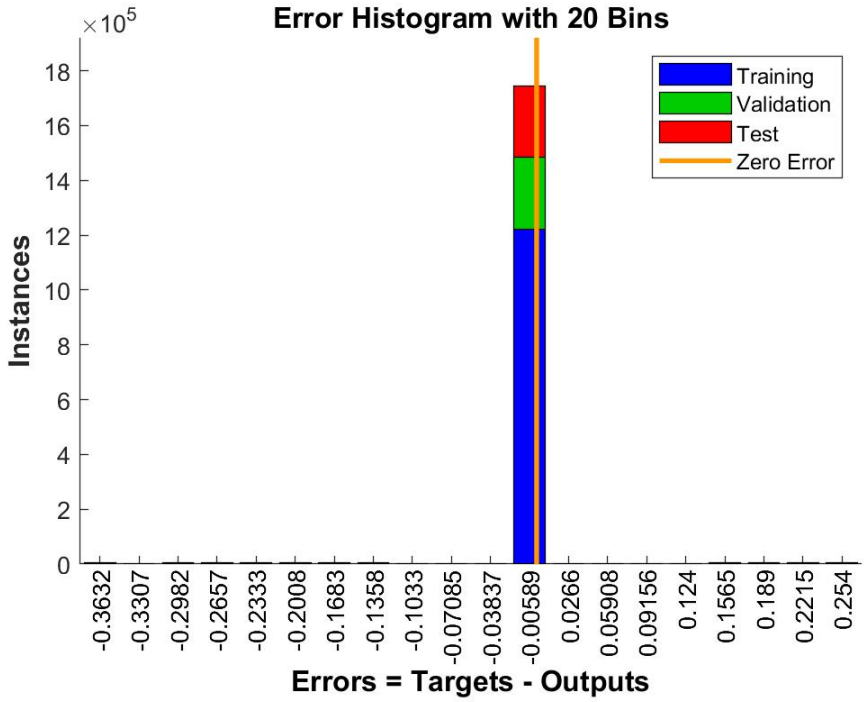

Figure 21: Error histogram of Feedforward NN for KS with SVD

To show how the parameter ρ affect Lorenz system and prediction ability, multiple values of ρ are used to train NN model and test their ability. First, $\rho = 10$ is used to create data to train a NN model and $\rho = 10, 17, 35$ are used to create testing data to test the model. Figure 22 shows some trajectories of the Lorenz system with $\rho = 10$, and figure 23 and 24 show how the model changes along training. Those are very promising graphs and figure 25 and 26 actually show that the model can very well predict how Lorenz system evolves (training and testing data use the same $\rho = 10$).

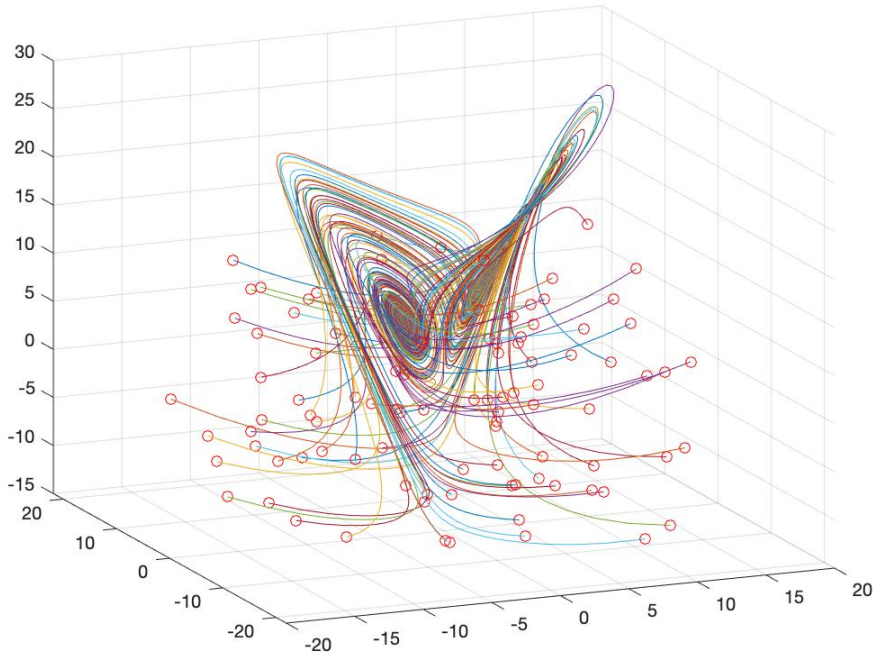

Figure 22: Trajectories of Lorenz system with $\rho = 10$

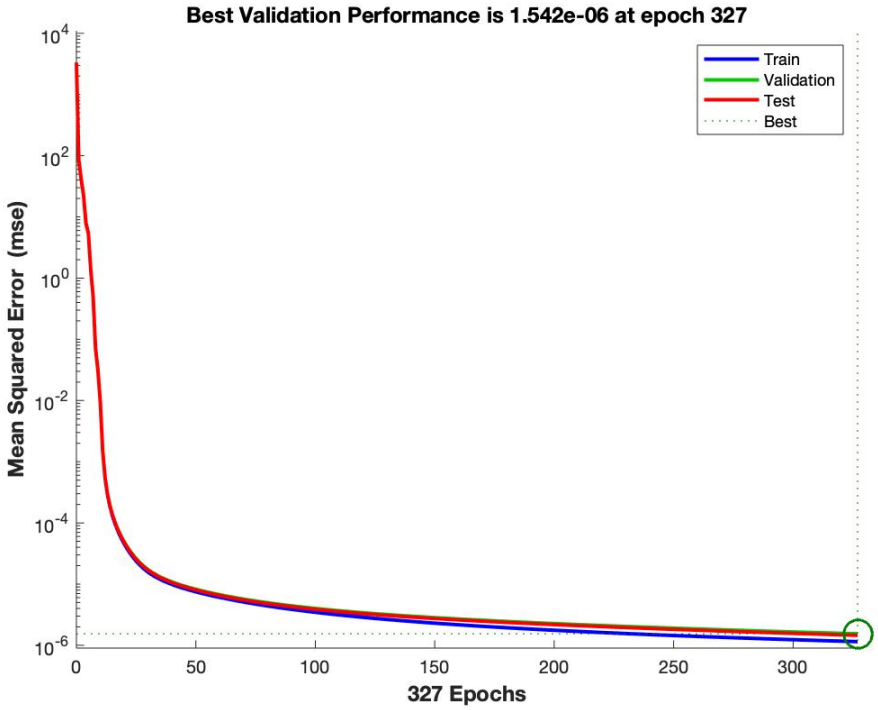

Figure 23: Performance of Feedforward NN for Lorenz system with $\rho = 10$D

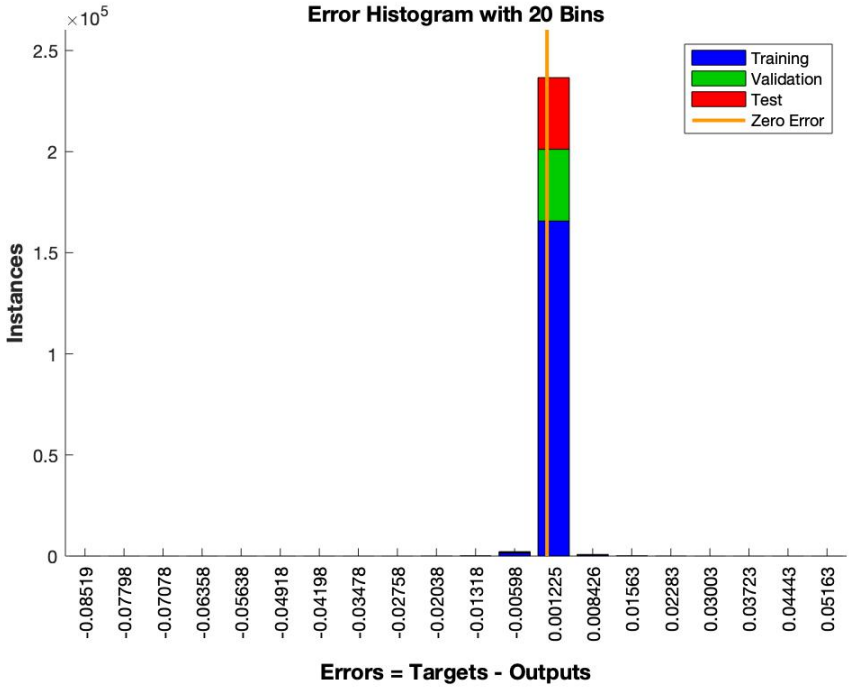

Figure 24: Error histogram of Feedforward NN for Lorenz system with $\rho = 10$

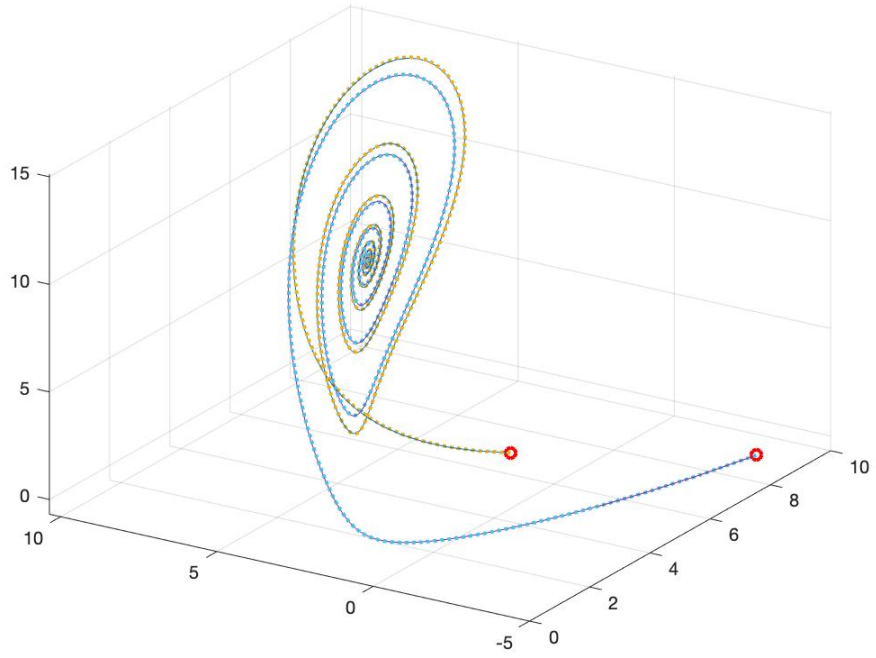

Figure 25: Comparison of real ($\rho = 10$) and predicted ($\rho = 10$) trajectories

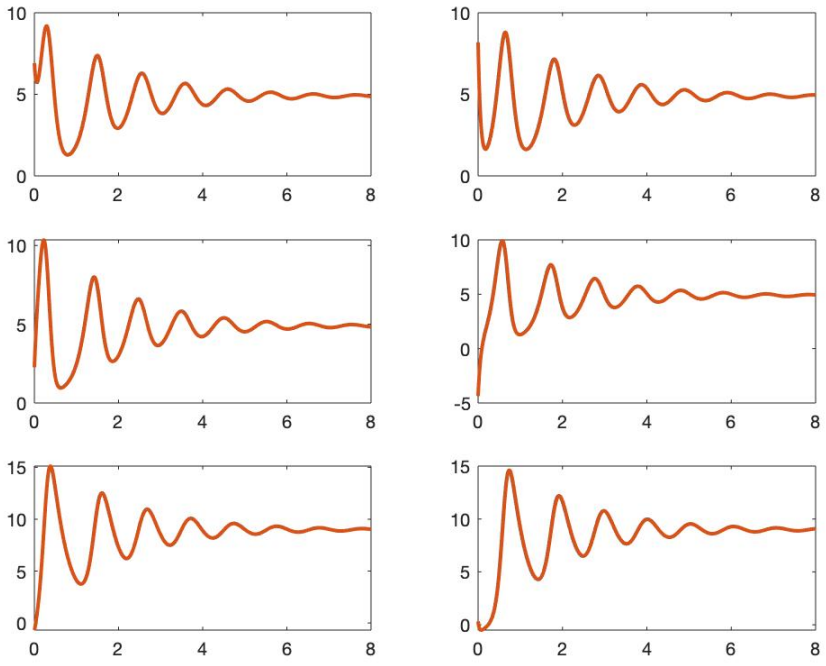

Figure 26: Comparison of real ($\rho = 10$) and predicted ($\rho = 10$) trajectories

Now, $\rho = 17$ is used to create testing data to test the model. Figure 27- 30 show that the model can predict the basic shape of how the system with $\rho = 17$ evolves, but since ρ is different, the model cannot completely predict the exact path like before.

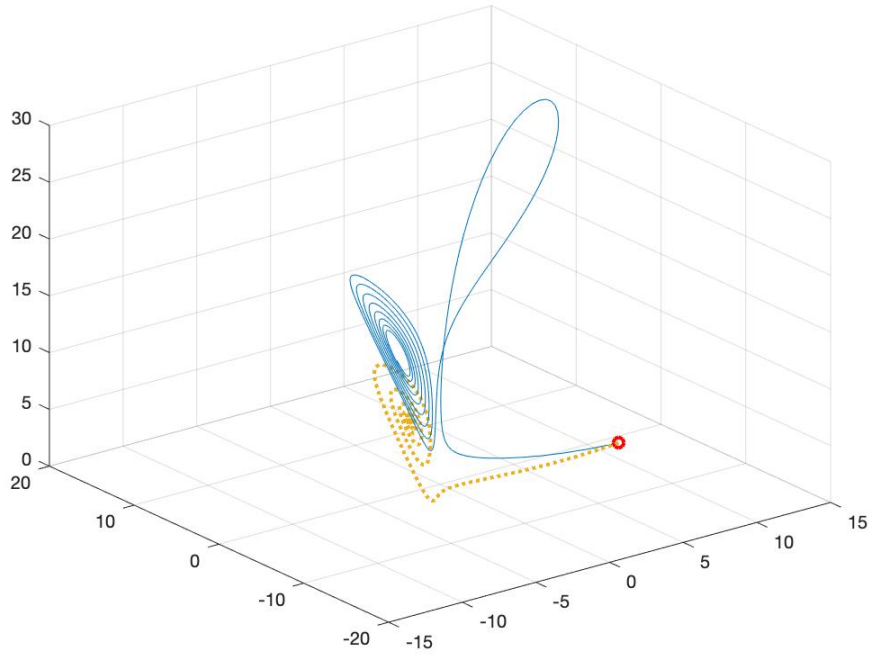

Figure 27: Comparison of real ($\rho = 10$) and predicted ($\rho = 17$) trajectories

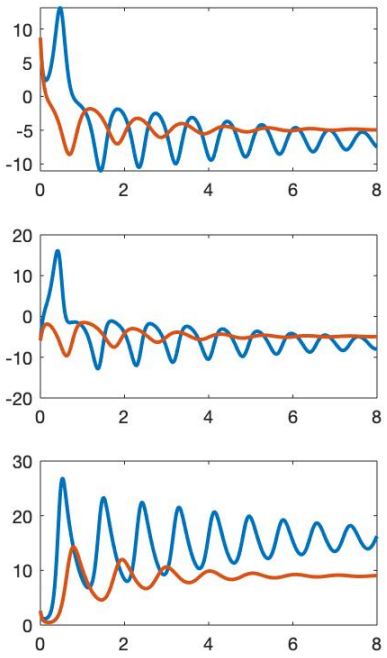

Figure 28: Comparison of real ($\rho = 10$) and predicted ($\rho = 17$) trajectories

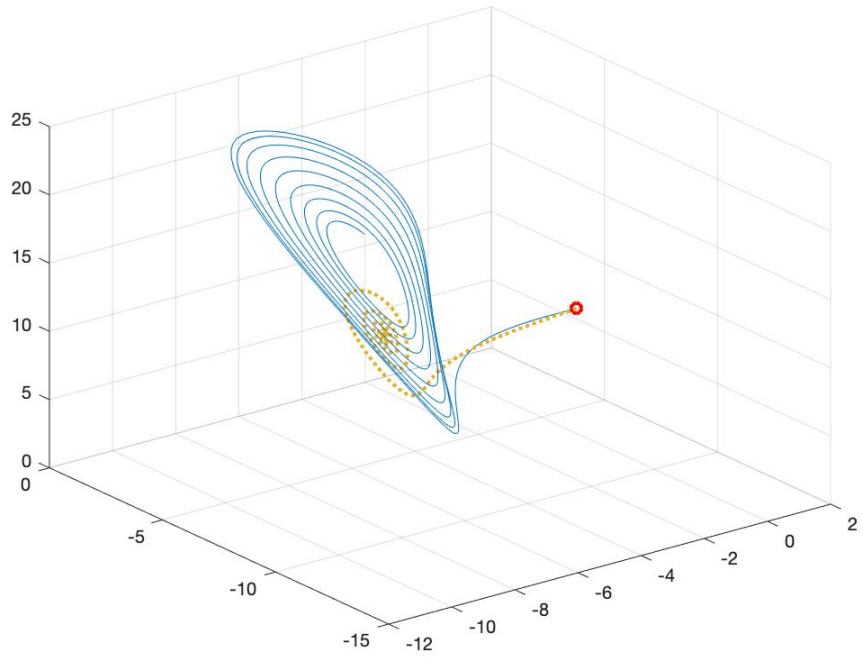

Figure 29: Comparison of real ($\rho = 10$) and predicted ($\rho = 17$) trajectories

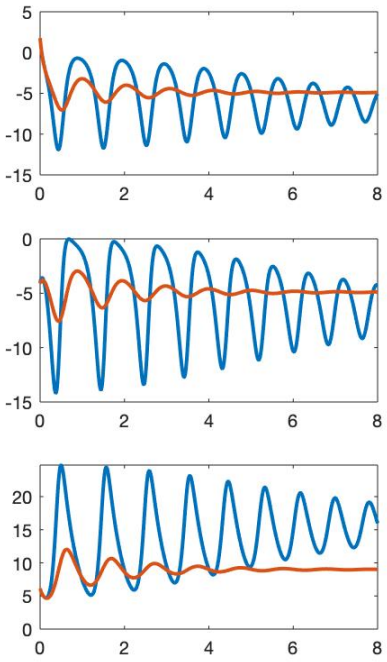

Figure 30: Comparison of real ($\rho = 10$) and predicted ($\rho = 17$) trajectories

Then, $\rho = 35$ is used to create testing data to test the model. Figure 31- 34 show that the model can also predict the basic shape of how the system with $\rho = 35$ evolves, but the model cannot completely predict the exact path like before either. However, compare the result with $\rho = 17, 35$, we can see with $\rho = 35$, the predicted result is more off, which is reasonable since the parameter is further away from $\rho = 10$.

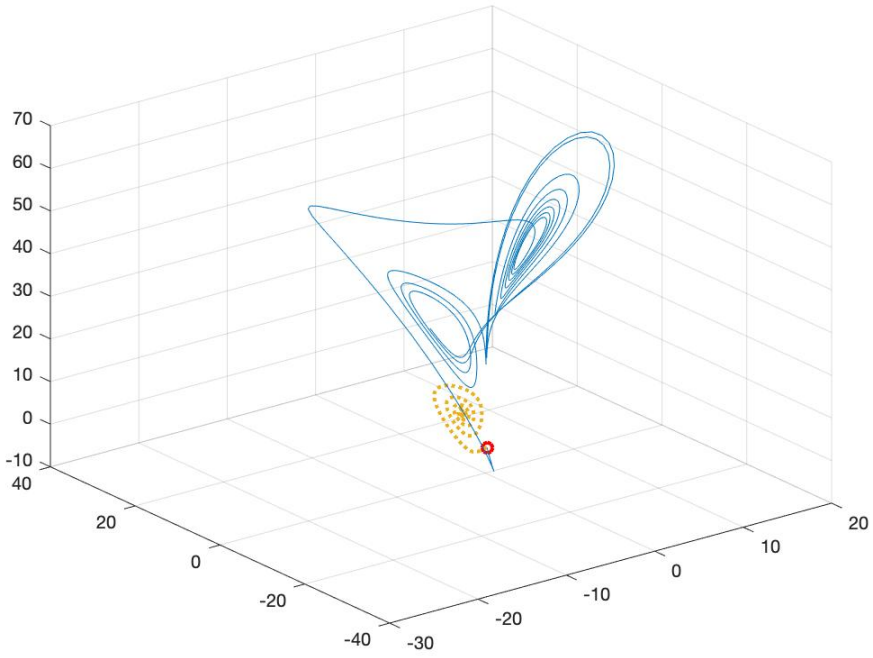

Figure 31: Comparison of real ($\rho = 10$) and predicted ($\rho = 35$) trajectories

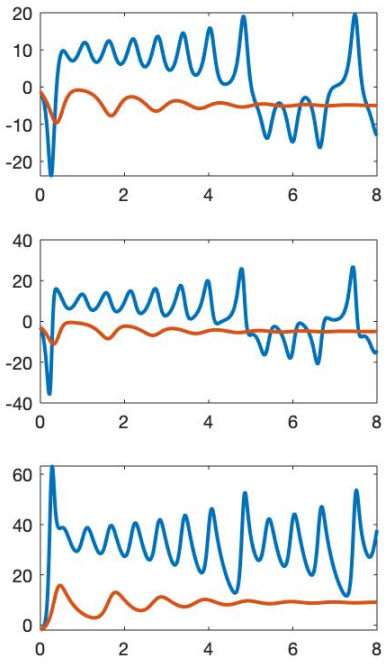

Figure 32: Comparison of real ($\rho = 10$) and predicted ($\rho = 35$) trajectories

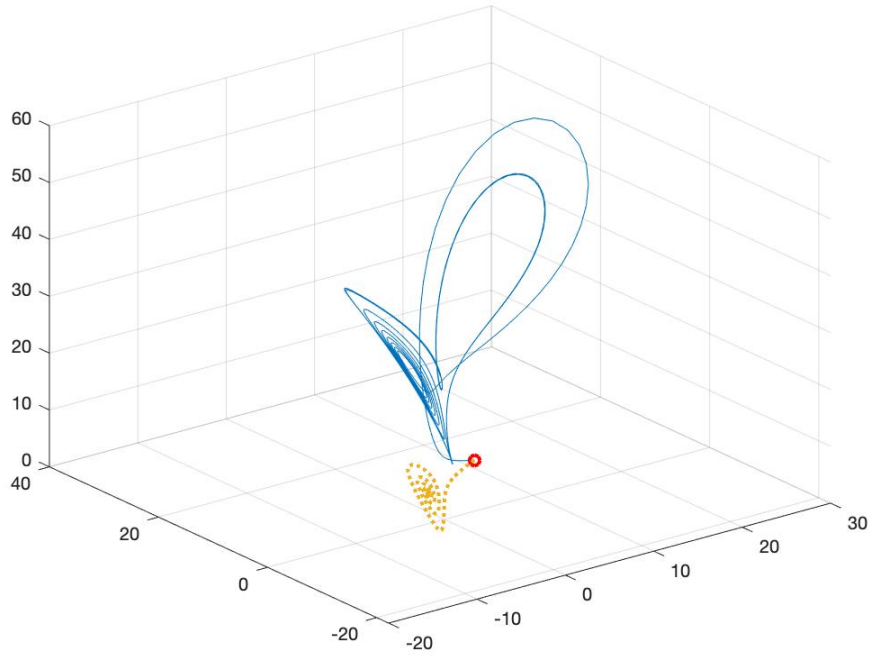

Figure 33: Comparison of real ($\rho = 10$) and predicted ($\rho = 35$) trajectories

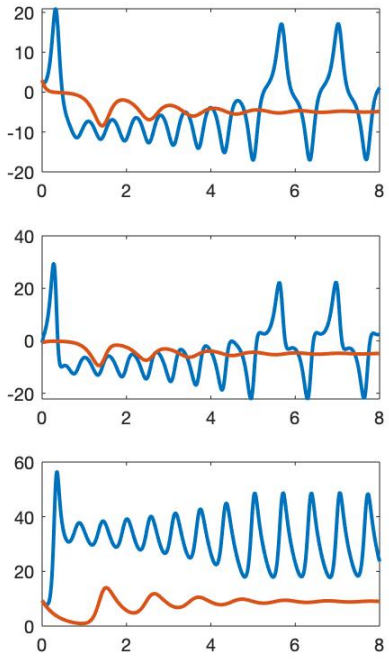

Figure 34: Comparison of real ($\rho = 10$) and predicted ($\rho = 35$) trajectories

We now use $\rho = 28$ to create data to train a NN model and $\rho = 28, 17, 35$ are used to create testing data to test the model. Figure 35 shows some trajectories of the Lorenz system with $\rho = 28$, and figure 36 and 37 show how the model changes along training. Those are very promising graphs and figure 38 and 39 actually show that the model can very well predict how Lorenz system evolves (training and testing data use the same $\rho = 28$).

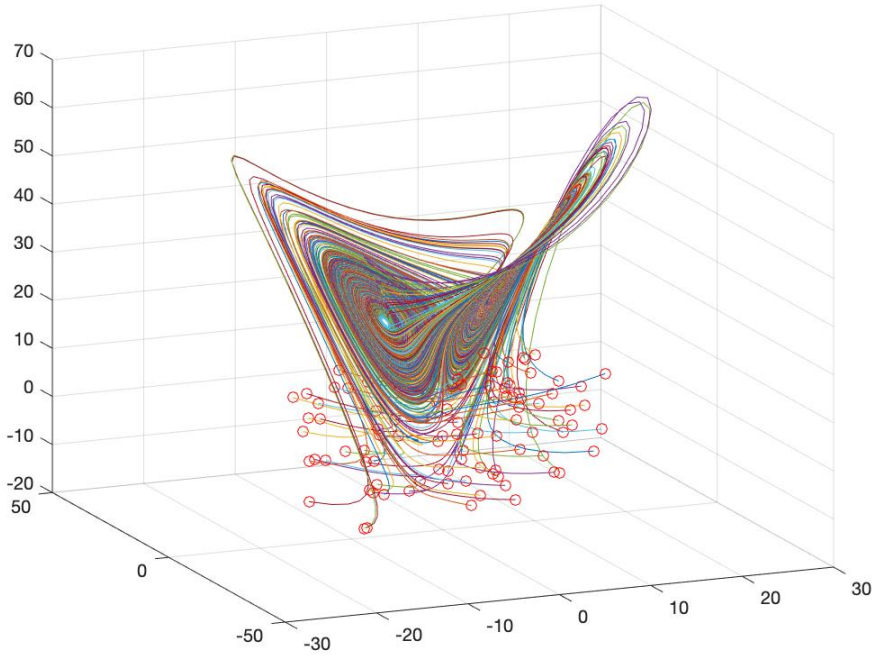

Figure 35: Trajectories of Lorenz system with $\rho = 28$

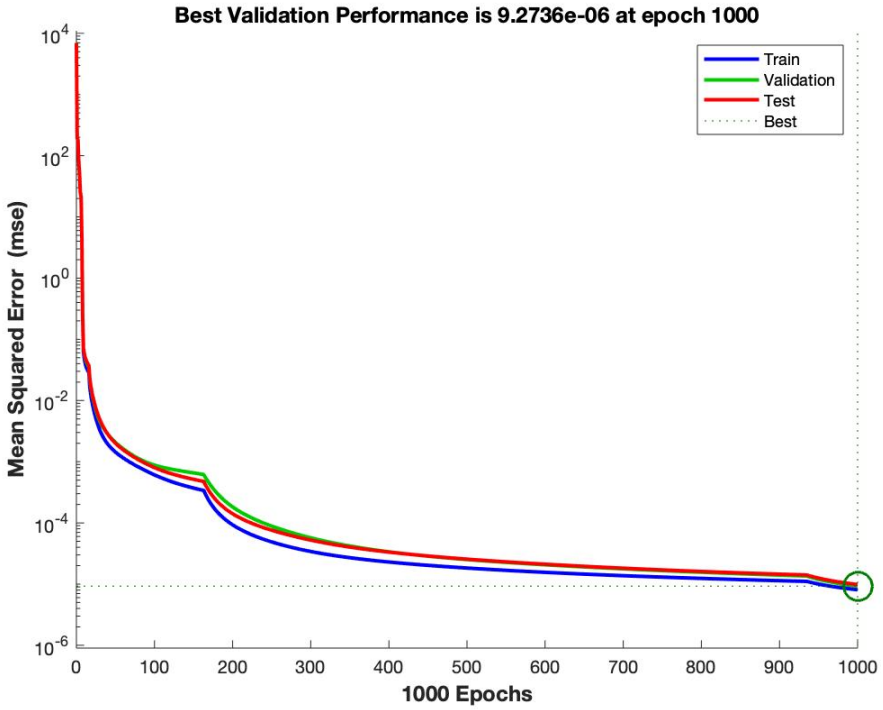

Figure 36: Performance of Feedforward NN for Lorenz system with $\rho = 28$

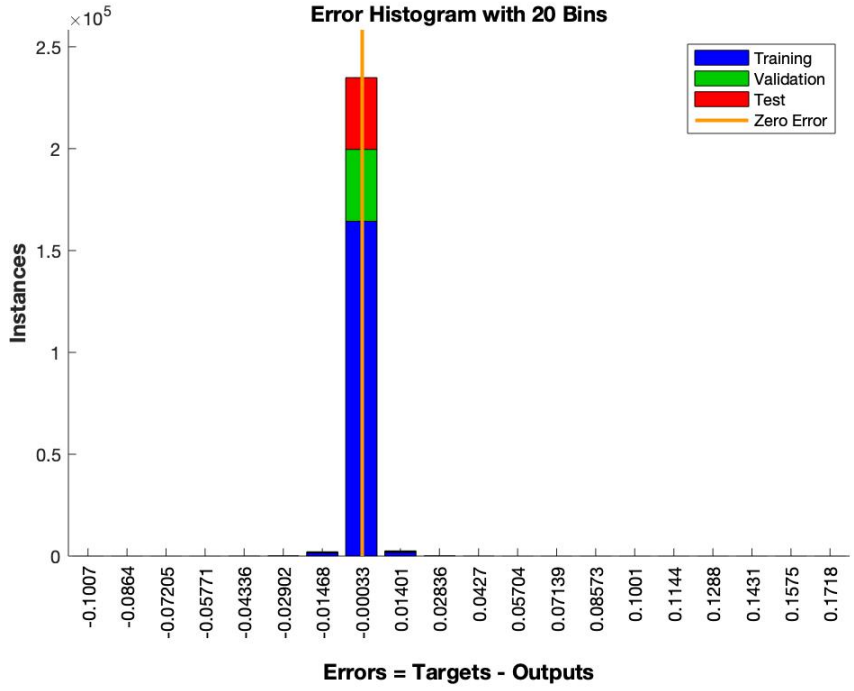

Figure 37: Error histogram of Feedforward NN for Lorenz system with $\rho = 28$

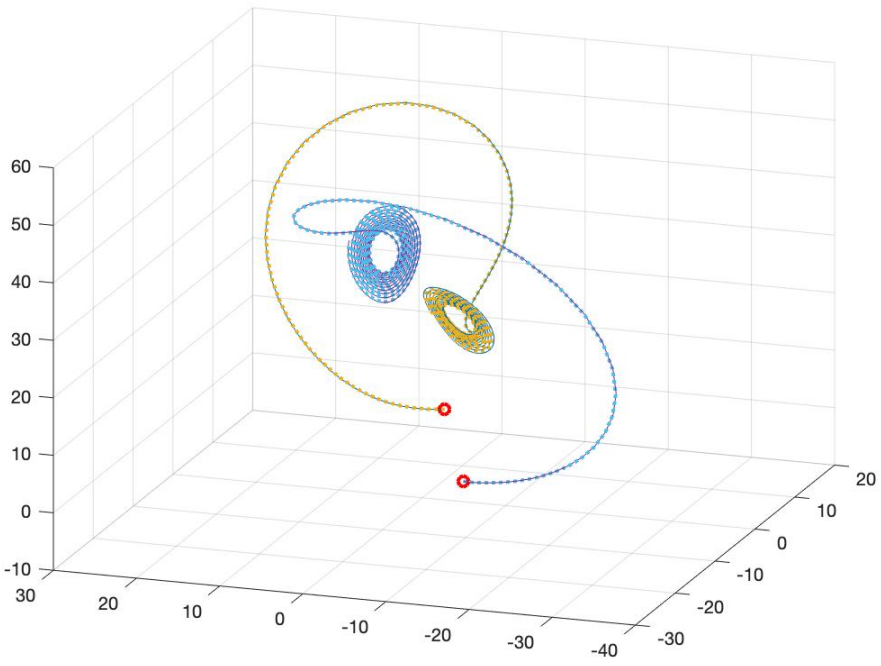

Figure 38: Comparison of real ($\rho = 28$) and predicted ($\rho = 28$) trajectories

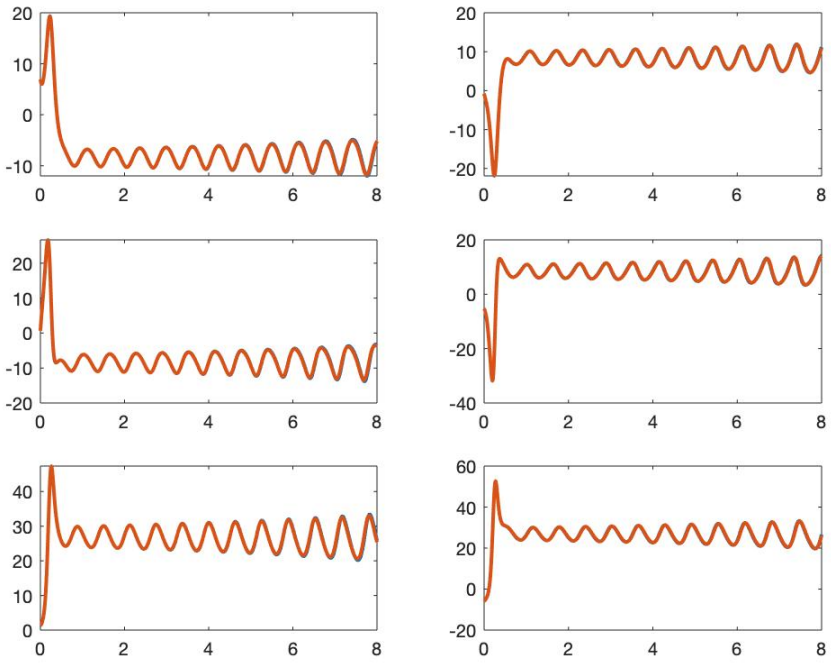

Figure 39: Comparison of real ($\rho = 28$) and predicted ($\rho = 28$) trajectories

Now, $\rho = 17$ is used to create testing data to test the model. Figure 40- 41 show that the model can predict the basic shape of how the system with $\rho = 17$ evolves, but since ρ is different, the model cannot completely predict the exact path like before.

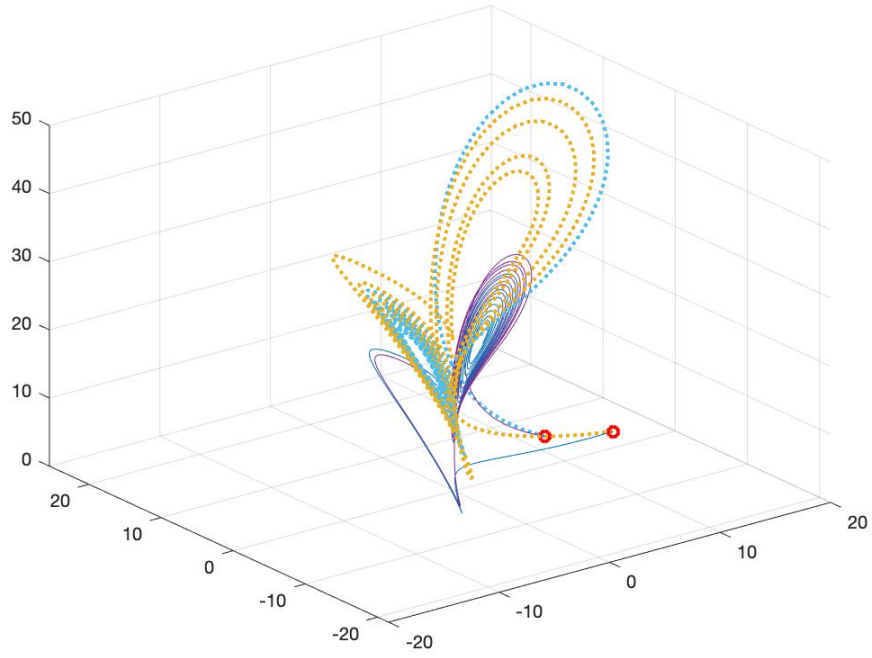

Figure 40: Comparison of real ($\rho = 28$) and predicted ($\rho = 17$) trajectories

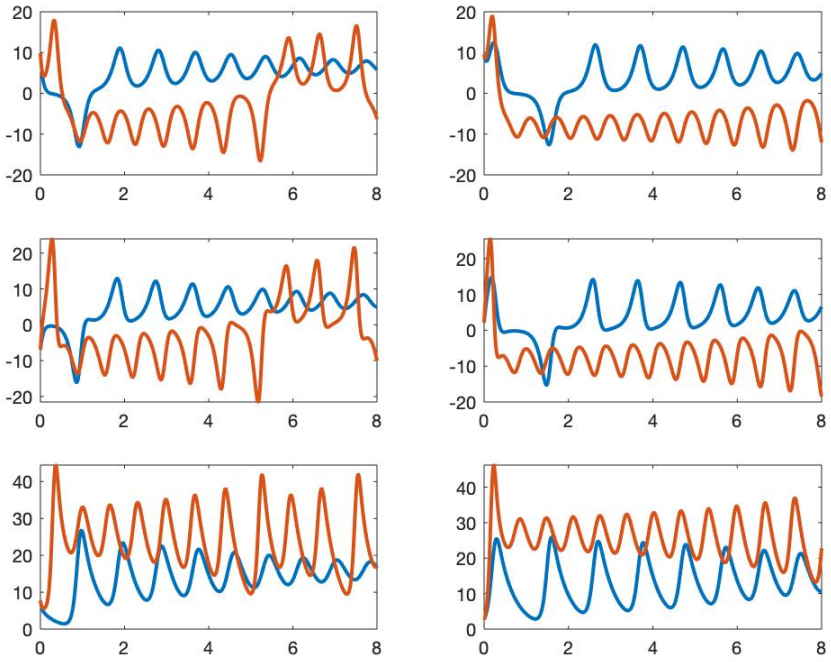

Figure 41: Comparison of real ($\rho = 28$) and predicted ($\rho = 17$) trajectories

Then, $\rho = 35$ is used to create testing data to test the model. Figure 42- 43 show that the model can also predict the basic shape of how the system with $\rho = 35$ evolves, but the model cannot completely predict the exact path like before either. However, compare the result with $\rho = 17, 35$, we can see with $\rho = 17$, the predicted result is more off, which is reasonable since the parameter is further away from $\rho = 28$.

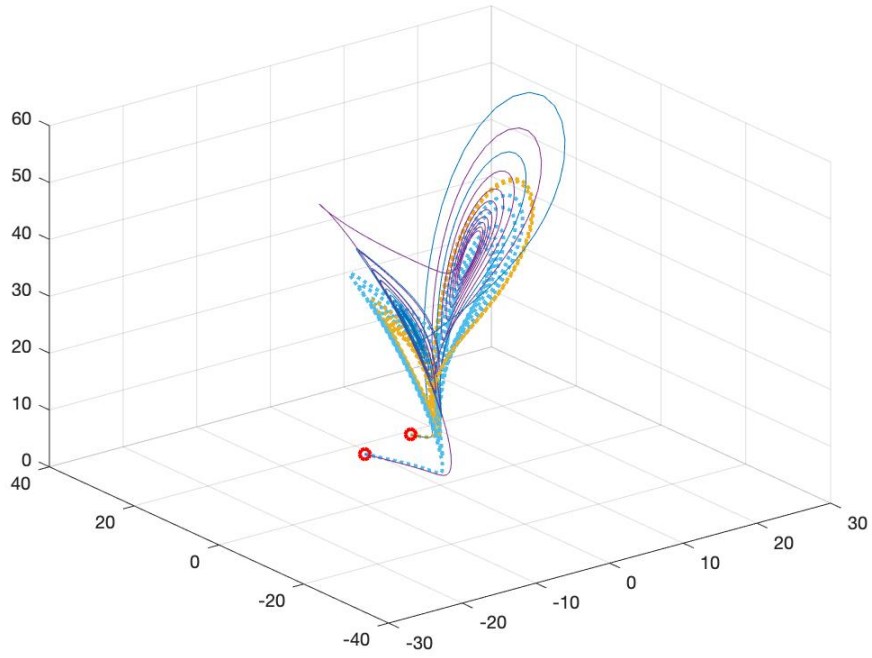

Figure 42: Comparison of real ($\rho = 28$) and predicted ($\rho = 35$) trajectories

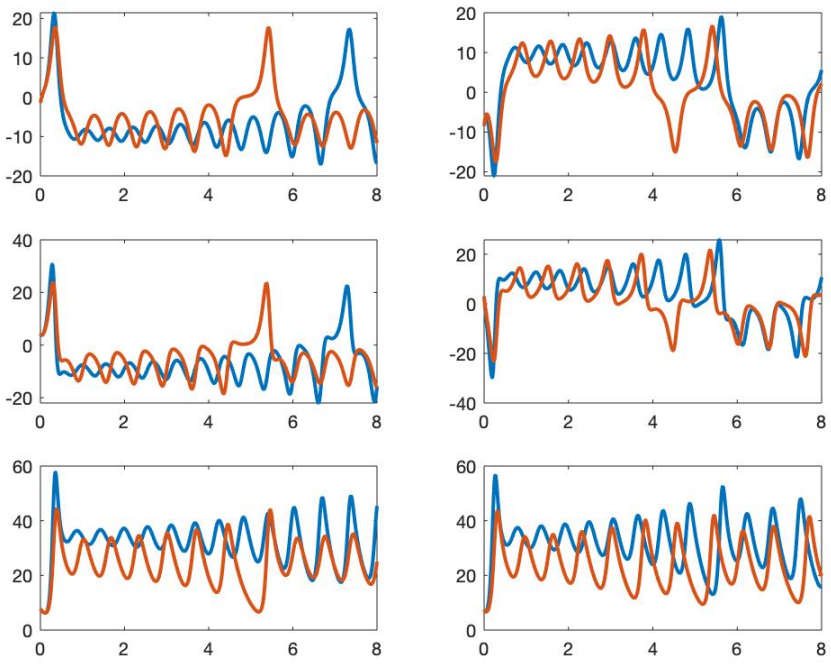

Figure 43: Comparison of real ($\rho = 28$) and predicted ($\rho = 35$) trajectories

Now, $\rho = 40$ is used to create data to train a NN model and $\rho = 40, 17, 35$ are used to create testing data to test the model. Figure 44 shows some trajectories of the Lorenz system with $\rho = 40$, and figure 45 and 46 show how the model changes along training. Those are very promising graphs and figure 47 and 48 actually show that the model can predict how Lorenz system evolves before certain time (training and testing data use the same $\rho = 40$), but after a while, the paths fall off, which indicates that the larger ρ is, the earlier or easier the model is gonna be different from the true path.

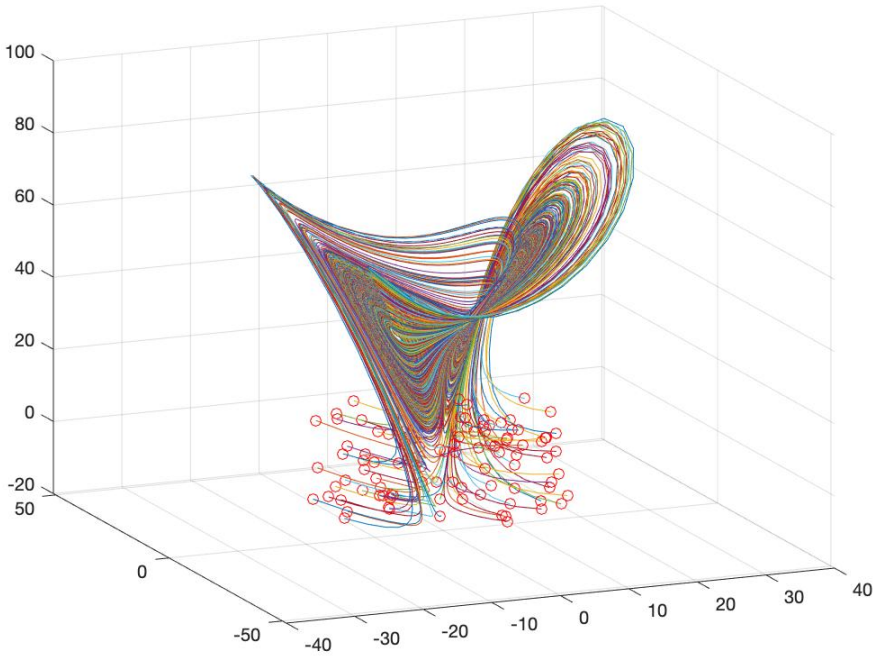

Figure 44: Trajectories of Lorenz system with $\rho = 40$

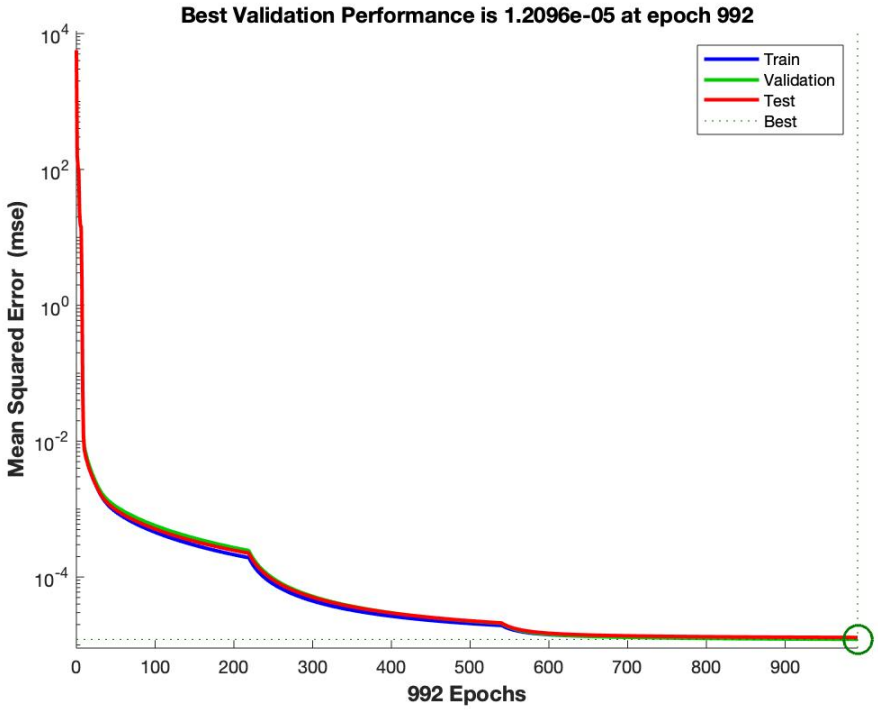

Figure 45: Error histogram of Feedforward NN for Lorenz system with $\rho = 40$

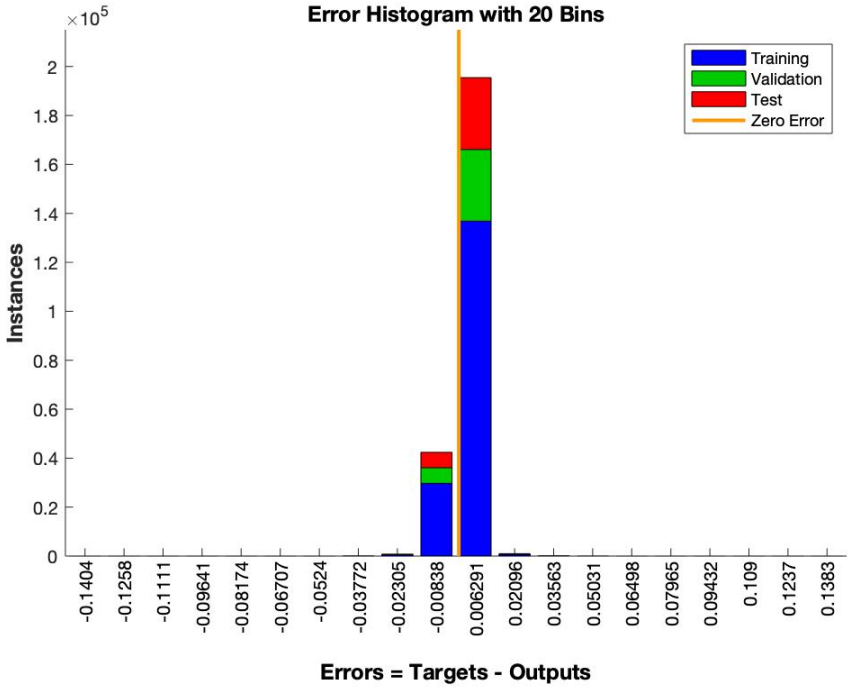

Figure 46: Error histogram of Feedforward NN for Lorenz system with $\rho = 40$

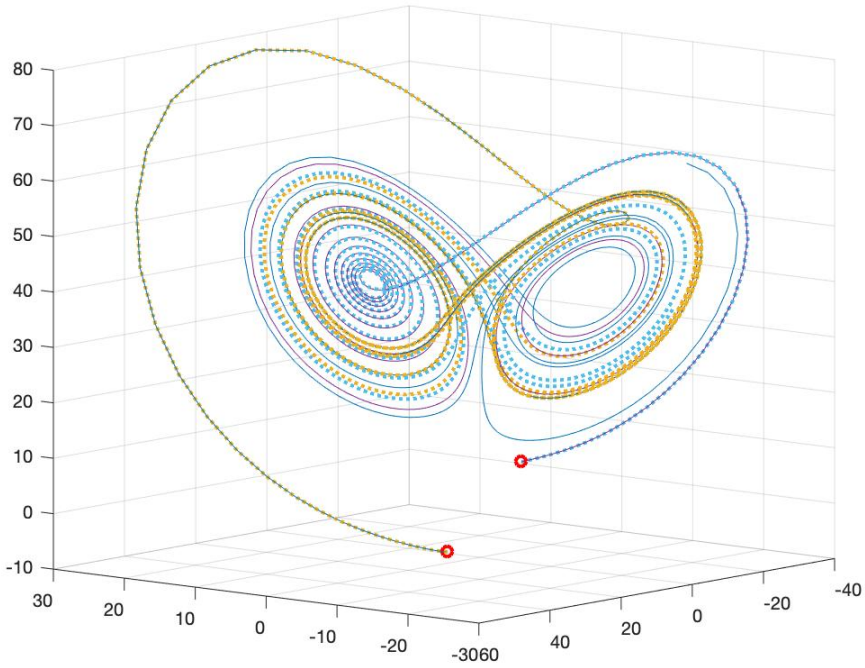

Figure 47: Comparison of real ($\rho = 40$) and predicted ($\rho = 40$) trajectories

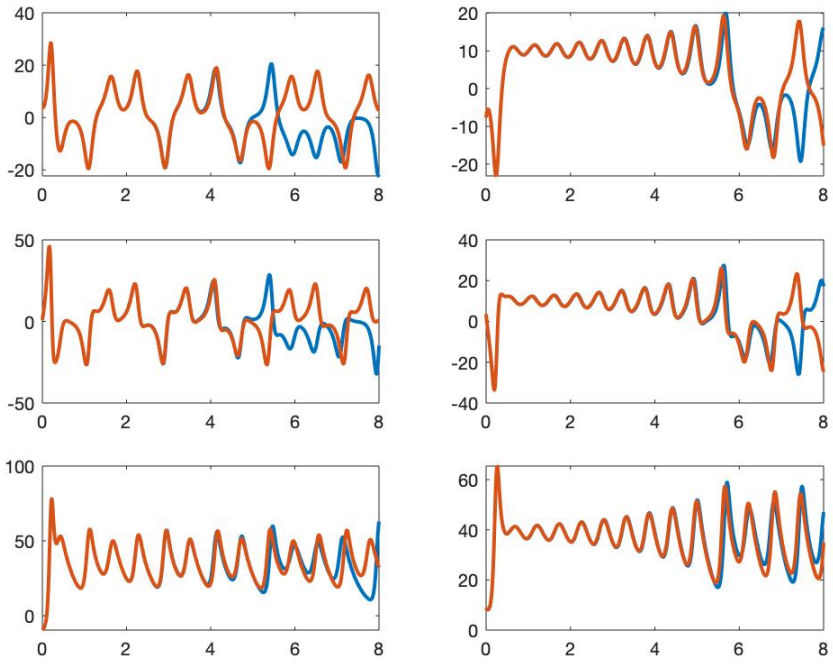

Figure 48: Comparison of real ($\rho = 40$) and predicted ($\rho = 40$) trajectories

Now, $\rho = 17$ is used to create testing data to test the model. Figure 49- 50 show that the model can predict the basic shape of how the system with $\rho = 17$ evolves, but since ρ is different, the model cannot completely predict the exact path like before.

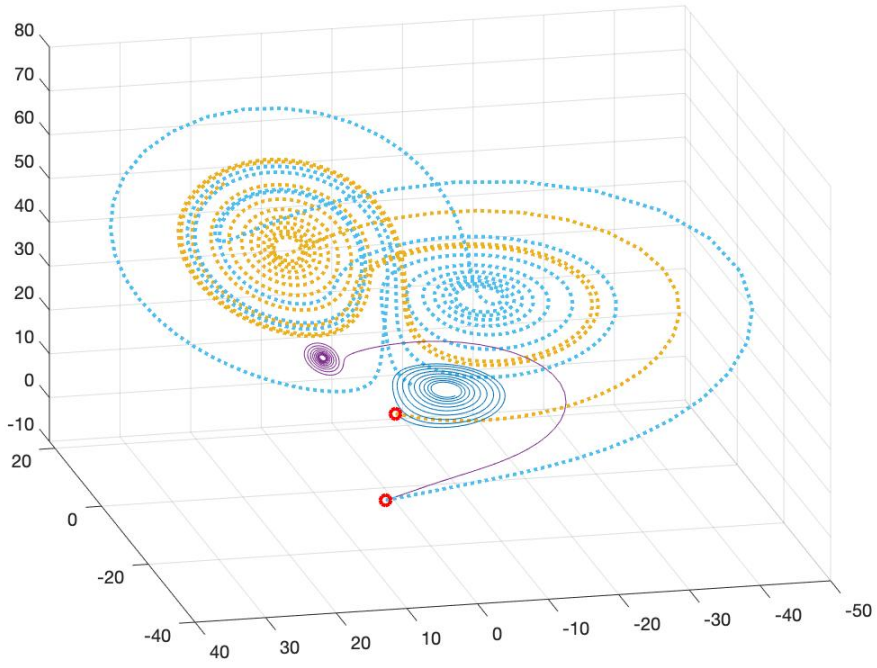

Figure 49: Comparison of real ($\rho = 40$) and predicted ($\rho = 17$) trajectories

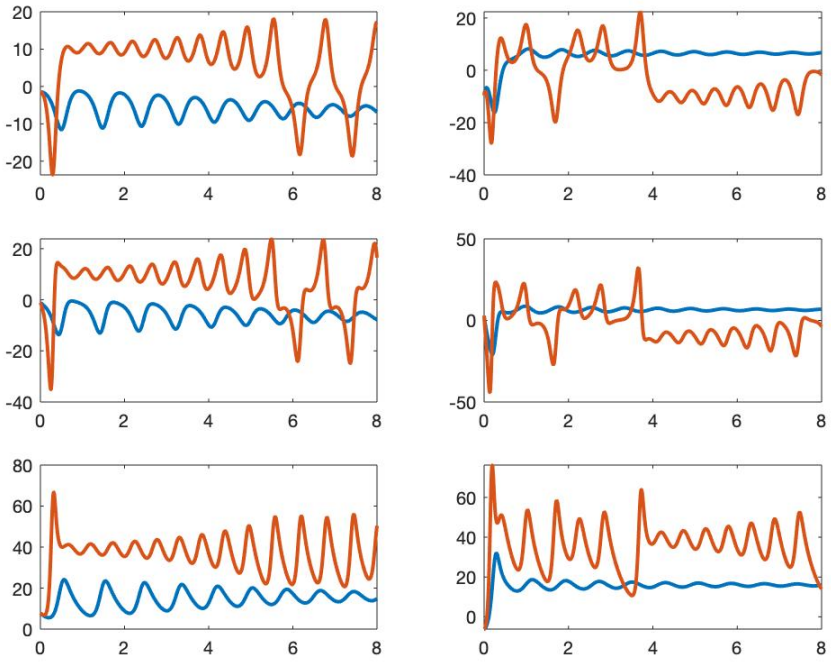

Figure 50: Comparison of real ($\rho = 40$) and predicted ($\rho = 17$) trajectories

Then, $\rho = 35$ is used to create testing data to test the model. Figure 51- 52 show that the model can also predict the basic shape of how the system with $\rho = 35$ evolves, but the model cannot completely predict the exact path like before either. However, compare the result with $\rho = 17, 35$, we can see with $\rho = 17$, the predicted result is more off, which is reasonable since the parameter is further away from $\rho = 40$.

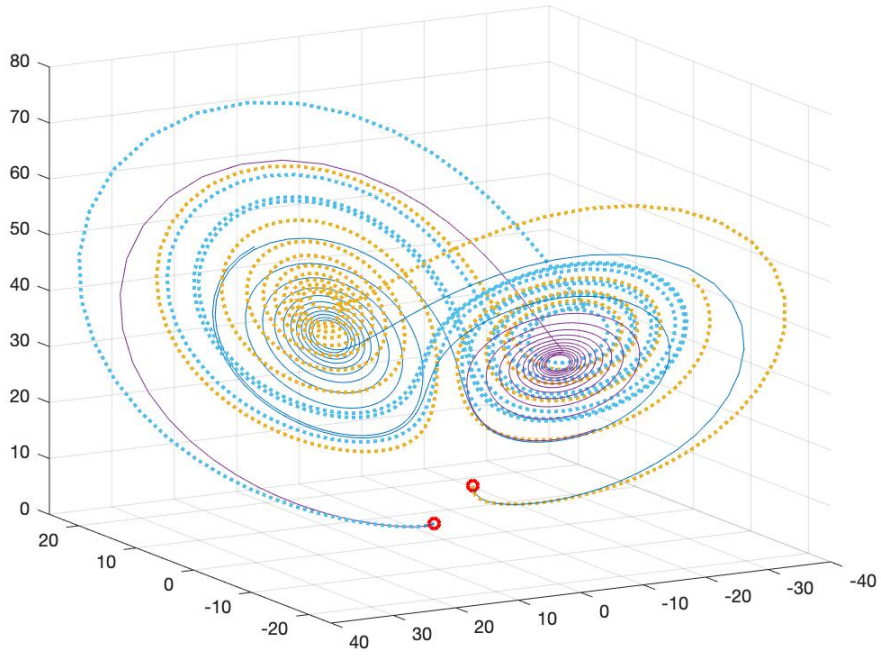

Figure 51: Comparison of real ($\rho = 40$) and predicted ($\rho = 35$) trajectories

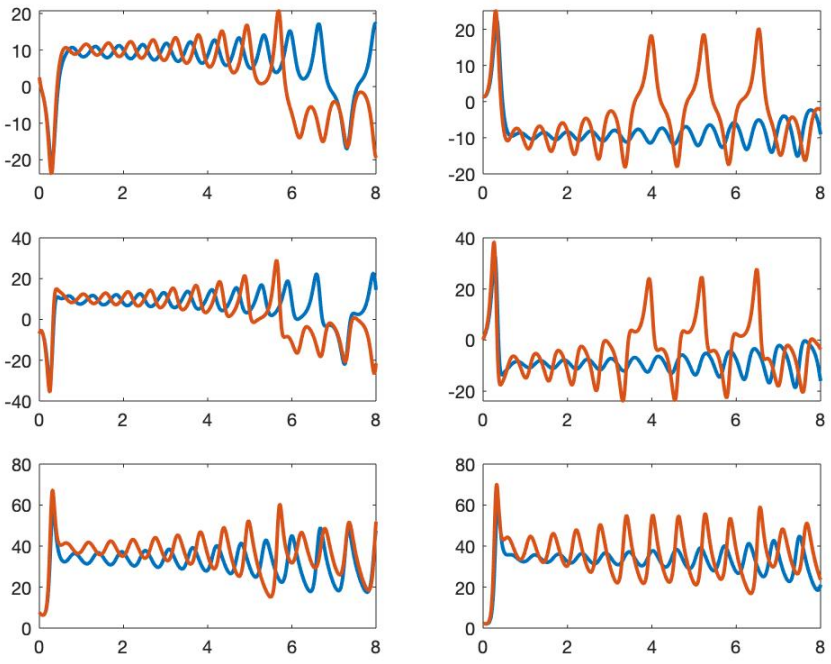

Figure 52: Comparison of real ($\rho = 40$) and predicted ($\rho = 35$) trajectories

5 Summary and Conclusions

Through this project, we can see again how we can use NN to help us make prediction for dynamic system, even for chaotic systems. However, for higher-dimension system, extra future work would be needed to make more accurate predictions.

References

[1] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

Appendix A MATLAB Code

```
close all; clear all; clc

% Simulate KS system.
% Kuramoto-Sivashinsky equation (from Trefethen)
% u_t = -u*u_x - u_xx - u_xxxx,  periodic BCs

rng(42);
N = 1024/8; training_input = []; training_output = [];
x = 32*pi*(1:N)'/N;
for i = 1:100
    u = cos(rand(1,1)*x/16).*(1+sin(rand(1,1)*x/16)); % Change to random result.
    v = fft(u);

    % % % % % %
    %Spatial grid and initial condition:
    h = 0.025;
    k = [0:N/2-1 0 -N/2+1:-1]'/16;
    L = k.^2 - k.^4;
    E = exp(h*L); E2 = exp(h*L/2);
    M = 16;
    r = exp(1i*pi*((1:M)-.5)/M);
    LR = h*L(:,ones(M,1)) + r(ones(N,1),:);
    Q = h*real(mean( (exp(LR/2)-1)./LR ,2));
    f1 = h*real(mean( (-4-LR+exp(LR).*(4-3*LR+LR.^2))./LR.^3 ,2));
    f2 = h*real(mean( (2+LR+exp(LR).*(-2+LR))./LR.^3 ,2));
    f3 = h*real(mean( (-4-3*LR-LR.^2+exp(LR).*(4-LR))./LR.^3 ,2));

    % Main time-stepping loop:
    uu = u; tt = 0;
    tmax = 100; nmax = round(tmax/h); nplt = floor((tmax/250)/h); g = -0.5i*k;

    for n = 1:nmax
        t = n*h;
        Nv = g.*fft(real(ifft(v)).^2);
        a = E2.*v + Q.*Nv;
        Na = g.*fft(real(ifft(a)).^2);
        b = E2.*v + Q.*Na;
        Nb = g.*fft(real(ifft(b)).^2);
        c = E2.*a + Q.*(2*Nb-Nv);
        Nc = g.*fft(real(ifft(c)).^2);
```

```
        v = E.*v + Nv.*f1 + 2*(Na+Nb).*f2 + Nc.*f3;
        if mod(n,nplt)==0
            u = real(ifft(v));
            uu = [uu,u]; tt = [tt,t];
        end
    end

    % Input & Output as training data.
    training_input = [training_input; uu(:,1:end-1)'];
    training_output = [training_output; uu(:,2:end)'];

%     close all;
%     % Plot results:
%     surf(tt,x,uu), shading interp, colormap(hot), axis tight
%     % view([-90 90]), colormap(autumn);
%     set(gca,'zlim',[-5 50])
%     save('kuramoto_sivishinky.mat','x','tt','uu')
%     figure(2), pcolor(x,tt,uu.'),shading interp, colormap(hot), axis off, pause(1.)
end

%%

% Seperate training and testing data.
% q = randperm(N);
% training_u = uu(q(1:1000),:); testing_u = uu(q(1001:end),:);

% % Input & Output as training data.
% training_input = []; training_output = [];
% figure(3)
% for i = 1:1000
%     y = training_u(i,:);
%     training_input = [training_input; y(1:end-1)'];
%     training_output = [training_output; y(2:end)'];
%     plot(tt, y, 'b', 0, y(1), 'ro'), hold on,
% end

%%

net = feedforwardnet([50 50 50]);
net.layers{1}.transferFcn = 'logsig';
net.layers{2}.transferFcn = 'radbas';
net.layers{3}.transferFcn = 'purelin';
net = train(net, training_input.', training_output.');

%% LSTM.

layers = [sequenceInputLayer(N)
    lstmLayer(200, 'OutputMode', 'sequence')
    fullyConnectedLayer(50)
    dropoutLayer(.1)
    fullyConnectedLayer(N)
    regressionLayer];

options = trainingOptions('adam', 'MaxEpochs', 2000, 'MiniBatchSize', 5, ...
```

```
    'Plots', 'training-progress');

net = trainNetwork(training_input.', training_output.', layers, options);

%%

close all;
for i = 1:10

    u = cos(rand(1,1)*x/16).*(1+sin(rand(1,1)*x/16)); % Change to random result.
    v = fft(u);

    % % % % % %
    %Spatial grid and initial condition:
    h = 0.025;
    k = [0:N/2-1 0 -N/2+1:-1]'/16;
    L = k.^2 - k.^4;
    E = exp(h*L); E2 = exp(h*L/2);
    M = 16;
    r = exp(1i*pi*((1:M)-.5)/M);
    LR = h*L(:,ones(M,1)) + r(ones(N,1),:);
    Q = h*real(mean( (exp(LR/2)-1)./LR ,2));
    f1 = h*real(mean( (-4-LR+exp(LR).*(4-3*LR+LR.^2))./LR.^3 ,2));
    f2 = h*real(mean( (2+LR+exp(LR).*(-2+LR))./LR.^3 ,2));
    f3 = h*real(mean( (-4-3*LR-LR.^2+exp(LR).*(4-LR))./LR.^3 ,2));

    % Main time-stepping loop:
    uu = u; tt = 0;
    tmax = 100; nmax = round(tmax/h); nplt = floor((tmax/250)/h); g = -0.5i*k;

    for n = 1:nmax
        t = n*h;
        Nv = g.*fft(real(ifft(v)).^2);
        a = E2.*v + Q.*Nv;
        Na = g.*fft(real(ifft(a)).^2);
        b = E2.*v + Q.*Na;
        Nb = g.*fft(real(ifft(b)).^2);
        c = E2.*a + Q.*(2*Nb-Nv);
        Nc = g.*fft(real(ifft(c)).^2);
        v = E.*v + Nv.*f1 + 2*(Na+Nb).*f2 + Nc.*f3;
        if mod(n,nplt)==0
            u = real(ifft(v));
            uu = [uu,u]; tt = [tt,t];
        end
    end

    % Plot the difference btw the result of Real Trajectory and NN result.

    figure(2*i+2)
    pcolor(x,tt,uu.'),shading interp, colormap(hot), axis off,
    title("Real Trajectory"),

    figure(2*i+3)
    uunn = zeros(N, 251); uunn(:,1) = u; x0 = u;
```

```
    for j = 2:length(tt)
        % y0 = net(x0);
        y0 = predict(net, x0, 'MiniBatchSize', 1);
        uunn(:,j) = y0; x0 = y0;
    end
    pcolor(x,tt,uunn.'),shading interp, colormap(hot), axis off,
    title("Trajectory with NN"),

end

%% SVD the result.

[U_in, S_in, V_in] = svd(training_input.');
[U_out, S_out, V_out] = svd(training_output.');

figure(1)
plot(diag(S_in), 'ro','LineWidth',[2])
figure(2)
plot(diag(S_out), 'bo','LineWidth',[2])
% 60 ranks.

%%
rank = 100;
input =  training_input.' \U_in(:, 1:rank);
output = training_output.' \U_out(:, 1:rank);

%%

net = feedforwardnet([10 20 10]);
net.layers{1}.transferFcn = 'logsig';
net.layers{2}.transferFcn = 'radbas';
net.layers{3}.transferFcn = 'purelin';
net = train(net, input.', output.');

%%

close all;
xx = 32*pi*(1:rank)'/rank;

for i = 1:10

    u = cos(rand(1,1)*x/16).*(1+sin(rand(1,1)*x/16)); % Change to random result.
    v = fft(u);

    % % % % % %
    %Spatial grid and initial condition:
    h = 0.025;
    k = [0:N/2-1 0 -N/2+1:-1]'/16;
    L = k.^2 - k.^4;
    E = exp(h*L); E2 = exp(h*L/2);
    M = 16;
    r = exp(1i*pi*((1:M)-.5)/M);
    LR = h*L(:,ones(M,1)) + r(ones(N,1),:);
    Q = h*real(mean( (exp(LR/2)-1)./LR ,2));
```

```
    f1 = h*real(mean( (-4-LR+exp(LR).*(4-3*LR+LR.^2))./LR.^3  ,2));
    f2 = h*real(mean( (2+LR+exp(LR).*(-2+LR))./LR.^3  ,2));
    f3 = h*real(mean( (-4-3*LR-LR.^2+exp(LR).*(4-LR))./LR.^3  ,2));

    % Main time-stepping loop:
    uu = u; tt = 0;
    tmax = 100; nmax = round(tmax/h); nplt = floor((tmax/250)/h); g = -0.5i*k;

    for n = 1:nmax
        t = n*h;
        Nv = g.*fft(real(ifft(v)).^2);
        a = E2.*v + Q.*Nv;
        Na = g.*fft(real(ifft(a)).^2);
        b = E2.*v + Q.*Na;
        Nb = g.*fft(real(ifft(b)).^2);
        c = E2.*a + Q.*(2*Nb-Nv);
        Nc = g.*fft(real(ifft(c)).^2);
        v = E.*v + Nv.*f1 + 2*(Na+Nb).*f2 + Nc.*f3;
        if mod(n,nplt)==0
            u = real(ifft(v));
            uu = [uu,u]; tt = [tt,t];
        end
    end

    % Plot the difference btw the result of Real Trajectory and NN result.

    % [u1, s1, v1] = svd(uu);

    figure(2*i+2)
    pcolor(xx,tt,(uu).'),shading interp, colormap(hot), axis off,
    title("Real Trajectory"),

    figure(2*i+3)
    % ux0 = uu\U_in(:, 1:rank);
    uunn = zeros(rank, 251); x0 = (u\U_in(:, 1:rank)).'; uunn(:,1) = x0;
    for j = 2:length(tt)
        y0 = net(x0);
        % y0 = predict(net, x0, 'MiniBatchSize', 1);
        uunn(:,j) = y0; x0 = y0;
    end
    % uunn = U_in(:, 1:rank) * uunn;
    pcolor(xx,tt,uunn.'),shading interp, colormap(hot), axis off,
    title("Trajectory with NN"),

end


%%


close all; clear all; clc

t=0:0.05:10;
d1=0.1; d2=0.1; beta=1.0;
```

```
L=20; n=512; N=n*n;
x2=linspace(-L/2,L/2,n+1); x=x2(1:n); y=x;
kx=(2*pi/L)*[0:(n/2-1) -n/2:-1]; ky=kx;

% INITIAL CONDITIONS

[X,Y]=meshgrid(x,y);
[KX,KY]=meshgrid(kx,ky);
K2=KX.^2+KY.^2; K22=reshape(K2,N,1);

m=1; % number of spirals

u = zeros(length(x),length(y),length(t));
v = zeros(length(x),length(y),length(t));

u(:,:,1)=tanh(sqrt(X.^2+Y.^2)).*cos(m*angle(X+i*Y)-(sqrt(X.^2+Y.^2)));
v(:,:,1)=tanh(sqrt(X.^2+Y.^2)).*sin(m*angle(X+i*Y)-(sqrt(X.^2+Y.^2)));

% REACTION-DIFFUSION
uvt=[reshape(fft2(u(:,:,1)),1,N) reshape(fft2(v(:,:,1)),1,N)].';
[t,uvsol]=ode45('reaction_diffusion_rhs',t,uvt,[],K22,d1,d2,beta,n,N);


for j=1:length(t)-1
ut=reshape((uvsol(j,1:N).'),n,n);
vt=reshape((uvsol(j,(N+1):(2*N)).'),n,n);
u(:,:,j+1)=real(ifft2(ut));
v(:,:,j+1)=real(ifft2(vt));

figure(1)
pcolor(x,y,v(:,:,j+1)); shading interp; colormap(hot); colorbar; drawnow;
end

save('reaction_diffusion_big.mat','t','x','y','u','v')

load reaction_diffusion_big
pcolor(x,y,u(:,:,end)); shading interp; colormap(hot)

%% Lorenz.

clear all, close all

% Simulate Lorenz system
dt=0.01; T=8; t=0:dt:T;
b=8/3; sig=10;
% r=10;
% r=28;
r=40;

Lorenz = @(t,x)([ sig * (x(2) - x(1))        ; ...
                  r * x(1)-x(1) * x(3) - x(2) ; ...
                  x(1) * x(2) - b*x(3)         ]);
ode_options = odeset('RelTol',1e-10, 'AbsTol',1e-11);
```

```
input=[]; output=[];
for j=1:100   % training trajectories
    x0=30*(rand(3,1)-0.5);
    [t,y] = ode45(Lorenz,t,x0);
    input=[input; y(1:end-1,:)];
    output=[output; y(2:end,:)];
    plot3(y(:,1),y(:,2),y(:,3)), hold on
    plot3(x0(1),x0(2),x0(3),'ro')
end
grid on, view(-23,18)

%%
net = feedforwardnet([10 10 10]);
net.layers{1}.transferFcn = 'logsig';
net.layers{2}.transferFcn = 'radbas';
net.layers{3}.transferFcn = 'purelin';
net = train(net,input.',output.');


%%
figure(2)

% r = 40;
% r=17;
r=35;

Lorenz = @(t,x)([ sig * (x(2) - x(1))        ; ...
                  r * x(1)-x(1) * x(3) - x(2) ; ...
                  x(1) * x(2) - b*x(3)         ]);
ode_options = odeset('RelTol',1e-10, 'AbsTol',1e-11);

x0=20*(rand(3,1)-0.5);
[t,y] = ode45(Lorenz,t,x0);
plot3(y(:,1),y(:,2),y(:,3)), hold on
plot3(x0(1),x0(2),x0(3),'ro','Linewidth',[2])
grid on

ynn(1,:)=x0;
for jj=2:length(t)
    y0=net(x0);
    ynn(jj,:)=y0.'; x0=y0;
end
plot3(ynn(:,1),ynn(:,2),ynn(:,3),':','Linewidth',[2])

figure(3)
subplot(3,2,1), plot(t,y(:,1),t,ynn(:,1),'Linewidth',[2])
subplot(3,2,3), plot(t,y(:,2),t,ynn(:,2),'Linewidth',[2])
subplot(3,2,5), plot(t,y(:,3),t,ynn(:,3),'Linewidth',[2])


figure(2)
x0=20*(rand(3,1)-0.5);
[t,y] = ode45(Lorenz,t,x0);
plot3(y(:,1),y(:,2),y(:,3)), hold on
```

```
plot3(x0(1),x0(2),x0(3),'ro','Linewidth',[2])
grid on

ynn(1,:)=x0;
for jj=2:length(t)
    y0=net(x0);
    ynn(jj,:)=y0.'; x0=y0;
end
plot3(ynn(:,1),ynn(:,2),ynn(:,3),':','Linewidth',[2])

figure(3)
subplot(3,2,2), plot(t,y(:,1),t,ynn(:,1),'Linewidth',[2])
subplot(3,2,4), plot(t,y(:,2),t,ynn(:,2),'Linewidth',[2])
subplot(3,2,6), plot(t,y(:,3),t,ynn(:,3),'Linewidth',[2])

%%
figure(2), view(-75,15)
figure(3)
subplot(3,2,1), set(gca,'Fontsize',[15],'Xlim',[0 8])
subplot(3,2,2), set(gca,'Fontsize',[15],'Xlim',[0 8])
subplot(3,2,3), set(gca,'Fontsize',[15],'Xlim',[0 8])
subplot(3,2,4), set(gca,'Fontsize',[15],'Xlim',[0 8])
subplot(3,2,5), set(gca,'Fontsize',[15],'Xlim',[0 8])
subplot(3,2,6), set(gca,'Fontsize',[15],'Xlim',[0 8])
legend('Lorenz','NN')
```