ECSE 6170 Modeling and Simulation for Cyber-Physical Systems
Spring 2021 Final Project
Control System Parameter Optimization with Deep Learning

Hayleigh Sanders

Abstract: The following project implemented a novel approach to control system optimization using deep learning. A neural network was trained to generate the transfer functions of the control system components based on a system response. Given an ideal system response, the neural network was able to predict control system parameters that resulted in a model which outperformed a conventionally tuned model in suppressing forced oscillations, achieving 98.42% suppression compared to 91.95% for the conventional model. However, a marginal amount of stability was lost in exchange for greater performance.

1 Introduction

Forced oscillations (FOs) pose a critical threat to synchronous power systems, particularly in those integrated with renewable energy generators. FOs in the 0.1 to 2Hz range are the most prevalent and problematic according to [1]. Trudnowski [2] proposes a feedback controller design where a counter-oscillation is induced 180 degrees out of phase with the FO, which cancels it out in a manner similar to ANC noise cancellation. The transfer functions of the control system components are determined mathematically based on the FO frequency ω_c. A fast Fourier transform (FFT) is utilized to obtain ω_c.

The purpose of this project is to evaluate a novel neural network based approach to control system parameter optimization. The simple linear example with $\omega_c = 1Hz$ given by Trudnowski [2] was used as the control system model. The neural network will be trained to predict the transfer functions of the control system components based on the system response. Therefore, given an ideal system response, the model will be able to predict optimal control system parameters. This method does not require knowledge of ω_c. A python script will interface with a Modelica control system model constructed in Dymola software to obtain the system response, and the neural network will be constructed using the Tensorflow package.

This approach is considered novel because no reference literature could be found employing a similar method for control system optimization. Shipman [3] used deep learning to tune the gain components of a PID controller. Likewise, Alwan [4] and Wang [5] trained a neural network to replace a controller in a digital feedback system.

2 Modeling Approach

2.1 Control System Model

2.1.1 Trudnowski Feedback Controller

The feedback controller proposed by Trudnowski [2] is given in Figure 1.

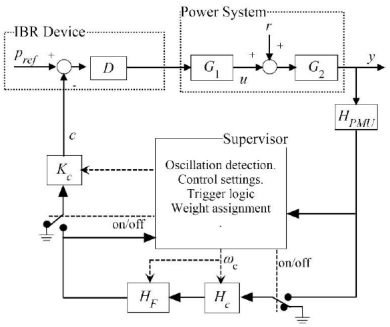

Figure 1: Feedback controller, Image credit: Trudnowski [2]

D is the transfer function representing the real-power injection device, such as a battery or IBR. G_1 and G_2 represent the transfer functions of an arbitrarily split power system, G_1G_2. A rogue FO injected between G_1 and G_2 in the power system is represented by r. The fundamental frequency of r is represented by ω_c. H_{PMU} is the transfer function of a measurement device, such as a PMU. H_F is the transfer function of a bandpass filter tuned to ω_c. H_C is the transfer function of the control compensator. K_C is the control gain. P_{ref} is the reference power of the IBR device. The modulation signal c is produced by the feedback controller. This signal induces a counter-oscillation u which is 180 degrees out of phase with the FO, which cancels it out. The power system response is measured at y.

The IBR device is modulated by the feedback controller signal c, then injects power into the power system containing a FO r. This FO is measured by the PMU at y, and sent through the filter, compensator and gain which produces the modulation signal c. The Supervisor detects a FO from the PMU measurement using a FFT, and enables the H_C, H_F and K_C components. Otherwise, these components are disabled.

The transfer function of the bandpass filter H_F is given by Equation 1.

$$H_F(S) = \frac{2\varsigma\omega_c s}{s^2 + 2\varsigma\omega_c s + \omega_c^2} \tag{1}$$

Where ς is the damping ratio. The phase-lead controller $H_C(s)$ is given by Equation 2.

$$H_C(S) = \frac{K(\alpha T s + 1)}{Ts + 1} \tag{2}$$

Where α and T are given in Equations 3 and 4.

$$\alpha = \frac{1 + \sin(-\angle G(j\omega_c))}{1 - \sin(-\angle G(j\omega_c))} \tag{3}$$

$$T = \frac{1}{\omega_c \sqrt{\alpha}} \tag{4}$$

The gain K_C is given in Equation 5.

$$K_C = \frac{|(1 + j\omega_c T)|}{|(1 + j\omega_c \alpha T) G(j\omega_c)|} \tag{5}$$

2.1.2 Simple Linear Power System Example

The transfer functions for a simple linear power system are given in Equations 6, 7 and 8.

$$D(s) = \frac{20}{s + 20} \tag{6}$$

$$G_1(s)G_2(s) = \frac{10}{s + 10} \tag{7}$$

$$H_{PMU}(s) = 1 \tag{8}$$

For a ω_c =1Hz oscillation, the control system parameters are given in Equations 9, 10 and 11.

$$K_c = 9.4 \tag{9}$$

$$Hf = \frac{1.26s}{s^2 + 1.25s + 39.5} \tag{10}$$

$$Hc = \frac{0.455(0.432s + 1)}{0.0586s + 1} \tag{11}$$

The induced oscillation (c) and resulting control system response (y) is shown in Figure 2.

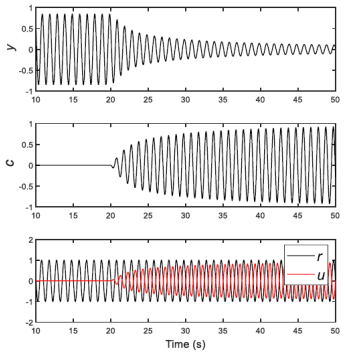

Figure 2: Simulation result, Image credit: Trudnowski [2]

2.2 Modelica Model and Package Structure

The control system was constructed in Modelica as shown in Figure 3, which replicates the example model given in Section 2.1.2. The transfer functions for D, G and H_{PMU} are given in Equations 6, 7 and 8. A sine wave with amplitude 1 and frequency 1Hz represents the FO from the example model. The reference input is a step function of height 0. The Supervisor component is not used in this model.

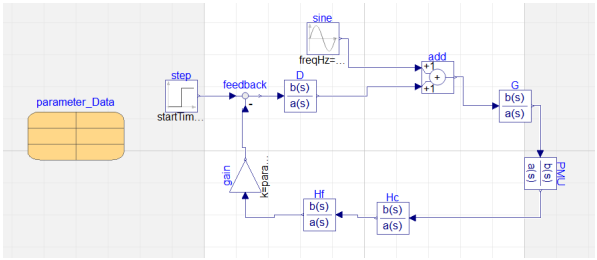

Figure 3: Modelica control system model

The model has the package structure given in Figure 4. The parameter "gain" is the value for the gain component. The parameters "HfNum" and "HfDenom" are the numerator and denominator of the transfer function for the filter H_f. The parameters "HcNum" and "HcDenom" are the numerator and denominator of the transfer function for the compensator H_c. These are declared as replaceable records so they can be changed during dynamic simulation.

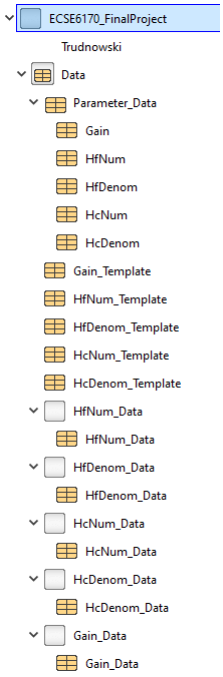

Figure 4: Model package structure

The parameters in the *parameter_Data* block are shown in Figure 5.

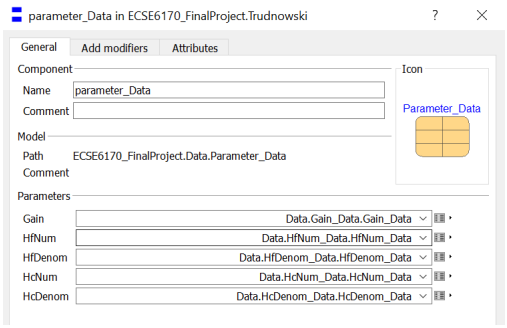

Figure 5: Parameter Data block

The parameters for H_f, H_c and *gain* blocks are given in Figure 6.

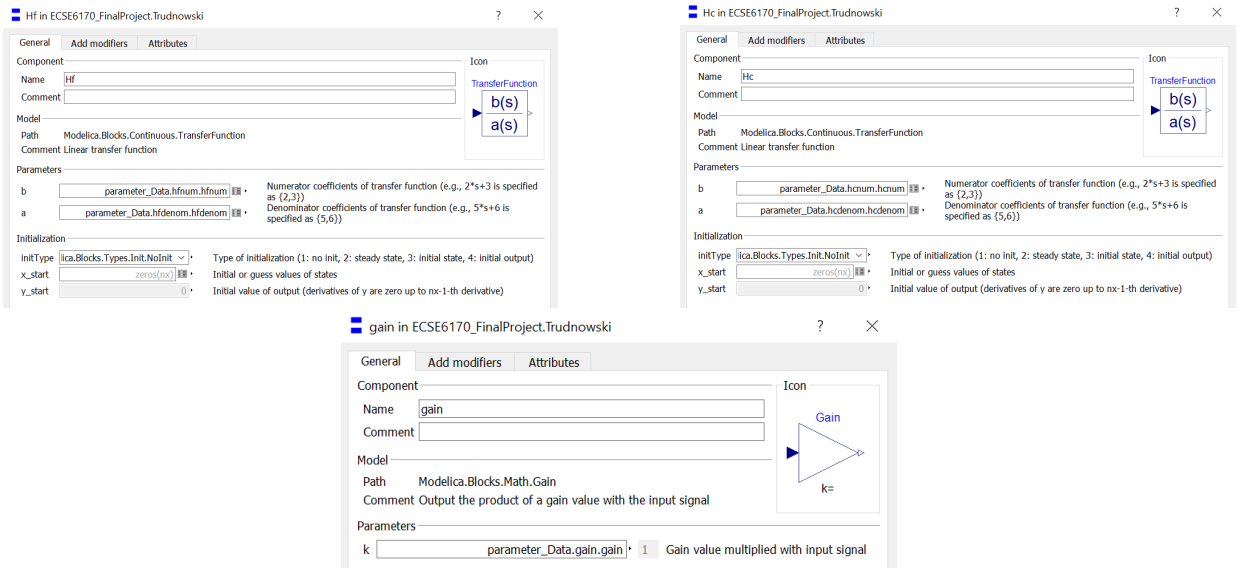

Figure 6: Control system component parameters

2.3 Python-Dymola Interface

A python script was written to generate data, then build and evaluate the neural network model using Tensorflow.

The get_ybar() function accepts the arguments (gain,a,b,c,d,e,f,g,h,i,j,k,l) corresponding to the coefficients from equations 1, 2 and 5 when expressed in Laplace form. The function writes the coefficients to the Data.mo file in the Modelica package structure as shown in Figure 7.

```
lines[91] = "      extends ECSE6170_FinalProject.Data.Gain_Template(gain="+str(gain)+");\n"
lines[67] = "      extends ECSE6170_FinalProject.Data.HfNum_Template(hfnum={"+str(a)+","+str(b)+","+str(c)+"});\n"
lines[73] = "      extends ECSE6170_FinalProject.Data.HfDenom_Template(hfdenom={"+str(d)+","+str(e)+","+str(ff)+"});\n"
lines[79] = "      extends ECSE6170_FinalProject.Data.HcNum_Template(hcnum={"+str(g)+","+str(h)+","+str(i)+"});\n"
lines[85] = "      extends ECSE6170_FinalProject.Data.HcDenom_Template(hcdenom={"+str(j)+","+str(k)+","+str(l)+"});\n"
```

Figure 7: Text modification of the Data.mo file

The function then simulates the model as shown in Figure 8. The simulation runs from 0 to 20 seconds using the "Dassl" solver and a tolerance of 0.0001.

```
result = dymola.simulateModel("ECSE6170_FinalProject.Trudnowski",0,20,method='Dassl',tolerance=0.0001,resultFile="TrudnowskiTest")
if not result:
    print("!: Simulation failed")
    log = dymola.getLastErrorLog()
    print(log)
```

Figure 8: Execute Dymola simulation

The simulation result is stored in the file "TrudnowskiTest.mat". The code given in Figure 9 reads in the data corresponding to the PMU output (PMU.y). The data is returned as an array of size 502.

```
mat = scipy.io.loadmat("C:/Users/Hayleigh Sanders/Documents/Dymola/TrudnowskiTest.mat")

for i in mat["data_2"][10]:
    q.append(i)
```

Figure 9: Read data to an array

To improve the program runtime, the get_ybar() function was executed with multithreading.

2.4 Neural Network Model

2.4.1 Training Data Generation

The transfer functions for the gain, Hf and Hc can be expressed in terms of their unique coefficients given by Equations 12, 13 and 14.

$$K_c = \alpha \tag{12}$$

$$H_f(s) = \frac{a * s^2 + b * s + c}{d * s^2 + e * s + f} \tag{13}$$

$$H_c(s) = \frac{g * s^2 + h * s + i}{j * s^2 + k * s + l} \tag{14}$$

The training data set is formed from n feature vectors and n corresponding label vectors, where n is the number of training samples. The label vectors $Y[n]$ consist of the 13 coefficients from Equations 12, 13 and 14. Each label was generated from a normal distribution $N(y, 5)$, where y is the value of each control system parameter from the original model given in Equations 9, 10 and 11.

$$Y[n] = \{\alpha_n, a_n, b_n, c_n, d_n, e_n, f_n, g_n, h_n, i_n, j_n, k_n, l_n\} \tag{15}$$

Using the n label vectors, n feature vectors were generated by simulating the model. Each simulation yields a vector of length 502 representing the time series output signal of the PMU component (PMU.y).

$$X[n] = \{x_1, x_2, x_3 ... x_{502}\} \tag{16}$$

2.4.2 Neural Network Architecture

A neural network with one input layer, two hidden layers and one output layer was constructed using Tensorflow, which can be visualized in Figure 10. The input layer accepts a tensor of size $(n, 502)$ and the output layer produces a tensor of size $(n, 13)$. A weight matrix exists between each layer. The weights are initialized to random values and an initial prediction $\hat{y}[n]$ for each $x[n]$ is produced. This output is used to produce the output gradient, which is backpropagated through each layer of the neural network to obtain the weight gradients for each layer. Based on these gradients, the weight matrices are updated and the process is repeated for a number of iterations.

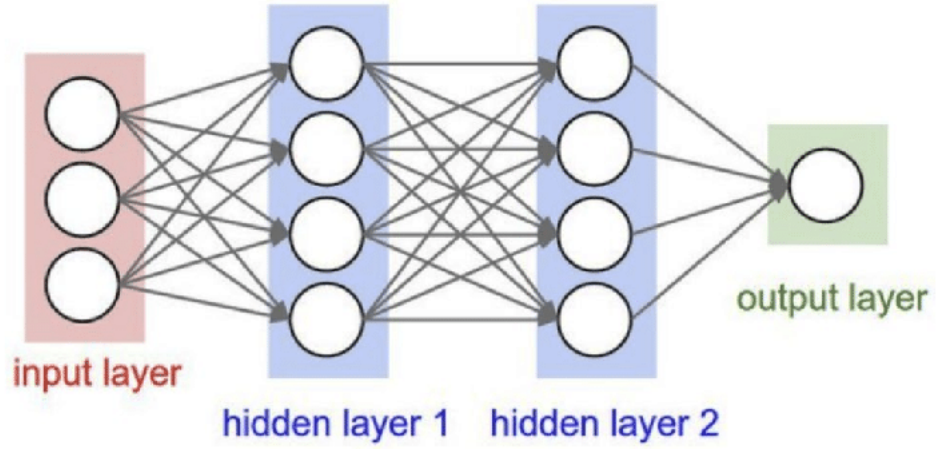

Figure 10: Neural network with two hidden layers, Image credit: Gaudio et al [6]

2.4.3 Loss Function

This project utilized the Huber loss function as shown in Equation 17. The Huber loss was chosen because it is robust against outlier data. This is important because the dataset contained a small number of bad training examples (ie the signal response would explode to very large values) that persist despite an attempt to reject unsuitable data.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2} * (y - f(x))^2 & for\ |y - f(x)| \leq \delta, \\ \delta * |y - f(x)| - \frac{1}{2} * \delta^2 & otherwise \end{cases} \tag{17}$$

2.4.4 Forward Pass

Starting at the input layer, for each $x[n]$ in the training data set of size n, its corresponding output $\hat{y}[n]$ is computed from the current weights $W = [W^1, W^2, W^3]$ and their biases W_0^1, W_0^2, W_0^3 with the following:

$$H^1[n] = ReLu((W^1)^T * x[n] + W_0^1) \tag{18}$$

$$H^2[n] = ReLu((W^2)^T * H^1 + W_0^2) \tag{19}$$

$$\hat{y}[n] = ReLu((W^3)^T * H^2 + W_0^3) \tag{20}$$

The ReLu function is given by,

$$ReLu(z) = max(0, z) \tag{21}$$

This process is split into multiple steps:

$$Z^1 = W^1 X + W_{0,1} \tag{22}$$

$$H^1 = ReLu(Z^1) \tag{23}$$

$$Z^2 = W^2 H^1 + W_{0,2} \tag{24}$$

$$H^2 = ReLu(Z^2) \tag{25}$$

$$Z^3 = W^3 H^2 + W_{0,3} \tag{26}$$

$$\hat{Y} = ReLu(Z^3) \tag{27}$$

2.4.5 Backward Pass

Starting at the output layer, the output gradient is computed and passed over the neural network layers. The weight gradients were found recursively with the following process.

Find the error at the output layer:

$$dZ^3 = (\hat{Y} - Y) \tag{28}$$

Find the gradients at the output layer:

$$dW^3 = (1/n) * dZ^3 * H^{2T} \tag{29}$$

$$dW_{0,3} = (1/n) * sum(dZ^3) \tag{30}$$

Backpropagate through the second layer:

$$dH^2 = W^{3T} * dZ^3 \tag{31}$$

$$dZ^2 = dH2 * dReLu(Z^2) \tag{32}$$

Find the gradients at the second layer:

$$dW^2 = (1/n) * dZ^2 * H^{1T} \tag{33}$$

$$dW_{0,2} = (1/n) * sum(dZ^2) \tag{34}$$

Backpropagate through to the first layer:

$$dH^1 = W^{2T} * dZ^2 \tag{35}$$

$$dZ^1 = dH^1 * dReLu(Z^1) \tag{36}$$

Find the gradients at the first layer:

$$dW^1 = (1/n) * dZ^1 * X^T \tag{37}$$

$$dW_{0,1} = (1/n) * sum(dZ^1) \tag{38}$$

Where dReLu is the derivative of the Relu function given by,

$$dReLu(z) = \begin{cases} 0 & z < 0 \\ 1 & z > 0 \\ undefined & z = 0 \end{cases} \tag{39}$$

The weights and their biases for each layer are then updated using the gradients found in the backward pass process with the following:

$$W^1 = W^1 - \eta dW^1 \tag{40}$$

$$W_{0,1} = W_{0,1} - \eta dW_{0,1} \tag{41}$$

$$W^2 = W^2 - \eta dW^2 \tag{42}$$

$$W_{0,2} = W_{0,2} - \eta dW_{0,2} \tag{43}$$

$$W^3 = W^3 - \eta dW^3 \tag{44}$$

$$W_{0,3} = W_{0,3} - \eta dW_{0,3} \tag{45}$$

$$W^4 = W^4 - \eta dW^4 \tag{46}$$

$$W_{0,4} = W_{0,4} - \eta dW_{0,4} \tag{47}$$

$$W^5 = W^5 - \eta dW^5 \tag{48}$$

$$W_{0,5} = W_{0,5} - \eta dW_{0,5} \tag{49}$$

Where η is the learning rate.

2.4.6 Stochastic Gradient Descent

Stochastic gradient descent (SGD) was employed through the use of the Tensorflow "Adam" optimizer to increase the learning efficiency. The SGD algorithm is given in Equation 50.

$$W_k^{t+1} = W_k^{t+1} - \eta * [\frac{1}{S} * \sum_{x[m],y[m] \in D} \frac{\partial L(x[m]y[m]W)}{\partial W}] \tag{50}$$

Instead of using the entire dataset to compute the gradients, the algorithm uses a subset of the data.

2.4.7 Model Hyperparameters

The model hyperparameters were chosen by trial and error to produce a result that minimized the total MAPE (mean absolute percentage error). Two hidden layers were used for the neural network because according to Vamvoudakis [7], a two-layer neural network is sufficient for feedback control purposes. 150 nodes were chosen for each hidden layer and the learning rate was set to the default $\eta = .001$ for the Tensorflow "Adam" optimizer. 6,000 training data X,Y pairs were generated and the model was trained for 50 epochs.

2.4.8 Training Data Screening

The training data contained a small number of bad examples that caused the system response to explode into very large values, preventing the neural network from training properly. As a result, training data containing $X[n]$ values greater than 10^8 were rejected. This resulted in a new dataset of size $n = 5,950$.

3 Analysis and Results

3.1 Model Evaluation Method

Model performance will vary among training sessions. For this reason, K-fold cross validation was utilized to evaluate the model. First, the dataset is split into k equally sized partitions, referred to as "folds". The first fold is retained as testing data and the other k-1 folds are used as training

data for a model instance. The model is trained and evaluated, then discarded. The next fold is used as testing data with a new model instance and the cycle is repeated until each fold is used as validation data. The performance results from each iteration are averaged. This project used k = 10.

3.2 Model Evaluation Metrics

The metrics given in Equations 51, 52 and 53 were utilized to evaluate the model performance. The root mean-square error (RMSE) is given by,

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{51}$$

The mean absolute error (MAE) is given by,

$$MAE = \frac{\sum_{i=1}^{n}|\hat{y}_i - y_i|}{n} \tag{52}$$

The mean absolute percentage error (MAPE) is given by,

$$MAPE = \frac{\sum_{i=1}^{n}\frac{y_i - \hat{y}_i}{y_i}}{n} \tag{53}$$

3.3 Results

3.3.1 Original Neural Network Design

The training loss over 50 epochs is given in Figure 11.

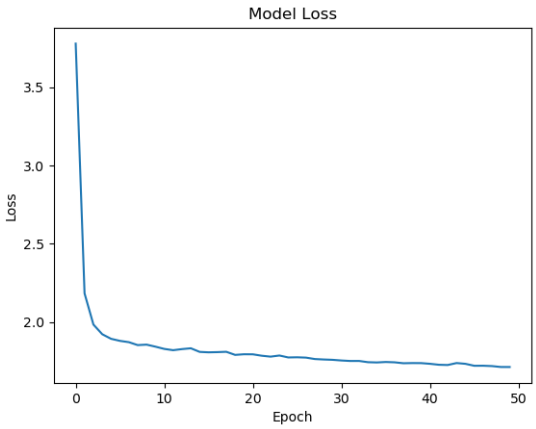

Figure 11: Model training loss vs epochs

Because a number of parameter configurations can produce a similar control system response, the original response was compared with the one produced by predicted parameters. For each test data vector X, the trained model produced a vector of predicted parameters $\hat{Y}$. The Modelica model of the control system was simulated using these parameters in order to produce the response signal $\hat{X}$. The actual vs predicted control system response for several validation data examples is shown in Figure 12.

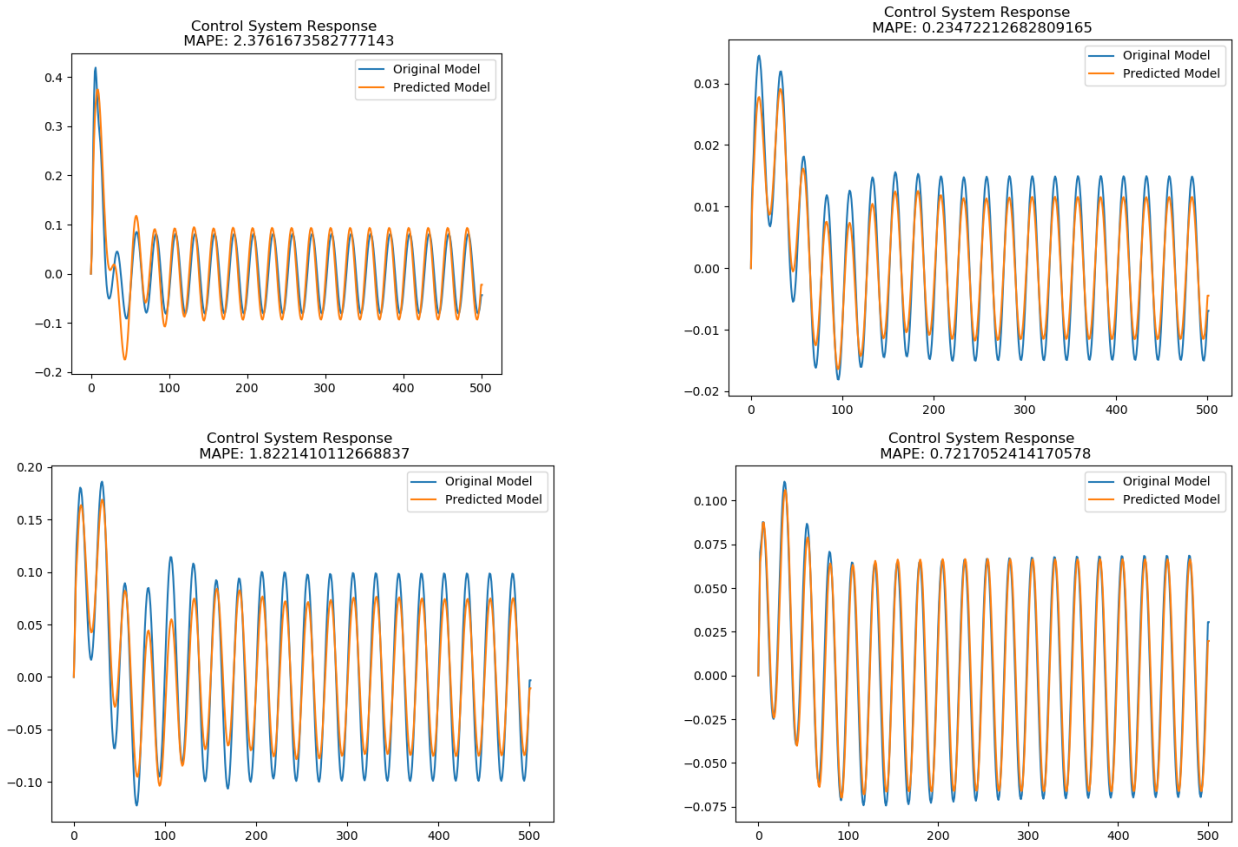

Figure 12: Actual vs predicted control system response

Using the K-fold cross validation procedure outlined in Section 3.1, the evaluation metrics described in Section 3.2 were applied to the training data for each $(X,\hat{X})$. The results are summarized in Table 1.

Table 1: Model Evaluation Results

MAPE (%)	MAE	RMSE	Huber Loss
4.92%	0.04	0.05	1.76

With a MAPE of 4.92%, the model can be considered highly accurate in its prediction of the feedback control system transfer functions given a system response.

Given a desired system response of $Z = [0_1, 0_2, 0_3 ... 0_{502}]$, which is equivalent to the reference signal, the model predicted the following parameters:

$$K_c = 14.25 \tag{54}$$

$$Hf = \frac{6.46s^2 + 4.98s + 6.02}{2.57s^2 + 3.79s + 37.68} \tag{55}$$

$$Hc = \frac{6s^2 + 4.97s + 7.22}{0.61s^2 + 0.55s + 0.11} \tag{56}$$

The resulting system response is plotted below in Figure 13 alongside the system response using the original parameters from Equations 9, 10 and 11.

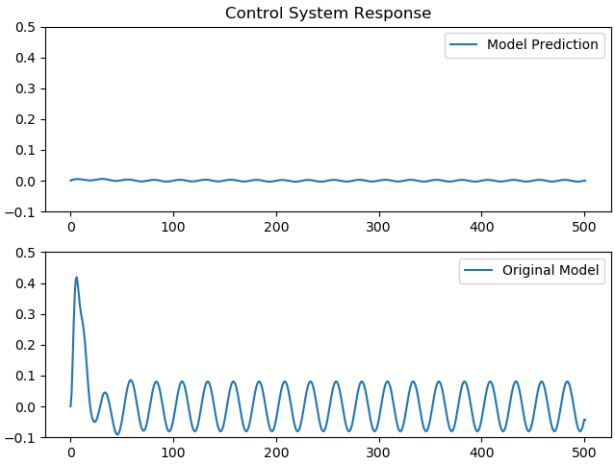

Figure 13: Optimized control system output vs original model output

The optimized model demonstrates 99.77% FO suppression, compared to 91.95% FO suppression from the original model. However, a critical flaw with this model was revealed upon further analysis. The state space matrices [A,B,C,D] for the model were obtained in Dymola using the Full Linear Analysis tool. The condition number of the A matrix was then computed. The condition number of a matrix determines the sensitivity of the system to small changes in the input data (ie noise). In this case, the condition number was revealed to be $2.02 * 10^4$. This is indicative of ill-conditioning - that is, small perturbations in the input data due to noise or some other source will produce large changes in the output signal.

According to Brown [8], random noise associated with a field PMU can range from 0.5% to 15%, so one could expect up to 101%-3030% error on the output. This means that one could expect as low as only 77.66% FO suppression in the least noisy case, and up to 6.7 times magnification of the FO in the most noisy case, making the problem even worse. For reference, the condition number of the original model is $1.57 * 10^2$, corresponding to up to 0.785%-23.55% error on the output. This means that one could expect as low as 91.88% FO suppression in the least noisy case, and as low as 90.06% FO suppression in the most noisy case.

3.3.2 Revised Neural Network Design

Clearly, some changes had to be made to the neural network to compensate for A-matrix conditioning. The goal of this modification is to find an optimized model with an A-matrix condition number of order 10^2 or less. According to Alwan [4], increasing the number of hidden layers from two to four resulted in greater system stability. This method was attempted, but did not produce results with greater stability, so the original two-layer architecture was maintained.

The feature vector $X[n]$ was expanded with an entry for the A-matrix condition as shown in Equation 57. This will allow the neural network to learn the relationship between both the system stability and system response to the control system parameters. All other aspects of the neural network architecture and evaluation procedure remained unchanged.

$$X[n] = \{cond(A), x_1, x_2, x_3 ... x_{502}\} \tag{57}$$

In the get_ybar() function, the dymola.linearizeModel command was used to obtain the state space matrices of each X,Y test data pair as shown in Figure 14.

```
result_lin = dymola.linearizeModel("ECSE6170_FinalProject.Trudnowski",startTime=0.0,stopTime=20.0,method='Dassl',tolerance=0.0001,resultFile='dslin')
if not result_lin:
    print("1: Linearization failed")
```

Figure 14: Model linearization

The numpy.linalg.cond() function was used to obtain the condition number from the A matrix as shown in Figure 15, and this condition number was appended to the beginning of the feature vector.

```
dslin = scipy.io.loadmat("C:/Users/Hayleigh Sanders/Documents/Dymola/dslin.mat")
c = LA.cond(dslin['ABCD'])
```

Figure 15: Obtaining the condition number

The improved model was trained and the loss vs epochs is shown in Figure 16.

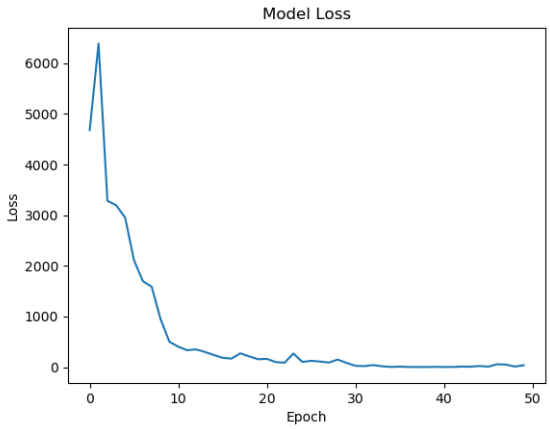

Figure 16: Loss vs epochs for the improved model

The improved model was then evaluated using the same metrics and the results are shown in Table 2.

Table 2: Improved Model Evaluation Results

MAPE (%)	MAE	RMSE	Huber Loss
10.07	.10	.12	3.52

Although the MAPE appeared to roughly double compared to the previous neural network design, the model performance is still acceptably accurate.

The ideal condition number of a system is 1. Given a desired system response of $Z = [1_1, 0_2, 0_3 ... 0_{503}]$, the improved model predicted the following parameters:

$$K_c = 22.06 \tag{58}$$

$$Hf = \frac{0.72s^2 + 13.24s + 12.91}{2.51s^2 + 4.65s + 84.72} \tag{59}$$

$$Hc = \frac{4.88s^2 + 3.21s + 5.37}{4.19s^2 + 2.17s + 3.94} \tag{60}$$

The resulting system response is plotted below in Figure 17 alongside the system response using the original parameters from Equations 9, 10 and 11.

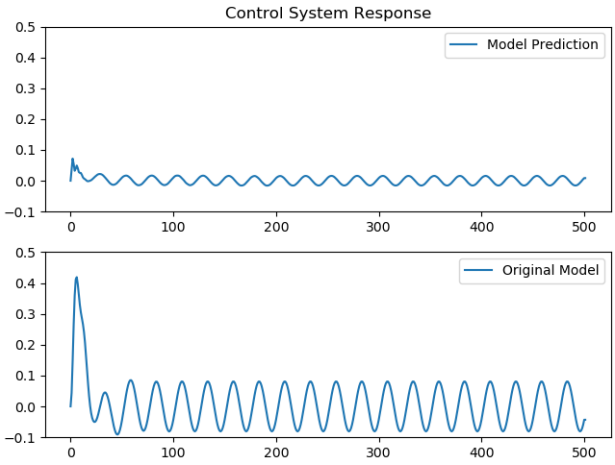

Figure 17: Optimized control system output vs original model output

The model demonstrated 98.42% FO suppression, and the condition of the A matrix was found to be $3.15 * 10^2$. This means one could expect up to 15.75% to 47.25% error in the output. This corresponds to as low as 98.18% FO suppression in the least noisy case, and as low as 97.68% FO suppression in the most noisy case, so this control system can be considered acceptably stable. These results are summarized in Table 3.

Table 3: Control System Performance

FO Suppression	Noiseless Case	0.5% Noise Case	15% Noise Case
Original Model	91.95%	91.88%	90.06%
Optimized Model	98.42%	98.18%	97.68%

4 Conclusion

This project successfully demonstrated that a neural network based approach to control system parameter optimization can produce acceptably stable results with a high level of FO suppression given no prior knowledge of the FO frequency ω_c. The optimal model found by this method outperformed a control system tuned by conventional means, although a marginal amount of stability was sacrificed in exchange for greater FO suppression. The model was able to predict the control system parameters based on a system response with high accuracy. However, finding the optimal parameters to an acceptably stable control system required the addition of the condition of the system A matrix to the feature vector.

A major hindrance to this project was the slow speed of each Dymola simulation, even with the use of multithreading. Using the Python-Dymola interface, each call to dymola.simulateModel took about 8 seconds. This means that it took about 14 hours to generate 6,000 training data points, and another 14 hours to evaluate the neural network model.

References

[1] R. Xie and D. J. Trudnowski, "Distinguishing features of natural and forced oscillations," *Proceedings of the IEEE Power Energy Society General Meeting*, 2015.

[2] D. J. Trudnowski and R. Guttromson, "A Strategy for Forced Oscillation Suppression," *IEEE Transactions on Power Systems*, vol. 35, no. 6, pp. 4699–4708, 2020.

[3] W. J. Shipman, "Learning to Tune a Class of Controllers with Deep Reinforcement Learning," *Minerals*, vol. 11, no. 9, p. 989, 2021.

[4] N. A. S. Alwan and Z. M. Hussain, "Deep Learning Control for Digital Feedback Systems: Improved Performance with Robustness against Parameter Change," *Electronics*, vol. 10, no. 11, p. 1245, 2021.

[5] Y. Wang, K. Velswamy, and B. Huang, "A Novel Approach to Feedback Control with Deep Reinforcement Learning," *IFAC-PapersOnLine*, vol. 51, no. 18, pp. 31–36, 2018.

[6] M. T. Gaudio, G. Coppola, L. Zangari, and S. Curcio, "Artificial Intelligence-Based Optimization of Industrial Membrane Processes," *Earth Systems and Environment*, vol. 5, no. 2, p. 3, 2021.

[7] K. G. Vamvoudakis, F. L. Lewis, and S. S. Ge, " Neural Networks in Feedback Control Systems," *Volume II. Design, Instrumentation, and Controls Part 2. Instrumentation, Systems, Controls, and Mems*, 2015.

[8] M. Brown, M. Biswal, S. Brahma, S. J. Ranade, and H. Cao, " Characterizing and quantifying noise in pmu data," *IEEE Power and Energy Society General Meeting (PESGM)*, pp. 1–5, 2016.