Design of a Mass-Spring-Damper System Controller for Position Control Using a Neural Network

Hamid Nakhaei
Industrial Engineering Department, Sharif University of Technology, Tehran, Iran

1hamidnakhaei@gmail.com

In this project, an attempt is made to control a single degree of freedom mass-spring-damper dynamic system, which was previously modeled using a neural network. The designed controller is also a neural network. All simulations are performed in the MATLAB/Simulink environment. The controller training is generally unsupervised, with neural network weights chosen randomly. However, for time efficiency, an On-Off controller was initially used, and this controller was trained using supervised learning. Subsequently, unsupervised learning began. As a result, the neural network successfully optimized the control of the system.

Key words: Neural Network ; Mechanics ; Controller ; Mass-Spring-Damper

1. Introduction

Fig. 1 shows an overview of the dynamic system under study.

The system specifications are shown in Eq. (1).

$$\begin{aligned} K &= 24 \ {}^{\text{N}}\!/\!_{\text{m}} \\ b &= 8 \ {}^{\text{N.s}}\!/\!_{\text{m}} \\ m &= 25 \ \text{kg} \\ F &= 120 \ \text{N} \end{aligned} \tag{1}$$

The dynamic motion of this system is described by the differential equation given in Eq. (2).

$$m\ddot{x}(t) + kx(t) + b\dot{x}(t) = F(t) \tag{2}$$

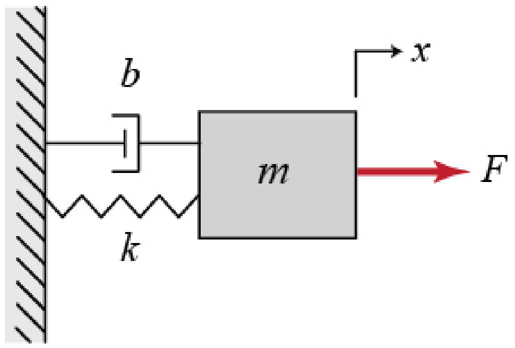

Figure 1 Overview of the Mass-Spring-Damper System

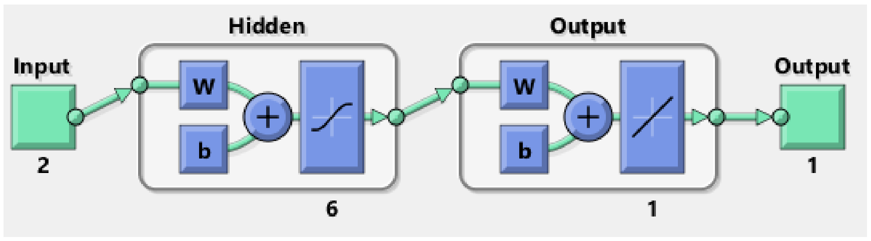

Figure 2 Neural Network Structure for Modeling the Dynamic Behavior of the System

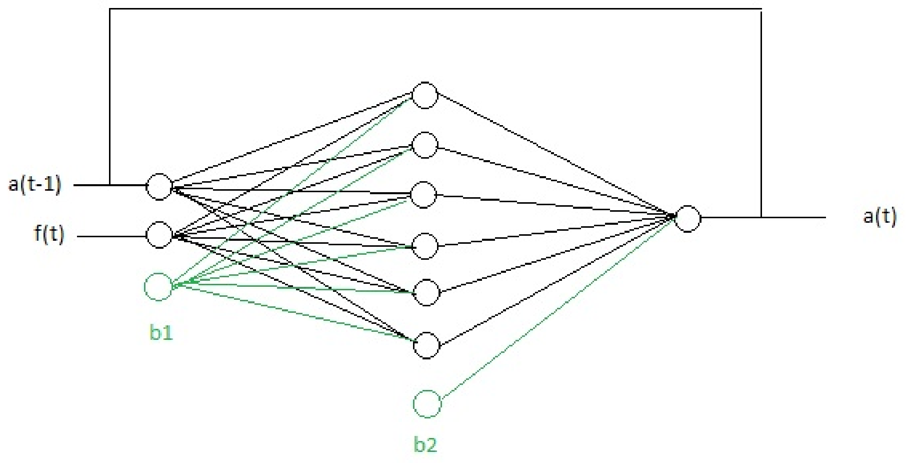

Figure 3 Neural Network Used for Modeling the Dynamic Behavior of the System

It is assumed that the damper and spring exhibit linear behavior and their properties do not change over time. Fig. 2 illustrates the structure of the neural network used for simulating the dynamic behavior of the system.

The inputs to the neural network are the acceleration at the previous time and the current force, and its output is the current acceleration. Fig. 3 demonstrates this concept clearly.

The neural network is able to successfully be trained with a sawtooth wave input and be tested with a square wave input. In this paper, a neural network is used to control the system. The training of the neural network controller is unsupervised, meaning no specific and predetermined data exists as input and desired output for the network to learn from. However, for time-saving and to guide the unsupervised learning process, the neural network was initially trained with an On-Off controller in a supervised manner. Then, the unsupervised learning process started to achieve the ideal controller.

1.1. Neural Network Configuration and Assumptions

The considered neural network for controller training includes two inputs, six neurons in the hidden layer, and one output. The first input is the difference between the desired position and the current

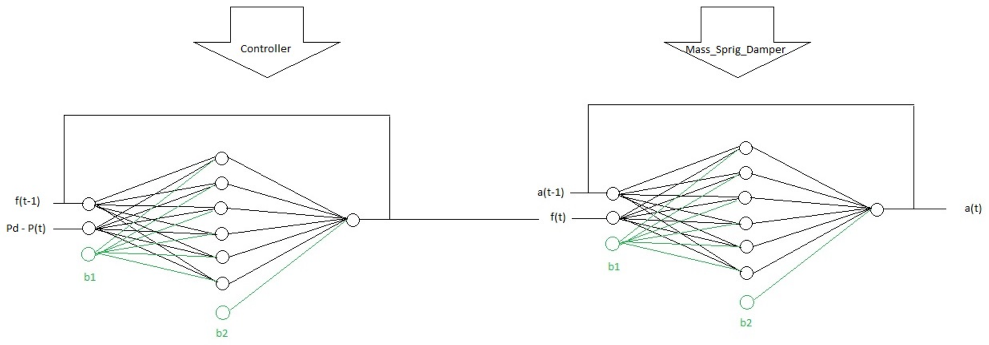

Figure 4 Overview of the Controller Neural Network and the Dynamic Model Neural Network

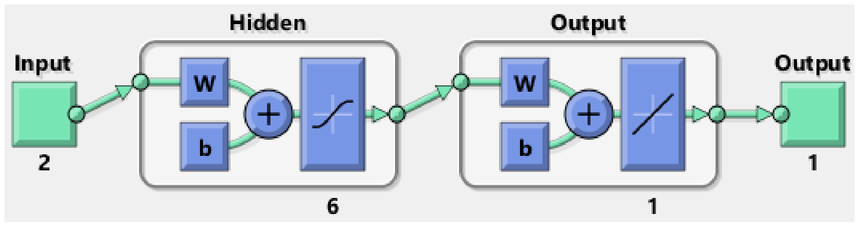

Figure 5 Controller Neural Network

position of the mass at time t, and the second input is a feedback from the force at the previous time $t-1$. The output of the neural network is the force at time t. Fig. 4 shows an overview of the controller neural network and the dynamic model neural network.

The goal of this project is to train the controller neural network. Whenever the term neural network is used, it refers to the controller neural network. Fig. 5 shows the controller neural network.

As shown, the network has two inputs and one output. All biases related to the hidden layer (represented as a column matrix) are denoted by b_1, and biases related to the output layer are denoted by b_2. The activation function in the first layer is a sigmoid function, and in the second layer, it is a linear function with a 45-degree slope. The learning algorithm is the Levenberg-Marquardt backpropagation, chosen for its lower memory usage and typically faster learning speed. The learning method is batch learning, with a maximum of 1000 epochs and a learning rate of 0.01. The error measurement function is the mean squared error (MSE) between the neural network output and the desired value. The simulation time for control is set to 30 seconds, with the controller training targeting a desired position of $p = 5\text{m}$.

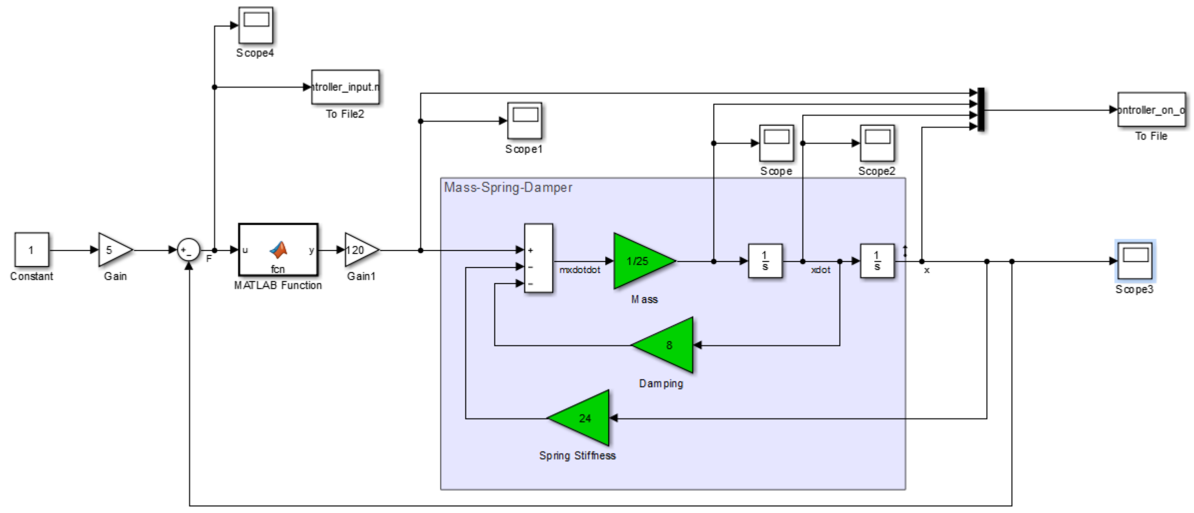

Figure 6 System and Controller Circuit Simulated in Simulink

2. Supervised Learning

Initially, a simple On-Off controller is placed as the system controller. To extract the On-Off controller data, this simple controller is defined in a MATLAB function and placed in the circuit. Fig. 6 shows the system circuit simulated in Simulink.

As observed, the blue rectangle models the mass-spring-damper, with feedback taken from the system position (p). This position is compared with the desired position ($p_d = 5$), and the difference is input to the function. This function is the On-Off controller. Data extracted from this simulation include:

- Difference between desired position and current position (controller input)
- Force applied to the mass-spring-damper system (controller output)
- Acceleration of the mass at each moment
- Speed of the mass at each moment
- Position of the mass at each moment

Figures 7 to 11 show graphs of position, speed, acceleration, force, and position error ($p_d - p$) over time.

Now, with the controller's input and output data, supervised training can begin. Supervised training of the controller has two phases. In the first phase, the force feedback is disconnected, and the system is trained with the actual force feedback. In the second phase, the force feedback is taken from the neural network output.

Results of On-Off controller training in the first training phase are as follows: Fig. 12 shows the R-value for the regression between the neural network output and the desired output. $R = 1$ occurs when all neural network output data lie on the 45-degree line. As seen, the R-value is 0.94524.

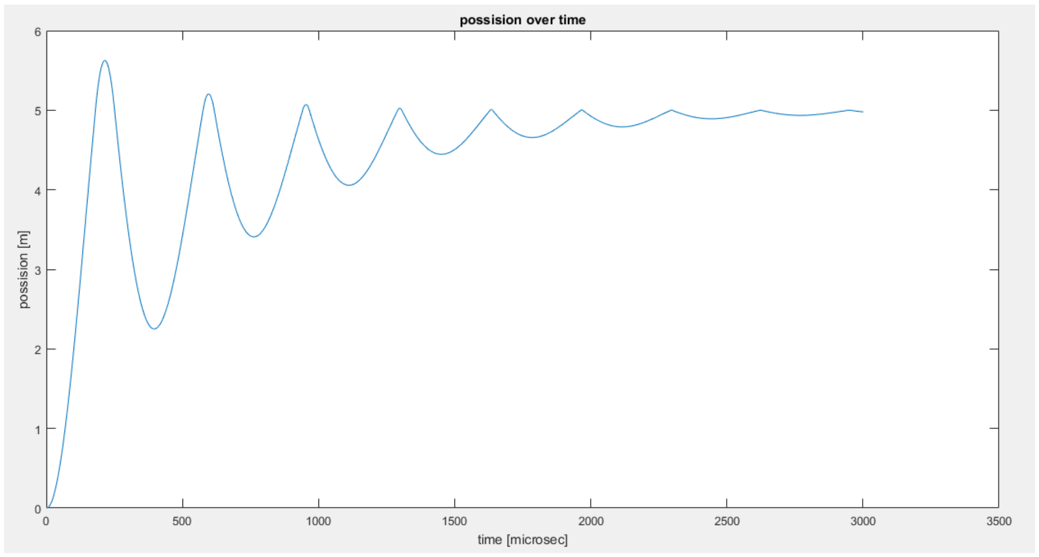

Figure 7 Position Over Time (On-Off Controller)

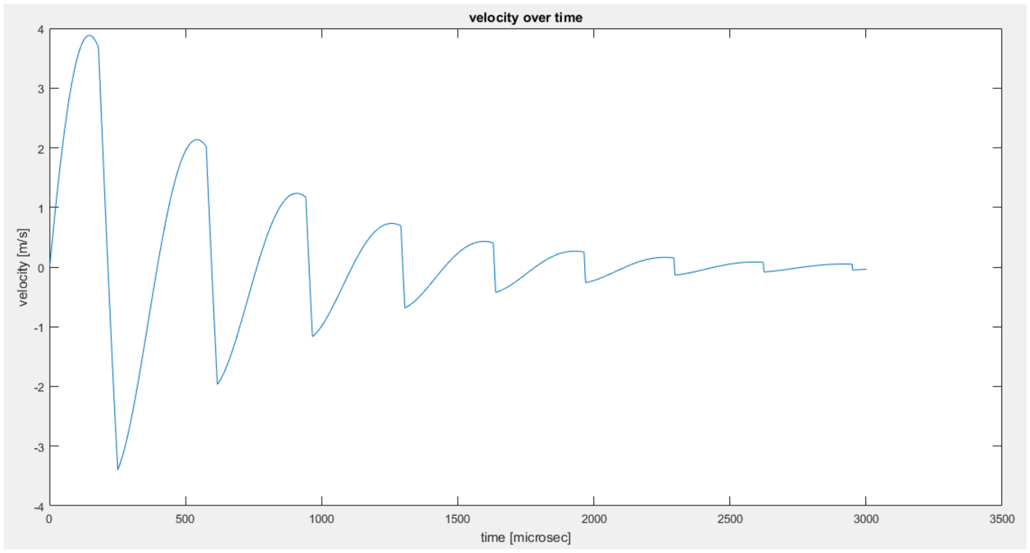

Figure 8 Speed Over Time (On-Off Controller)

In this training phase, 10% of the data is used for testing the learning progress, another 10% for validation, and 80% for training. During training, if the network is biased or not trained appropriately or gets stuck in local minima, validation data is used to prevent biased training. Training stops if validation errors reach 10. The momentum value, a learning parameter, prevents getting stuck in local minima. Fig. 13 shows gradient, momentum, and validation error over epochs during On-Off controller training (real feedback).

Fig. 14 shows mean squared errors (MSE) over epochs for training, testing, and validation data during On-Off controller training (real feedback).

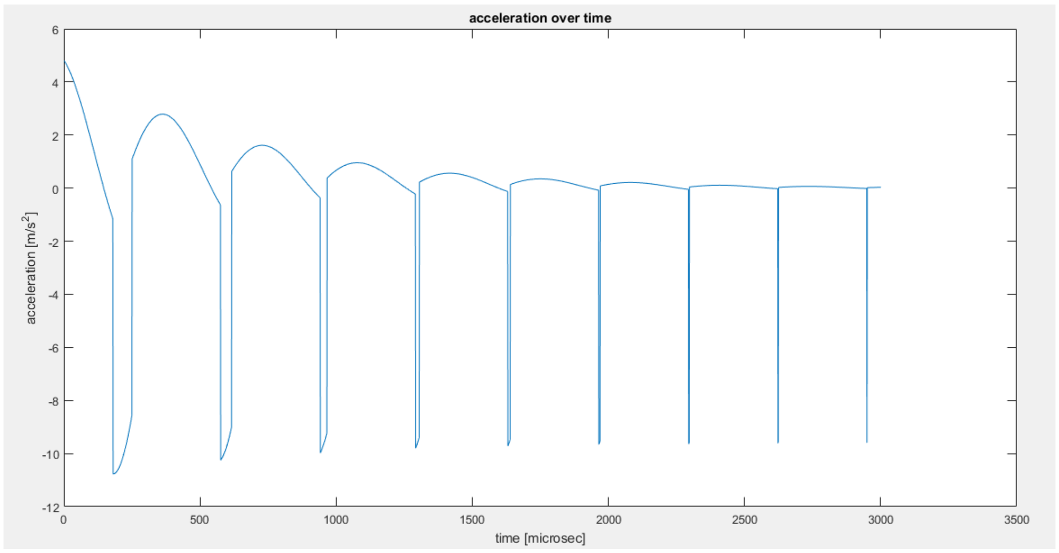

Figure 9 **Acceleration Over Time (On-Off Controller)**

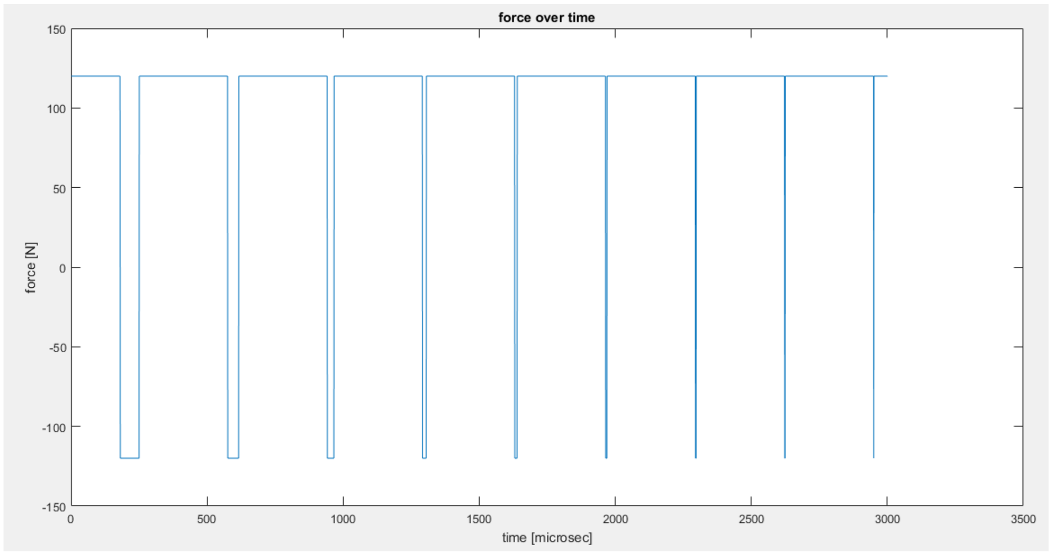

Figure 10 **Force Over Time (On-Off Controller)**

Training is done in 20 epochs. Finally, Fig. 15 compares the neural network output with the expected output. The neural network output is shown in green and continuous, while the expected output is blue and dashed.

In the second training phase, the neural network output is used as feedback. The configuration remains the same, and weights obtained from the previous network are used as initial weights. All data are used for training, with no data for testing or validation. Figures 16 to 19 show the training results.

The R-value is 0.99871, an acceptable value. Gradient, momentum, and validation error over epochs are shown in Fig. 17.

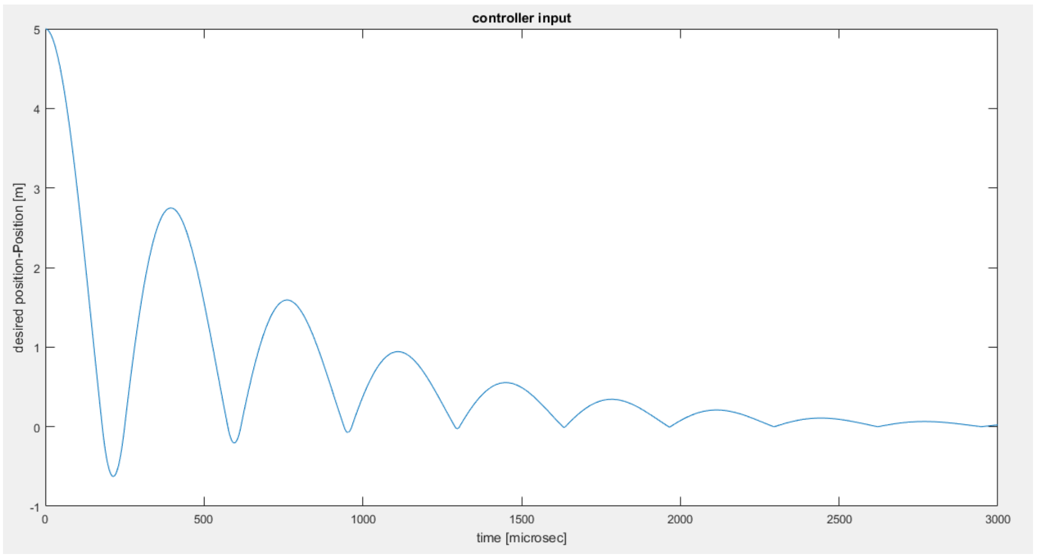

Figure 11 **Position Error Over Time (On-Off Controller)**

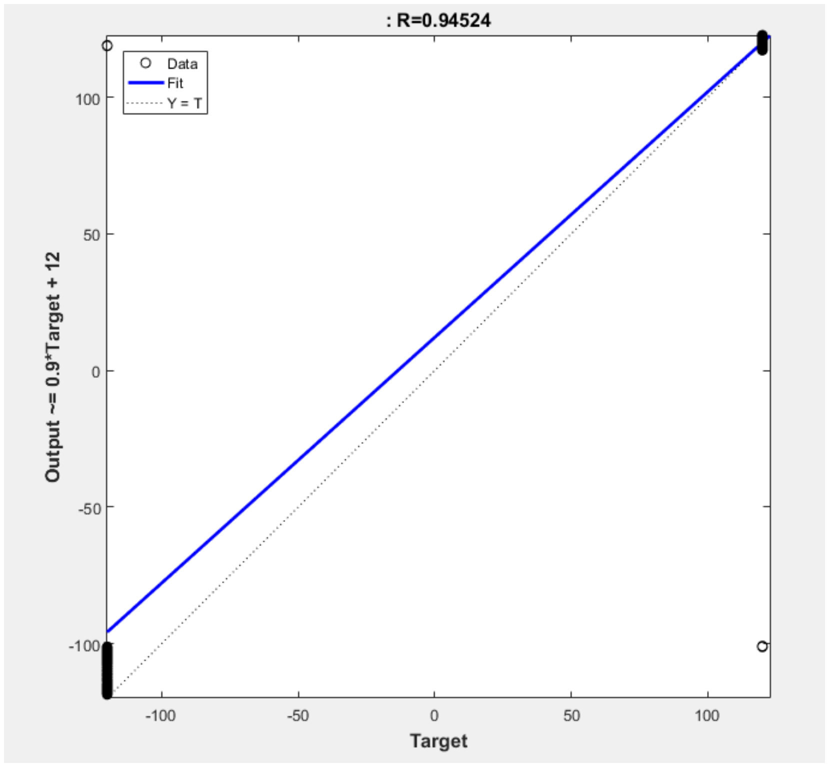

Figure 12 **Regression Graph for Neural Network Controller Output and Desired Output (Real Feedback)**

Mean squared errors (MSE) over epochs are shown in Fig. 18.

Finally, Fig. 19 compares the neural network output with the expected output. As seen, the neural network can closely follow the desired output.

Thus far, the neural network has successfully learned the On-Off controller.

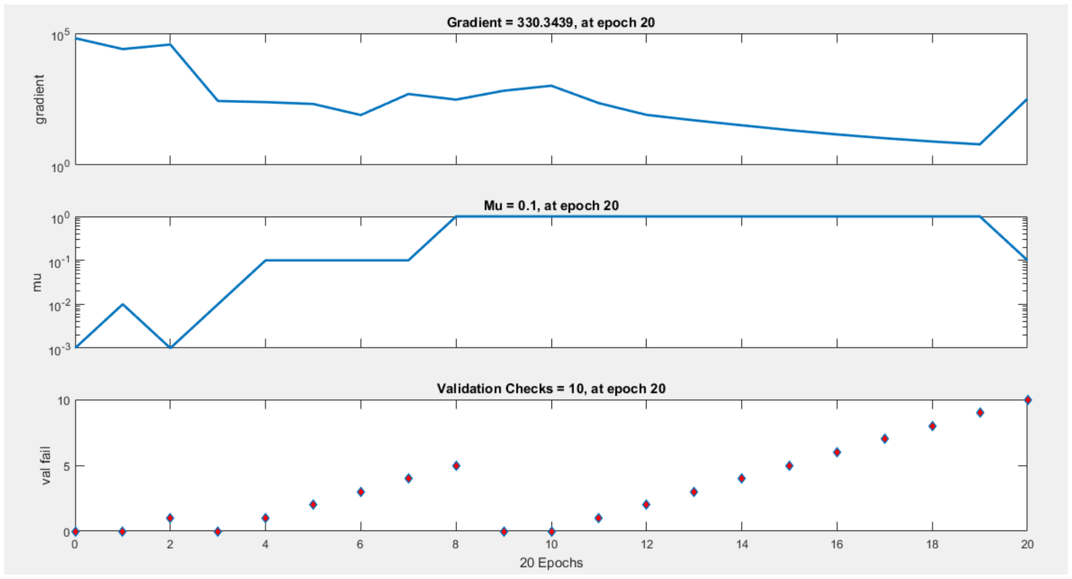

Figure 13 **Gradient, Momentum, and Validation Error Over Epochs During On-Off Controller Training (Real Feedback)**

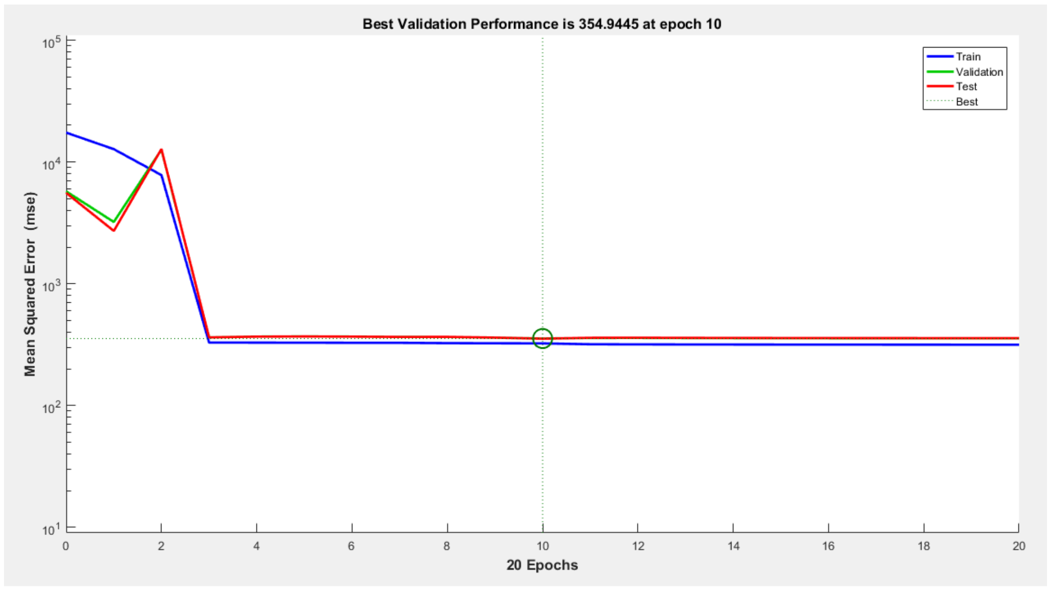

Figure 14 **Mean Squared Errors (MSE) Over Epochs During On-Off Controller Training (Real Feedback)**

3. Unsupervised Learning

Understanding unsupervised learning requires some prerequisites extracted from source [1], explained below. Consider the neural network structure shown in Fig. 20.

This network has no hidden layer. Assume n inputs,we have:

$$\begin{aligned} \mathbf{X} &= \{x_i\} \qquad \text{for } i = 0 : n-1 \\ \mathbf{W} &= \{w_i\} \qquad \text{for } i = 0 : n-1 \end{aligned} \tag{3}$$

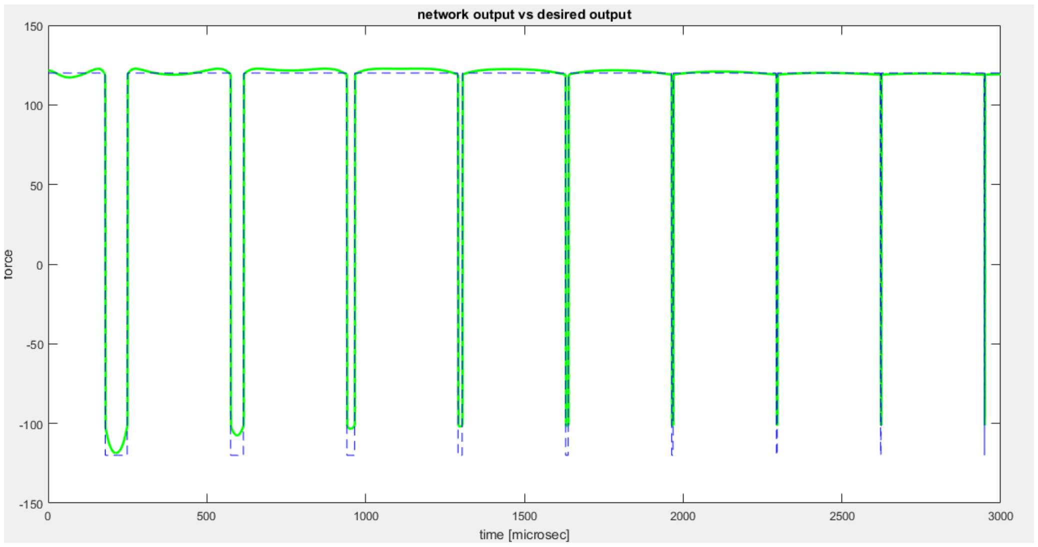

Figure 15 Neural Network Output vs. Expected Output (Real Feedback)

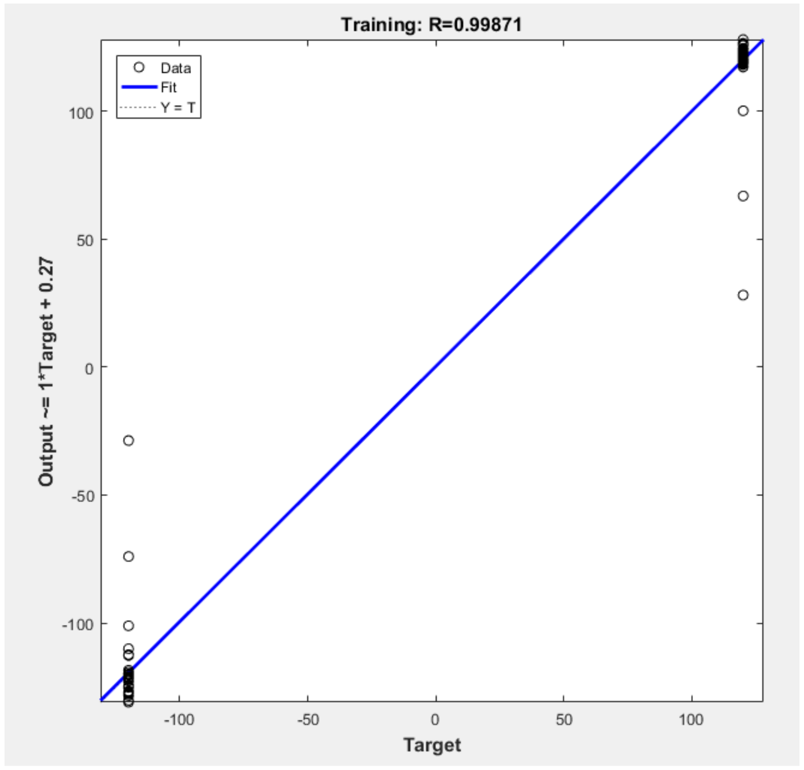

Figure 16 Regression Graph for Neural Network Controller Output and Desired Output (Network Feedback)

The network output y can be derived from the inputs $\mathbf{X}$ using Eq. (4), where f is a hyperbolic tangent sigmoid function.

$$s = \sum_{i=0}^{n-1} w_i x_i \qquad (4)$$
$$y = f\left(s\left(\mathbf{X}\right)\right)$$

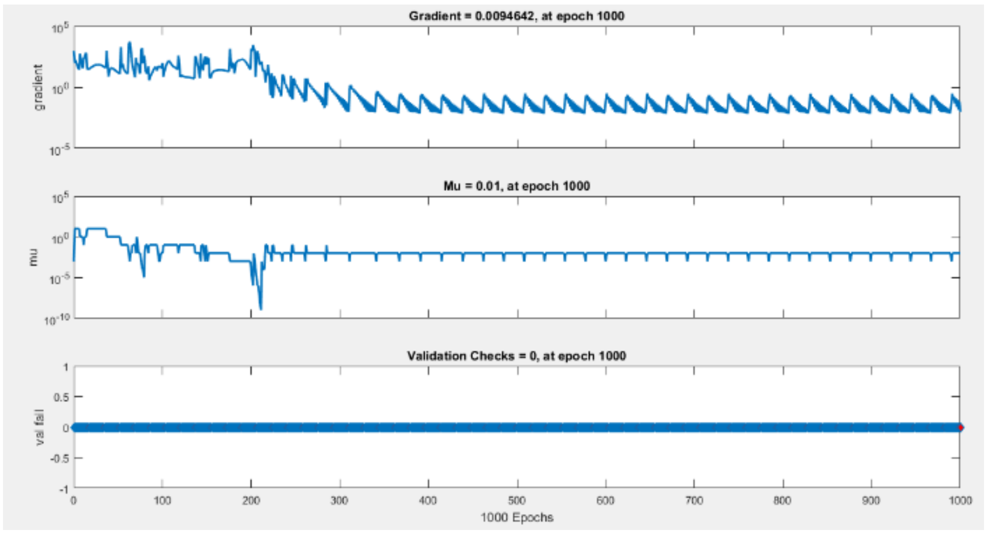

Figure 17 **Gradient, Momentum, and Validation Error Over Epochs During On-Off Controller Training (Network Feedback)**

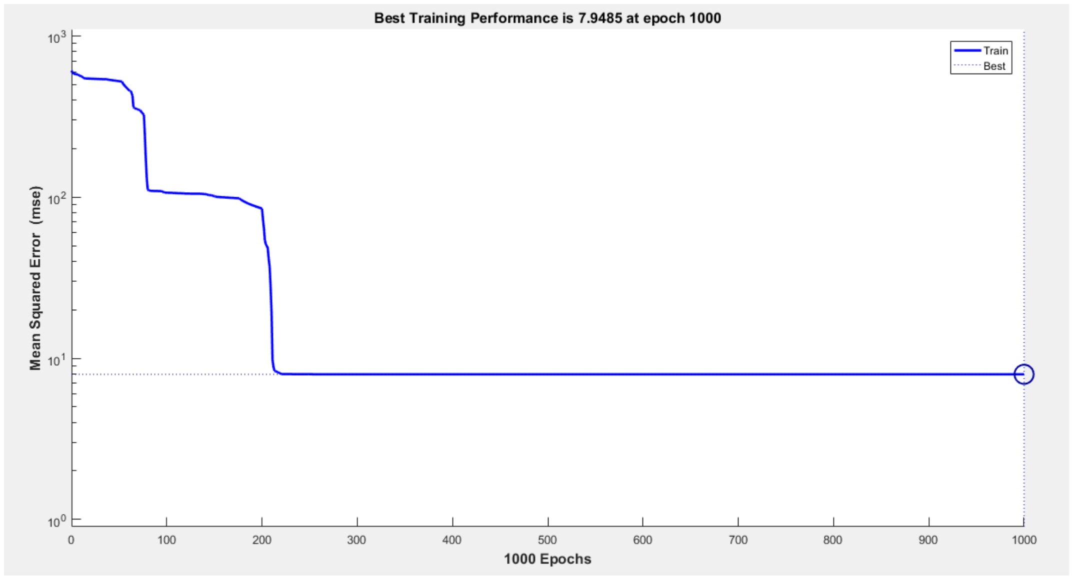

Figure 18 **Mean Squared Errors (MSE) Over Epochs During On-Off Controller Training (Network Feedback)**

Next, consider a neural network with one hidden layer, as shown in Fig. 21.

If the network inputs are $\mathbf{X}$, output is $y(\mathbf{X})$, and desired output for input $\mathbf{X}$ is $d(\mathbf{X})$, the error for an input $\mathbf{X}$ of size n is defined as follows:

$$J = \frac{\|(d(\mathbf{X}) - y(\mathbf{X}))\|^2}{n} \tag{5}$$

Consider the two layers shown in Fig. 21. The first layer is the hidden layer, and the second is the output layer. In each layer, the source neuron is indexed by m and the destination neuron by j. Thus, the weights for each layer are denoted as w_{jm}, where m is the source neuron index and j

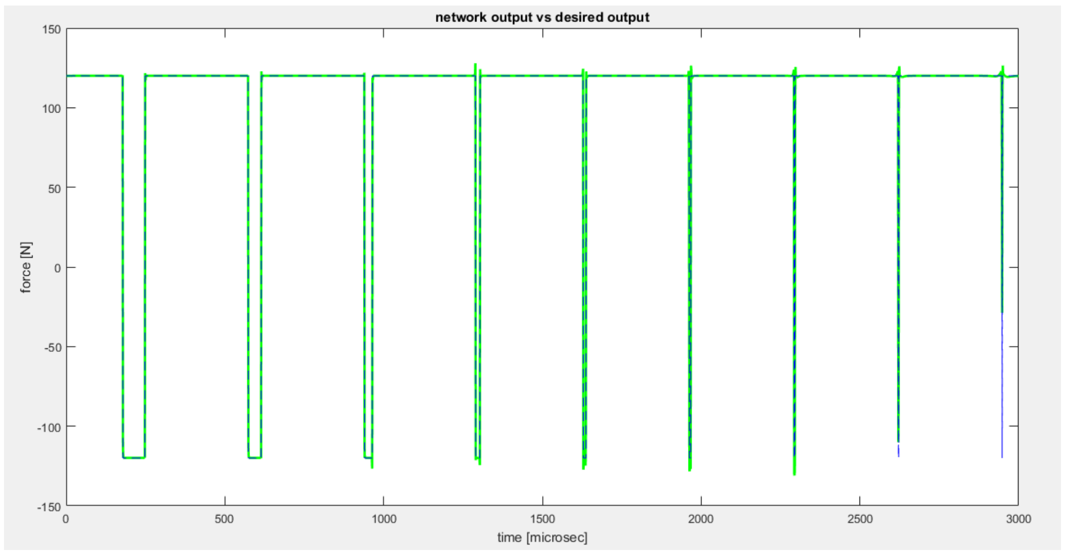

Figure 19 Neural Network Output vs. Expected Output (Network Feedback)

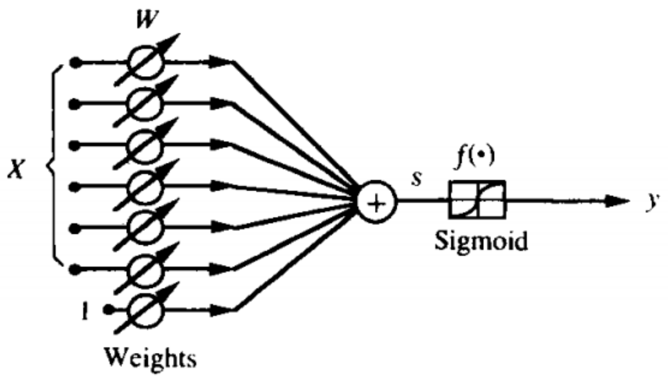

Figure 20 Simple Neural Network Without Hidden Layer [1]

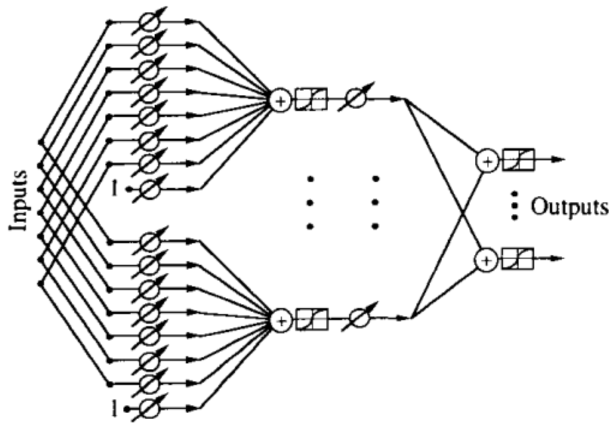

Figure 21 Simple Neural Network With Hidden Layer [1]

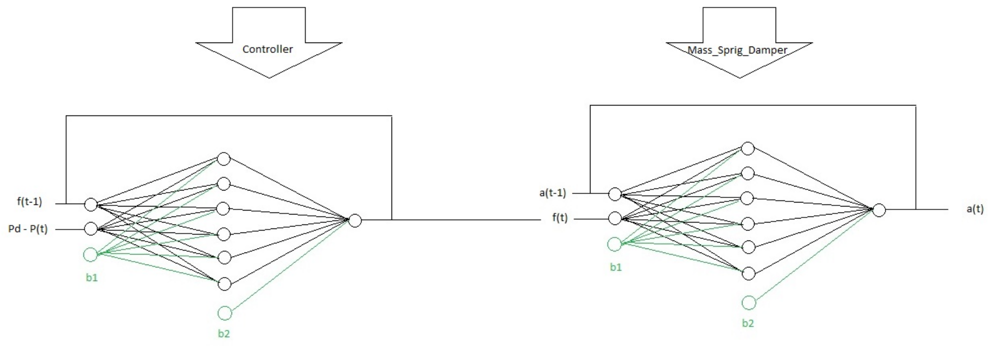

Figure 22 Overview of the Controller Neural Network and the Dynamic Model Neural Network

is the destination neuron index. For the output layer, the equivalent error δ is computed using Eq. (6):

$$\delta_m = \left(d_m\left(\mathbf{X}\right) - y_m\left(\mathbf{X}\right)\right).f'\left(s_m\left(\mathbf{X}\right)\right) \tag{6}$$

For the hidden layer, the equivalent error is computed using Eq. (7).

$$\delta_m = f'\left(s_m\left(\mathbf{X}\right)\right).\sum_j \delta_j w_{jm} \tag{7}$$

After calculating the equivalent error, the weights for each layer are updated using Eq. (8).

$$w_{im,new} = w_{im,old} + 2\mu\delta_m x_k \tag{8}$$

where i denotes the index of the source neuron.

With this introduction, we can explain the unsupervised training algorithm for the controller. Consider the neural network structure shown in Fig. 22

In Fig. 22, the neural network on the right models the mass-spring-damper system, and the neural network on the left models the controller. We aim for the controller to move the mass from position $p = 0$ to the desired position $p = p_d$, which in this project is set to $p_d = 5$m. The initial weights of the controller neural network, obtained in the previous section, provide a specific output for each given input. This output, which is the force, is applied to the system, guiding the mass-spring-damper system to a specific position. Therefore, with the initial (On-Off) controller, we have a specific position at each moment. Given that the sampling time is 0.01 seconds and the total simulation time is 30 seconds, we have 3000 positions, denoted by p_k, where k ranges from 0 to 3000. The controller's error is defined as follows:

$$J = \frac{\left\|p_d - p_k\right\|^2}{K} \tag{9}$$

where K is the number of samples, which is 3000 in this case. Eq. (9) represents the mean squared error between the positions obtained from the initial controller and the desired position ($p_d = 5$m). The goal is to update the weights of the controller neural network such that the defined J is minimized. For this, we need to calculate the force error that caused this position error.

The unsupervised training algorithm proceeds as follows: The error defined in Eq. (9) is back-propagated through the layers of the mass-spring-damper neural network (the right one) using Eq. (6) and Eq. (7) to obtain the force error. This force error corresponds to the position error. In other words, this force error caused the position error. At this stage, the weights of the mass-spring-damper neural network are not updated, and error back-propagation is only used to obtain the equivalent force error. This error is the output error of the controller neural network. Now, this error can be back-propagated through the layers of the controller neural network using Eq. (6) and Eq. (7), and the weights of the controller neural network can be updated using Eq. (8). This process continues in a loop until the defined J is minimized.

The controller neural network was trained using the described method. As mentioned, unsupervised training continues in a loop until the defined J is minimized. Unsupervised training also occurs in two phases. In the first phase, the force feedback is disconnected, and the actual force at time $t-1$ is fed to the network. In the second phase, the force feedback is taken from the neural network output. The training performance of the first phase (with real feedback) and the training characteristics are shown in Figures 23 to 25.

As observed, the R-value is 1, indicating that the neural network output perfectly aligns with the desired output. The gradient, momentum, validation error, and mean squared errors (MSE) are shown in Fig. 24 and Fig. 25, respectively.

The training performance of the second phase (with network feedback) and the training characteristics are shown in Figures 26 to 28.

As observed, this network is also well-trained. Given that the first training phase was well-executed, it is natural for the R-value to be 1 in this phase as well. The gradient, momentum, validation error, and mean squared errors (MSE) are shown in Fig. 27 and Fig. 28, respectively.

The weights and bias values of the trained neural network are shown in Eq. (10).

$$\mathbf{W}_1 = \begin{bmatrix} 5.5619 & -3.5087 \\ 1.39\times10^{-6} & -0.0376 \\ 284.99 & 284.13 \\ 270.88 & 298.61 \\ -359.39 & -1.3559 \\ -14561 & -1.5652 \end{bmatrix} \tag{10}$$

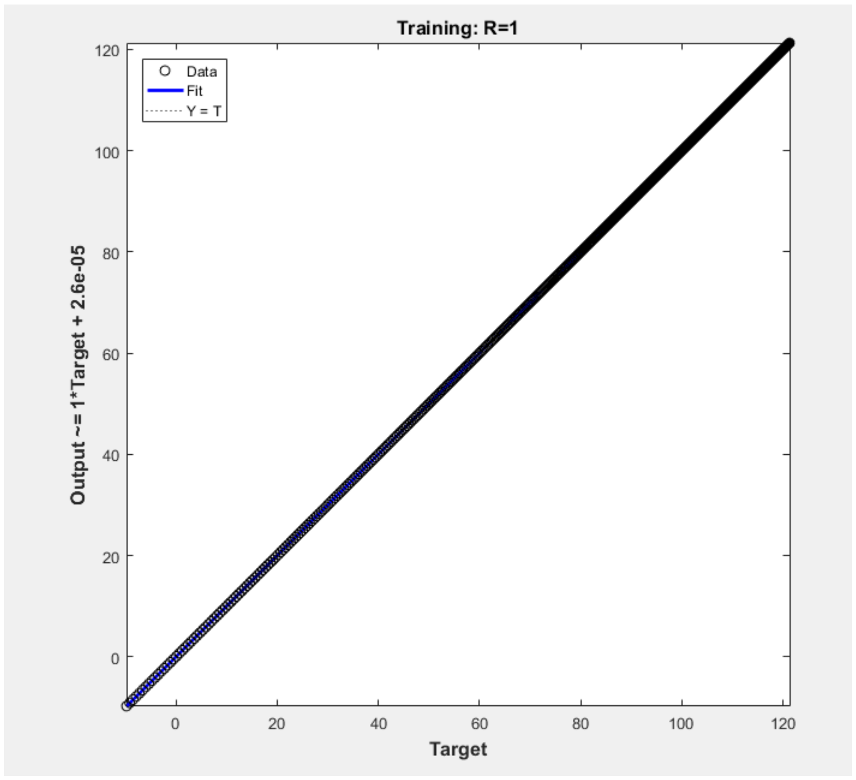

Figure 23 Regression Graph for Final Controller Neural Network Output and Desired Outputs (Real Feedback)

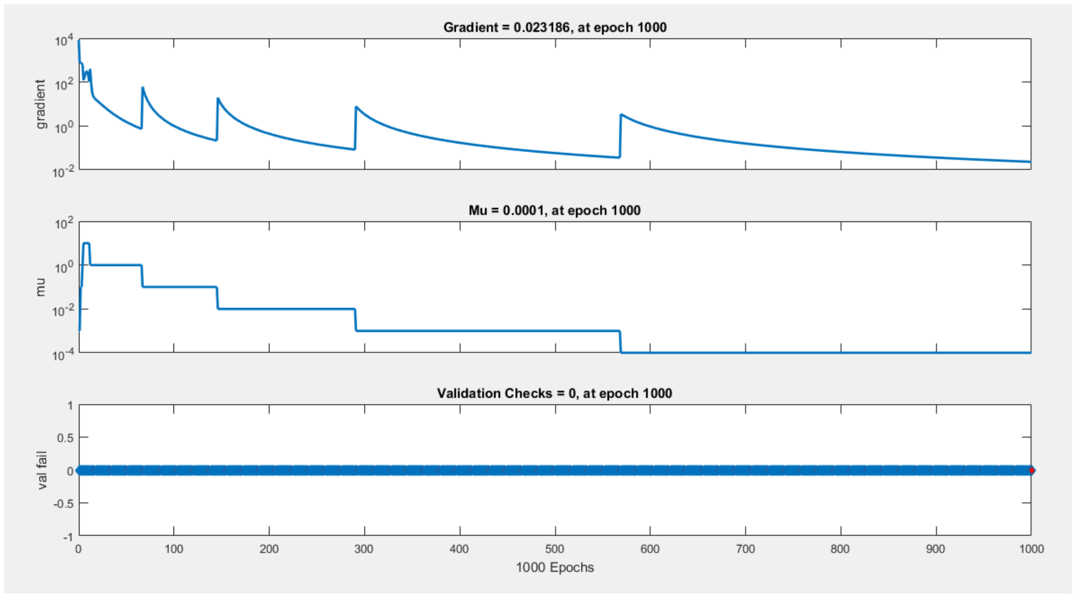

Figure 24 Gradient, Momentum, and Validation Error vs. Epochs During Final Controller Training (Real Feedback)

$$\mathbf{W}_2 = \begin{bmatrix} 0.43779 \\ -26.427 \\ 6.0653 \times 10^{-5} \\ -5.387 \times 10^{-5} \\ -6.249 \times 10^{-5} \\ 4.934 \times 10^{-5} \end{bmatrix}$$

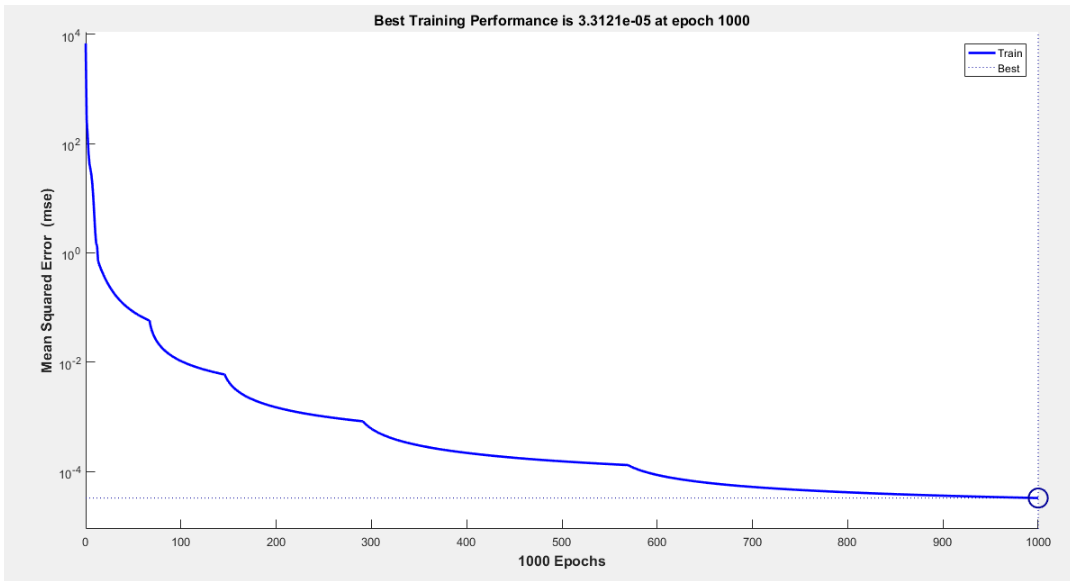

Figure 25 Mean Squared Errors (MSE) vs. Epochs During Final Controller Training (Real Feedback)

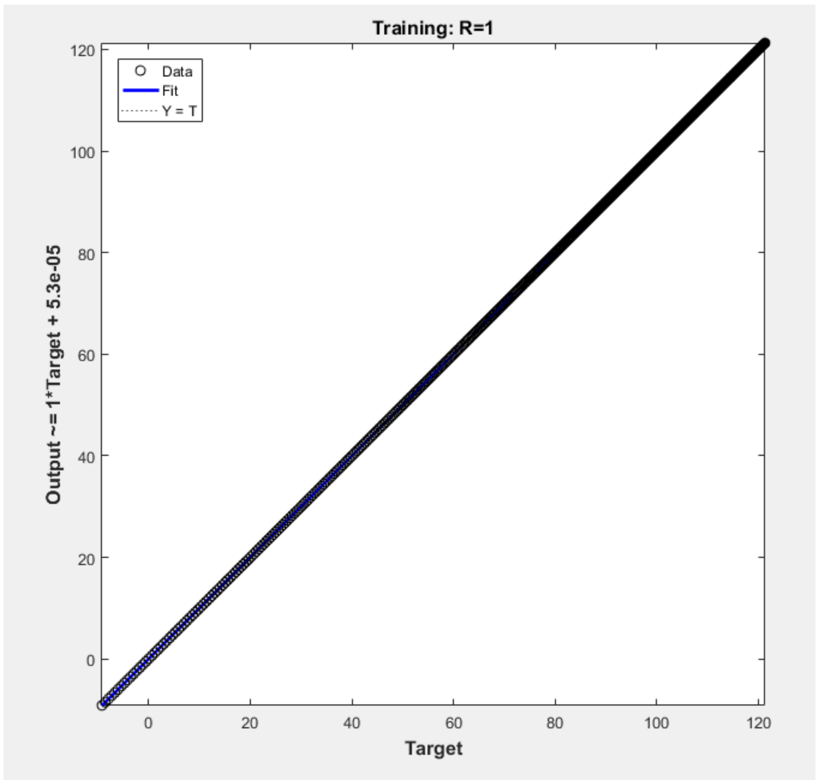

Figure 26 Regression Graph for Final Controller Neural Network Output and Desired Outputs (Network Feedback)

$$b_1 = \begin{bmatrix} -14.293 \\ 0.016377 \\ -78.707 \\ -84.796 \\ -284.7 \\ -11328 \end{bmatrix}$$

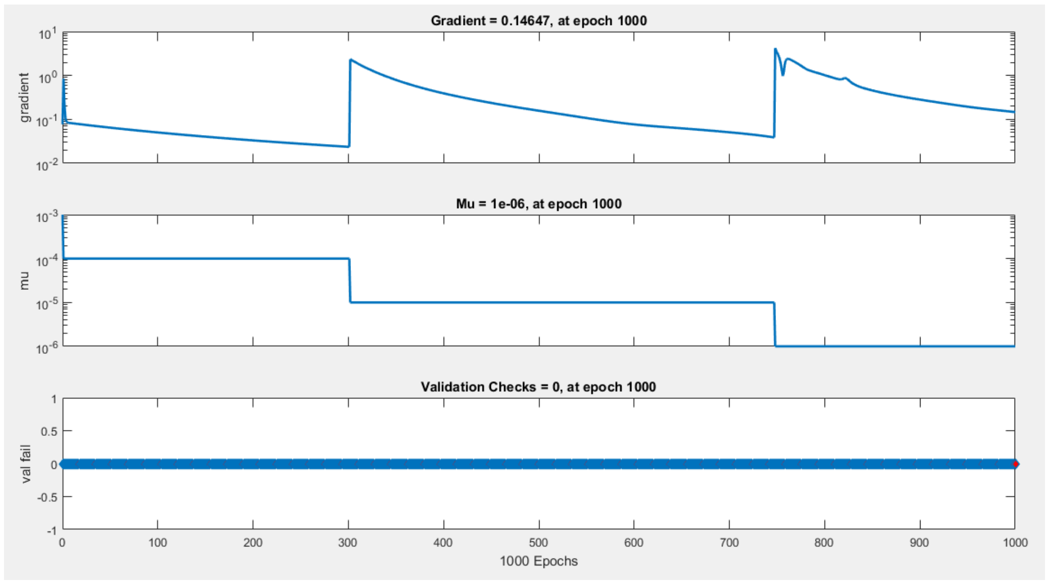

Figure 27 **Gradient, Momentum, and Validation Error vs. Epochs During Final Controller Training (Network Feedback)**

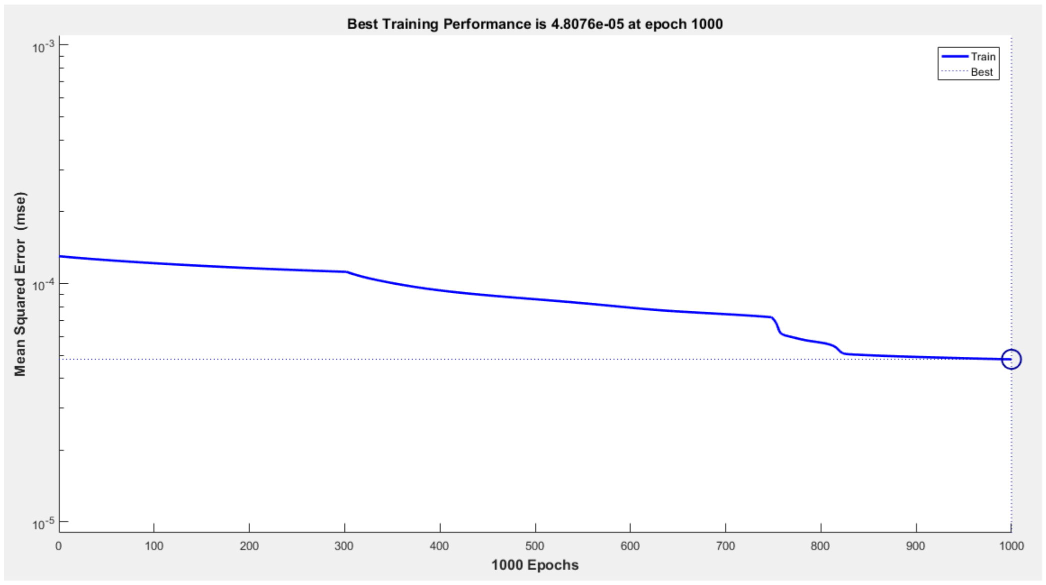

Figure 28 **Mean Squared Errors (MSE) vs. Epochs During Final Controller Training (Network Feedback)**

$$b_2 = 0.87549$$

Figures 29 to 32 show the position, velocity, acceleration, and force of the system before and after unsupervised training.

4. Conclusion

In this project, an attempt was made to train a neural network as a controller for the mass-spring-damper system designed in the previous project. Initially, to eliminate random initial weights and

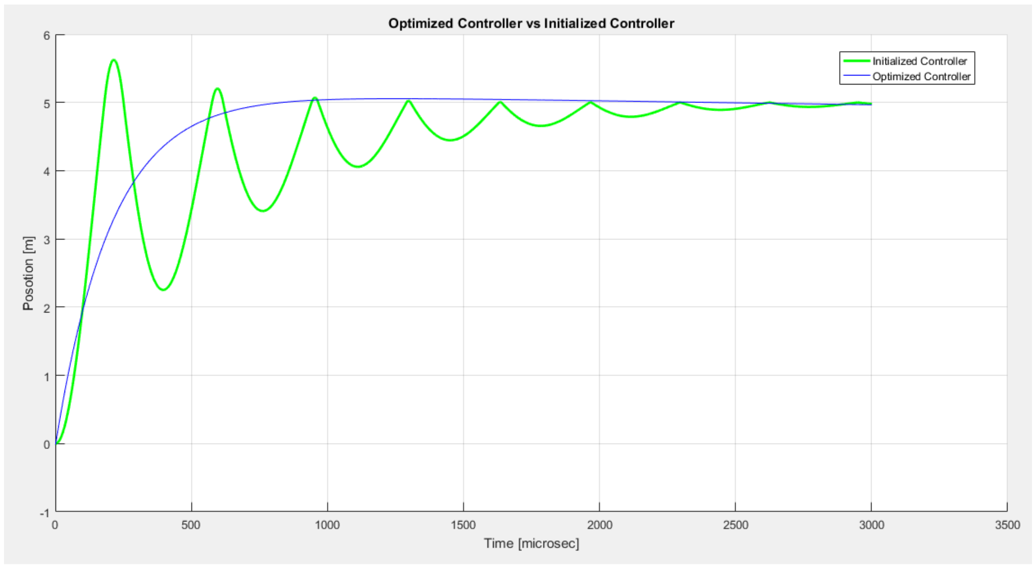

Figure 29 **Position vs. Time Before and After Unsupervised Training**

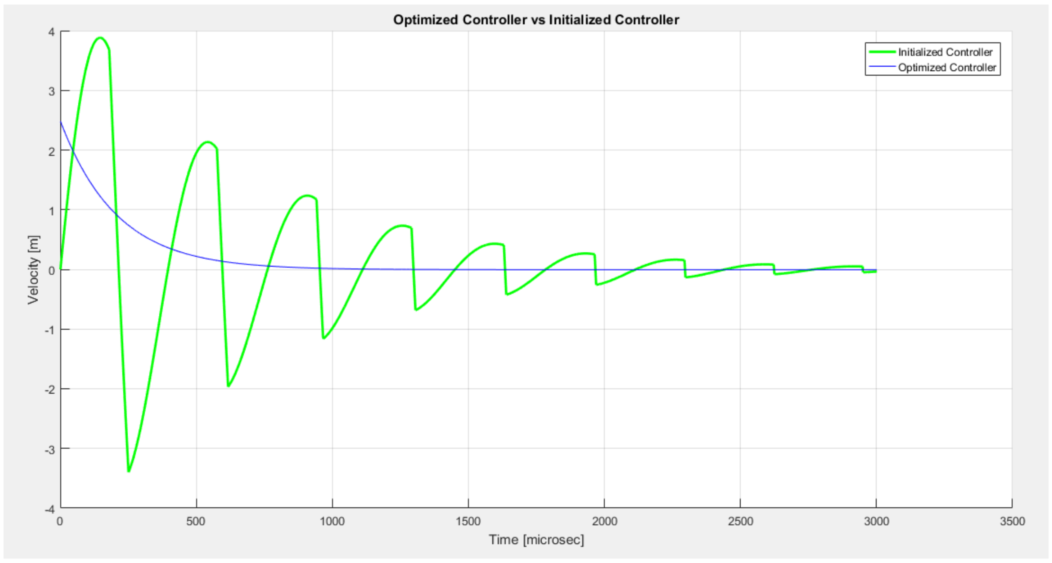

Figure 30 **Velocity vs. Time Before and After Unsupervised Training**

increase training speed, a simple On-Off controller was simulated in MATLAB/Simulink. After extracting the input and output data, this data was used for supervised training of the neural network. Subsequently, unsupervised learning was performed on the neural network. The position error at each moment compared to the desired position was back-propagated through the mass-spring-damper neural network model to obtain the equivalent force error at each moment. These errors were then used with Equations (6), (7), and (8) to update the neural network weights. The neural network was trained based on this algorithm, and satisfactory results were achieved.

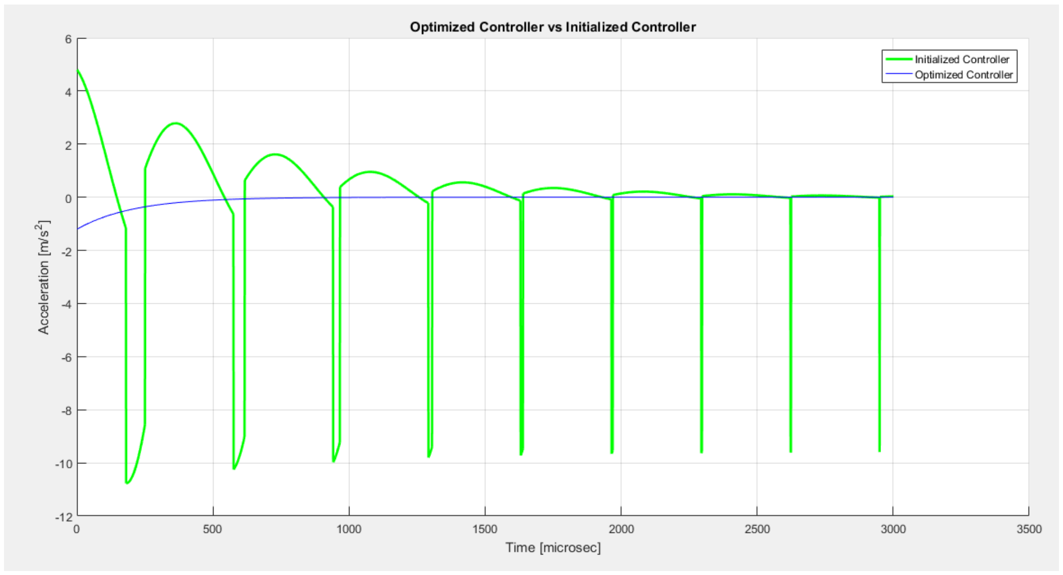

Figure 31 **Acceleration vs. Time Before and After Unsupervised Training**

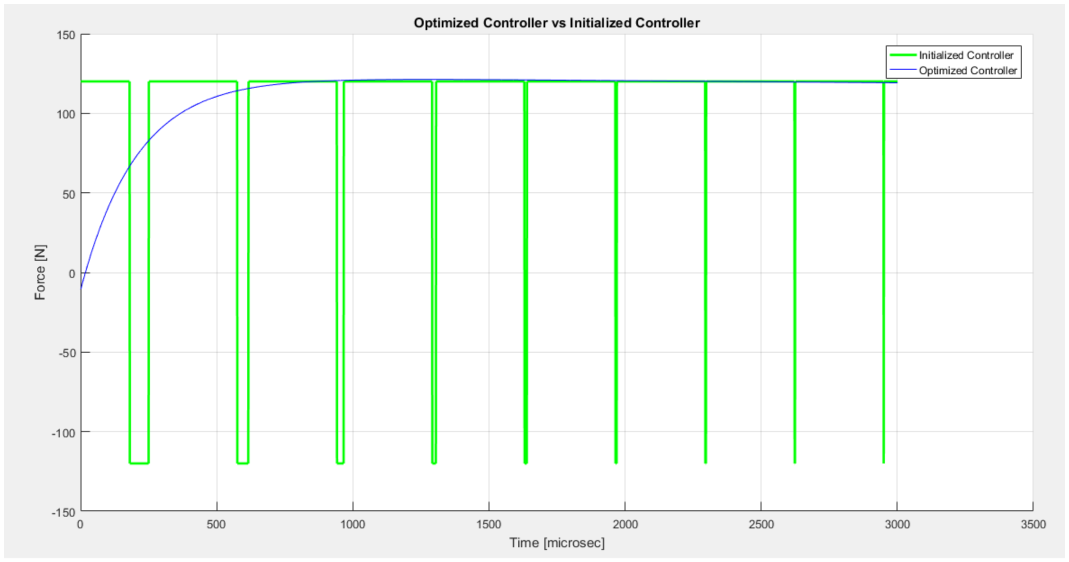

Figure 32 **Force vs. Time Before and After Unsupervised Training**

References

[1] D. Nguyen and B. Widrow, "Neural networks for self-learning control systems," *IEEE Control Systems Magazine*, vol. 10, no. 3, pp. 18–23, 1990.